Translational Expressiveness

Comparing Process Calculi using Encodings

vorgelegt von
Dipl.-Inform.
Kirstin Peters
aus Potsdam

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzende:	Prof. Dr. S. Glesner
Gutachter:	Prof. Dr. U. Nestmann
Gutachter:	Prof. Dr. J. Parrow
Gutachter:	Prof. Dr. D. Gorla

Tag der wissenschaftlichen Aussprache: 19. September 2012

Berlin 2012
D 83

Acknowledgements

First of all let me express my thanks to my supervisor Prof. Uwe Nestmann. I very much enjoyed my time as a PhD student in Berlin. It was Uwe who got me interested in concurrency theory and process calculi. I learned a lot from the countless discussions with him and within our group, and his door was open whenever I had a problem. Moreover, Uwe gave me the opportunity to do a lot of teaching—a part of my job I always loved and enjoyed.

Of course there were also a number of other researchers that supported me with fruitful discussions and fresh ideas. In particular let me thank the members of our research project on synchronous and asynchronous interactions in distributed systems: Prof. Ursula Goltz, Jens-Wolfhard Schicke-Uffmann, and Stephan Mennicke from Braunschweig, Prof. Rob van Glabbeek from Sydney, and Prof. Uwe Nestmann and Christian Hammerschmidt from Berlin. Most of the ideas of this thesis were developed during my work in this project and the discussions at our regular meetings. I also benefited a lot from the small project workshop with Prof. Rob van Glabbeek and Prof. Daniele Gorla in Berlin.

Apart from that I express my thanks to the group of german researchers in concurrency theory that meet every spring to exchange ideas and providing a platform for collaboration—in particular between the PhD students. The informal and familiar atmosphere at these meetings allowed me to present some of my ideas at an early stage to an interested and candid audience. The discussions improved not only my style of presentation but also led to new insights and ideas for ongoing work.

I would also like to thank Prof. Joachim Parrow, Prof. Björn Victor, Ioana Rodhe, Johannes Borgström, Johannes Åman Pohjola, Palle Raabjerg, Tjark Weber, Ramunas Gutkovas, and Alasdair Armstrong for the opportunity to visit Sweden and for the very nice time I had in this group. Since these visits took part at the end and after finishing the thesis, they might not have influenced the present document that much, but I am sure that they will influence my future research.

I also thank Prof. Uwe Nestmann, Prof. Joachim Parrow, and Prof. Daniele Gorla for taking the job to referee my thesis. Furthermore I would like to thank Margit Russ for all the help related to the non-scientific parts of my time as a PhD student.

Last but not least I express my special thanks to Christoph Wagner for bearing me in stressed times, cheering me up whenever I needed it, improving my latex code, and everything else he has done for me.

Summary

We study the relation between process calculi—in particular between variants of the pi-calculus and the join-calculus—that differ in their either *synchronous* or *asynchronous interaction* mechanism. Synchronous and asynchronous interactions are the two basic paradigms of interactions in distributed systems. While synchronous interaction is widely used in specification languages, asynchronous interaction is often better suited to implement real systems. We are interested in the conditions under which synchronous interactions can be implemented using just asynchronous interactions. We compare the different variants of the calculi with respect to their expressive power. To do so we analyse the existence of *encodings* between the languages, i.e., translations of the processes of one language into processes of another language. In particular we are interested in encodings that *preserve distributability*, i.e., in translations that to not reduce the degree of concurrency of the translated processes. We discuss positive as well as negative results between synchronous and asynchronous variants of the pi-calculus and the join-calculus.

Zusammenfassung

Wir untersuchen die Beziehung zwischen Prozesskalkülen – insbesondere zwischen verschiedenen Varianten des Pi-Kalküls und dem Join-Kalkül – die sich in den verwendeten Interaktionsmechanismen unterscheiden. Dabei unterscheiden wir im Wesentlichen zwischen *synchronen* und *asynchronen Interaktionsmechanismen*, als Basisformen von Interaktion in verteilten Systemen. Aufgrund ihrer größeren Ausdrucksstärke werden synchrone Interaktionsmechanismen oft in Spezifikationen benutzt, asynchrone Interaktionsmechanismen lassen sich in der Regel aber leichter in realen Systemen implementieren. Wir untersuchen unter welchen Bedingungen eine Abbildung synchroner Interaktionen in asynchrone Interaktionen möglich ist. Dazu vergleichen wir die Ausdrucksstärke verschiedener Varianten von Prozesskalkülen, indem wir untersuchen, ob zwischen diesen Sprachen eine *Kodierung* existieren kann. Besonders interessieren wir uns für die Möglichkeit einer Kodierung, welche den Grad der *Verteilbarkeit* der zu übersetzenden Prozesse bewahrt. Wir diskutieren verschiedene Resultate sowohl für die Möglichkeit als auch für die Unmöglichkeit einer solchen Kodierung zwischen Varianten des Pi-Kalküls und dem Join-Kalkül.

Contents

Brief Outline	
Chapter 1 Introduction	• Introduction to Translational Expressiveness • Overview over Translational Results • Overview over Results Comparing Synchronous and Asynchronous Interactions in the Pi-Calculus
Chapter 2 Process Calculi	• Introduction to Process Calculi • Variants of the Pi-Calculus: ◦ π_{m}: Synchronous Pi-Calculus with Mixed Choice ◦ π_{s}: Synchronous Pi-Calculus with Separate Choice ◦ π_{a}: Asynchronous Pi-Calculus (without Choice) ◦ π_{p}: Asynchronous Pi-Calculus with Polyadic Synchronisation • Introduction to the Join-Calculus (J) and CSP • Bisimulation and Observables in the Pi-Calculus
Chapter 3 Encodings and their Quality Criteria	• General Framework of Quality Criteria: ◦ Compositionality, Name Invariance, ◦ Operational Correspondence, ◦ Divergence Reflection, Success Sensitiveness • Domain-Specific Criterion: ◦ Preservation of Distributability
Chapter 4 Separating Languages	• Breaking Symmetries • No Good Encoding from π_{m} into π_{s}: ◦ Translates the Parallel Operator Homomorphically ◦ Preserves Distributability ◦ Reflects Causal Dependencies • No Good Encoding from π_{a} into J Preserves Distributability • Synchronisation Patterns: M and $\star$
Chapter 5 The Design of Encodings	• Encoding Functions: ◦ $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$: from π_{s} into π_{a} of [Nes00] ◦ $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$: from π_{m} into π_{p} ◦ $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$: from π_{m} into $\pi^{=}_{\mathrm{a}}$ • Unfolding of Polyadic Communications
Chapter 6 Properties of Encodings	• Correctness of the Encodings • Properties of the Encodings: ◦ Administrative, Impure, and Core Steps ◦ Translated Observables ◦ (un)observable, (in)active Junk ◦ Preservation of Distributability
Chapter 7 Concluding Remarks	• Summary of the Main Contributions • Hierarchy on Distributability in Pi-like Calculi • Further Research

1. Introduction

In today's world of wireless and mobile networking, parallel and distributed systems are omnipresent. Their analysis and verification, however, is still a challenging task. A distributed system is a system whose components are assumed to reside on different locations. In particular we assume that the components do not share a common clock. Different components of such a system can either interact or act concurrently. The interplay of interaction and concurrency in distributed concurrent systems results in an awfully large state space even of reasonable small systems. Interactions are either synchronous, i.e., interaction may happen immediately as atomic or simultaneous exchange of information, or asynchronously, i.e., interaction may take time. While synchronous interactions are widely used in specification languages, asynchronous interactions are often better suited to implement real systems. The context of this thesis is the formal analysis of both the expressive power and the (distributed) implementability of specification and programming languages for concurrent systems, by the study of synchronous and asynchronous interaction mechanisms. Concentrating on the computational essence of such languages, we focus on so-called process calculi—as exemplified by the pi-calculus—which contain as few syntactic primitives as possible.

Process calculi (or process algebra) is one area of formalisations of concurrent systems. Other areas are for example Petri nets [Pet62] or the Actor model [HBS73]. *A brief history of process algebra* can be found in [Bae05]. Over the time different process calculi emerge. Examples are CCS [Mil89], CSP [Hoa78], or ACP [BK82], just to name some of the most prominent ones. The pi-calculus [MPW92, Mil99] has become more recently prominent as a process calculus to reason about mobile systems. Note that each of these calculi denotes rather a family of process calculi. The number of different process calculi is in fact enormous. As discussed in [Nes06] there are many good reasons for this great number of different calculi. The most plausible is maybe that many of these different calculi stem from different practical needs. They are designed as domain-specific calculi capturing exactly the set of primitives necessary to model the desired system at a proper level of abstraction without overloading the theory with (for this domain) unnecessary operations. The large number of calculi calls for methods to compare different calculi or different variants within a family of process calculi with respect to their expressive power. The most prominent such method is language embedding using encodings [BP91]. Encodings—or the proof of their absence—do not only allow to compare the expressive power of languages but also formalise similarities and differences between the considered languages. So they provide a base for implementations of languages into real systems.

Accordingly, we discuss how languages can be compared by means of encodings in general and in particular how the synchronous and asynchronous variants of the pi-calculus are related.

1.1. Organisation of the Thesis

In principle there are two ways to read this thesis. As the title suggests, the common thread of this thesis is to provide an overview on the derivation of translational results, i.e., results that compare the expressive power of process calculi by means of encodings. As running example throughout the chapters we compare synchronous and asynchronous variants of the pi-calculus. In fact the main contributions of this work are the positive and negative results obtained from the consideration of this running example, i.e., from the comparison of different variants of the pi-calculus with respect to synchronous and asynchronous interaction mechanisms.

1.1.1. Translational Expressiveness

There are basically two ways to measure the expressive power of a language [Par08].

An *absolute result* is a result about the expressive power of a single process calculus, usually obtained by proving the ability (*positive absolute result*) or inability (*negative absolute result*) to solve some kind of problem (see [Par08, Gor10b] and even [LSZ74]), whereas a relative result compares two process calculi. However, the "absolute" of the first kind of results does not necessarily mean that they cannot be used to derive relative results. Indeed all absolute results presented in this thesis are used to compare two or more different process calculi. For us, the "absolute" rather refers to the proof of the respective result. More precisely, throughout this thesis we refer to all expressivity results that are derived for a single language (without any reference to another language) as absolute result.

Now, to *compare* the absolute expressive power of *two* languages, we may simply choose a problem that can be solved in one language, but not in the other language. Note that combining two absolute results that are both positive or both negative usually does not reveal much information, because it proves only that the considered languages do not differ with respect to the respective particular problem. Actually as soon as we compare two languages, it makes sense to use the term *relative expressive power*, as we can now relate the two languages. Unfortunately the terminology was introduced differently. It has been attributed (see [Par08, Gor10b]) to the comparison of the expressive power of two languages by means of the existence or non-existence of encodings from one language into the other language, subject to various conditions on the encoding. In our opinion, the term "relative expressive power" is misleading. First, as mentioned above, also the absolute expressive power can directly be used to *relate* two languages. Second, results on the encodability of a language have to be understood relative to the specific conditions on the encoding—it is not always clear to what aspect the "relative" refers. Thus, we prefer the notion of *translational expressive power* to refer to comparisons of the expressiveness of two languages by analysing the existence or non-existence of an encoding, subject to various conditions. A positive translational result, i.e., the proof of the existence of an encoding, is denoted as *encodability result*, whereas a negative translational result is denoted as *translational separation result*.

Chapter 2 introduces (the variants of) the process calculi that are considered within

this thesis as well as bisimulation as a technique to reason about the behaviour of process terms. As mentioned translational results are always subject to some conditions, denoted as *quality criteria* in the following. To require that translational results satisfy some quality criteria is necessary, to rule out trivial and meaningless encodings. In Chapter 3 we formally define encoding functions and discuss possible sets of quality criteria. Then we introduce a general framework of [Gor08b, Gor10b] which allows us to obtain reasonable and comparable translational results. Moreover, we discuss the extension of this framework by an additional domain-specific criterion in order to answer questions that go beyond the general expressive power of process calculi.

Negative translational results are discussed in Chapter 4. Thereby we show the relevance of absolute results for the derivation of translational separation results, discuss different methods to obtain absolute results, and analyse what conditions turn an absolute result suitable for the derivation of a translational separation result in a particular setting of quality criteria. Then, we show separation with respect to varying sets of quality criteria including different general criteria as they occur in the general framework as well as domain-specific criteria. We observe that translational separation results that are based on absolute results basically follow the same line of argument (at least if the absolute result is well-suited for the required set of quality criteria) and that sometimes several negative translational results with respect to different sets of quality criteria or even with respect to different languages can be derived from the same absolute result or comparable instances of the same absolute result. Moreover, we discuss how translational separation results can be transferred or adapted to separate other pairs of languages.

Chapter 5 discusses the design of encoding functions, i.e., positive translational results. For this we distinguish between the main idea of an encoding, denoted as its *concept*, and the *implementation* of this concept in terms of the target language. Then we show, as an example, how the concept and the implementation can be extended to obtain an encoding that covers a more expressive source language. To improve the presentation of the resulting rather complex encoding, we make use of an intermediate encoding which allows us to abstract from technical details.

How to prove that an encoding function satisfies a given set of criteria is discussed in Chapter 6. Here, we mainly focus on the structural and semantic criteria that form the general framework presented in Chapter 3, but consider also the prove of a domain-specific condition that is introduced in Chapter 3. Moreover, we introduce some type systems for the target languages of the presented encodings and discuss their use in the proof of the semantic criteria. We also use invariants as poof technique. Furthermore, we talk about typical properties of encodings. We analyse the kinds of steps that may be performed by an encoded process term in order to simulate some behaviour and discuss how a classification of such steps can guide the proof of semantic properties. We also discuss how observables are translated by an encoding function and how this influences the proof of correctness of this encoding and the set of criteria that can be satisfied by the respective encoding. Also the kinds of junk or garbage that are introduced by an encoding may influence the set of criteria it can satisfy. More precisely, the proof of correctness of an encoding has to ensure that the introduced junk does no harm with respect to the considered quality criteria.

Finally, Chapter 7 concludes by showing how translational results can be combined to obtain a hierarchy.

1.1.2. Synchrony and Asynchrony in the Pi-Calculus

We study the relation between process calculi that differ in their either synchronous or asynchronous interaction mechanism. Synchronous and asynchronous interactions are the two basic paradigms of interactions in distributed systems. While synchronous interaction is widely used in specification languages, asynchronous interaction is often better suited to implement real systems. We are interested in the conditions under which synchronous interactions can be implemented using just asynchronous interactions, in particular in the conditions under which it is possible to encode the synchronous pi-calculus into its asynchronous variant. Within this thesis we derive positive as well as negative translational results to answer this question.

A discussion on synchrony versus asynchrony cannot be separated from a discussion of choice. When processes communicate via message-passing along channels, they do not only listen to one channel at a time—they *concurrently listen* to a whole selection of channels. Choice operators just make this natural intuition explicit. Moreover, their mutual exclusion property allows us to concisely describe the particular effect of message-passing actions on the process's local state. Asynchronous send actions make no sense as part of a mutually exclusive selection, as they cannot be prevented from happening. Consequently, the asynchronous calculus only offers input-guarded choice or no choice at all. In contrast synchronous send actions also allow for the definition of mixed choice, i.e., selections of both input and output actions.

It is well known that there is a good encoding from the choice-free synchronous pi-calculus into its asynchronous variant [Bou92, HT91, Hon92a]. It is also well-known [Pal03, Gor10b] that there is no good encoding from the full pi-calculus—the synchronous pi-calculus including mixed choice—into its asynchronous variant if the encoding translates the parallel operator homomorphically. Palamidessi was the first to point out that mixed choice strictly raises the absolute expressive power of the synchronous pi-calculus compared to its asynchronous variant. The proof of this result analyses their different expressive power concerning leader election in symmetric networks. More precisely, Palamidessi proves that there is no symmetric network in the asynchronous pi-calculus that solves leader election, whereas there are such networks in the synchronous pi-calculus. The proof implicitly uses the fact that it is not possible in the asynchronous pi-calculus to break initial symmetries, while this is possible in the synchronous pi-calculus. To this end, a rather strong notion of symmetry consisting of a syntactic and a semantic component is used to ensure that solving leader election requires breaking initial symmetries. Later on, Gorla offered an arguably simpler proof that, instead of leader election in symmetric networks, employed the reducibility of "incestual" processes (mixed choices that include both enabled senders and receivers for the same channel) when running two copies in parallel. In both proofs the role of *breaking (initial) symmetries* is more or less apparent. In Section 4.1.3 we shed more light on this role by re-proving the absolute result in [Pal03]—based on a proper formalisation of what it means to

break symmetries—without referring to another problem domain like leader election. In Section 4.2.1 we use this result to reprove the translational separation results of [Pal03] and (the first setting of) [Gor10b], i.e., show that there cannot exist a reasonable encoding that translates the parallel operator homomorphically.

As already Gorla [Gor10b] states, the condition of homomorphic translation of the parallel operator is rather strict. Therefore Gorla proposes the weaker criterion of compositional translation of the source language operators. In Chapter 5 we discuss how this weakening of the structural condition on the encoding of the parallel operator turns the above separation results into an encodability result, i.e., there is an encoding from the synchronous π-calculus—including mixed choice—into its asynchronous variant with respect to the general framework of Gorla. Note that this general framework defines a set of minimal criteria for encodings: they must be compositional and preserve and reflect computations, deadlocks, divergence, and success. In Chapter 6 we show that all these conditions are satisfied by the proposed encoding of mixed choice. Moreover, we discuss different properties and also drawbacks of the proposed encoding in Chapter 5 and Chapter 6.

The homomorphic translation of the parallel operator was used in [Pal03], because encodings that preserve this criterion are ensured to *preserve distributability*. If we consider the implementability of languages in real distributed systems, this requirement is of great importance. Implementations that force programs to synchronise on each step or to reside on a single location turn synchronous and distributed specifications meaningless. Hence we are interested in translations that preserve the degree of distribution of the encoded processes. However, as we discuss in Section 3.4, the homomorphic translation of the parallel operator is too strict as criterion for separation results. Instead we propose a novel criterion and explain why it is better suited to measure preservation of distributability. In Section 4.2.2 we prove then that also the resulting weaker setting does not allow to encode mixed choice, i.e., we show that there exists no encoding that respects the criteria in the general framework of Gorla and also preserves distributability.

In Section 4.4 we improve this separation result by a more intuitive proof. For this we capture the difference between the synchronous and the asynchronous pi-calculus within a so-called *synchronisation pattern*, i.e., a property of process terms defined on their transition system. Moreover, in Section 4.3 we go a step further and analyse the possibility to implement the asynchronous pi-calculus within a distributed setting. Intuitively, the degree of distributability corresponds to the amount of parallel components that can act independently. Practical experience has shown that it is not possible to implement every pi-calculus term—not even every asynchronous one—in an asynchronous setting while preserving its degree of distributability, at least not with an automatic algorithm. To overcome these problems, the join-calculus was introduced as a model of distributed computation [Lév97]. It employs a *locality* principle by ensuring that there is always exactly one immobile receiver for each communication channel. We formally prove that this locality principle indeed results in a gap between the expressive power of the join-calculus and the asynchronous pi-calculus with respect to distributability, i.e., we prove that there exists no encoding with respect to the general framework that also preserves distributability. Again, we capture the difference between these two languages

within a synchronisation pattern. The obtained synchronisation patterns provide an intuitive description of the difference in the expressive power of the join-calculus, the asynchronous pi-calculus, and the synchronous pi-calculus.

We conclude in Chapter 7 by combining the results of this thesis within a hierarchy of synchronous and asynchronous interactions in the pi-calculus and interactions in the join-calculus with respect to distributability.

1.1.3. Main Goals

Our main goal is to analyse and compare the expressive power of synchronous and asynchronous variants of the pi-calculus (and also the join-calculus) with respect to distributability. For this we (1) analyse and reconsider existing positive and negative translational results in this direction, (2) discuss how distributability can be formalised and what it means for an encoding to preserve distributability, and (3) analyse how this new criterion influence the known positive and negative results. Moreover, we want to present some kind of guideline on the derivation of translational expressiveness results, although we do not believe that this thesis presents a complete overview on the current state of research in this direction.

1.2. Publications

Some material of this thesis is already published, accepted for publication, or was recently submitted for publication.

The results of Section 4.1.3 and Section 4.2.1 were already published in:

1. *Breaking Symmetries* by Kirstin Peters and Uwe Nestmann; published in the proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS '10) [PN10a]

Some of the proofs of these results were presented within the following technical report.

2. *Breaking Symmetries* by Kirstin Peters and Uwe Nestmann; provided as technical report at http://arxiv.org/abs/1007.4172v1 [PN10b]

In

3. *Breaking Symmetries* by Kirstin Peters and Uwe Nestmann; to appear in Mathematical Structures in Computer Science (MSCS '12) [PN12a]

we extended [PN10a] and [PN10b] by the main idea of the encoding of mixed choice as presented in Section 5.3. We also discussed the main properties and drawbacks of this encoding attempt and sketch the main line of its proof of correctness. Hence this paper contains a very early rough draft of Chapter 6.

Section 4.2.2 contains a revised version of a result presented in

4. *Synchrony vs Causality in the Asynchronous Pi-Calculus* by Kirstin Peters, Jens-Wolfhard Schicke, and Uwe Nestmann; published in the proceedings of the 18th International Workshop on Expressiveness in Concurrency (EXPRESS '11) [PSN11]

In contrast to the presentation in Section 4.2.2, [PSN11] concentrated more on causality than distributability. In fact it was published before the formalisation of the criterion to measure preservation of distributability in Section 3.4 was derived. A companion-paper considers the effect of encoding synchronous interaction with respect to causality in the context of Petri nets [SPG11]. A combination and revision of these two papers was submitted as:

- *Synchrony versus Causality in Distributed Systems* by Kirstin Peters, Jens-Wolfhard Schicke-Uffmann, Ursula Goltz, and Uwe Nestmann; submitted.

The encoding of mixed choice as it is presented in Section 5.2.4 was already introduced in

5. *Is It a "Good" Encoding of Mixed Choice?* by Kirstin Peters and Uwe Nestmann; published in the proceedings of the 15th International Conference of Foundations of Software Science and Computational Structures (FoSSaCS '12) [PN12b]

Also its main properties and drawbacks are discussed. For this [PN12b] contains a preliminary version of the criterion presented in Section 3.4 on the preservation of distributability. But, in contrast to the actual formulation of this criterion, the version presented in [PN12b] was to focused on the pi-calculus. Based on the results in [PSN11] it was shown that no good encoding of mixed choice can preserve distributability. This result—together with a revised version of the result in [PSN11]—is presented in Section 4.2.2. The technical report

6. *Is it a "Good" Encoding of Mixed Choice? (Technical Report)* by Kirstin Peters and Uwe Nestmann; provided as technical report at http://arxiv.org/abs/1201.1410v1 [PN12c]

extends [PN12b] by the proof of the correctness of the encoding of mixed choice. In particular Section 6.3.1 and the Sections 6.3.3 to Section 6.3.6 result from a revision of the proofs presented in [PN12c]. Also the intermediate encoding of mixed choice with respect to polyadic synchronisation of Section 5.2.2 is already contained in [PN12c] and used in [PN12b] to validate the proposed preliminary version of the quality criterion of the preservation of distributability.

Finally, a shortened version of Section 3.4, Section 4.3, and Section 4.4 was recently submitted.

- *Distributability in Process Calculi* by Kirstin Peters, Uwe Nestmann, and Ursula Goltz; submitted.

Remark: The above papers were developed in a close collaboration with my co-authors. I was part of all aspects of the work with the definitions, separation results, design of encodings, and all the proofs (except of the result on the relationship between synchrony and causality in the context of Petri nets in [SPG11] that is not contained in this thesis).

2. Process Calculi

Process calculi focus on the specification and manipulation of process terms as induced by a collection of operator symbols [Fok07]. Process calculi usually come with a well-developed mathematical theory. In particular, they have a compositional semantics. Hence, two process terms with the same semantics (possibly modulo some notion of equivalence) may be exchanged in any context without changing the overall behaviour. This is an important property and a precondition for program transformations and modular design [Kie98].

The three (families of) process calculi—the pi-calculus, the join-calculus, and the CSP-calculus—that are considered in this thesis certainly belong to the most famous process calculi. Our main interest is in the pi-calculus [MPW92]. It evolved from CCS—the calculus of communicating systems [Mil80]. CCS provides an algebraic notion for systems or programs. Its focus is on the communications between components of the system, where communications are modelled as two-way rendezvous. An action a synchronises with its communication partner $\overline{a}$ yielding an internal τ-event. This distinguishes CCS from CSP, where communication is between arbitrary many participants [Hoa78]. An introduction to CCS can be found e.g. in [Mil80, Mil89, Bru97, Mil99], where the last also contains an introduction to the pi-calculus.

In order to express mobile systems, [MPW92] introduces the pi-calculus (or the π-calculus) as an extension of CCS (following the work of [EN86, EN00]). In the pi-calculus synchronising actions have the form $y(x)$ and $\overline{y}\langle z\rangle$, where the former represents the reception of a *value* x over a *link* or *channel* y and the later represents the transmission of the value z over link y. The transmission of links changes the structure of a system and, thus, expresses mobility. For this, links as well as the transmitted values are both taken from the same domain. Implementations of the pi-calculus into a concurrent programming language are for example considered in [PT97].

The pi-calculus is a well-known and frequently used process calculus to model concurrent systems. Therein, intuitively, the degree of distributability corresponds to the amount of parallel components that can act independently (see Section 3.4). However, practical experience has shown that it is not possible to implement every pi-calculus term—not even every asynchronous one—in an asynchronous setting while preserving its degree of distributability, at least not with an automatic algorithm. To overcome these problems, the join-calculus was introduced as a model of distributed computation [Lév97]. It employs a locality principle by ensuring that there is always exactly one immobile receiver for each communication channel. More precisely, for every name, exactly one receiver is defined at the time of the name's creation, and communication occurs only on so-defined channels [Fou98]. Apart from that, the join-calculus can be considered as a member (of the asynchronous branch) of the pi-calculus family.

In the following we present some basic definitions for process calculi in Section 2.1 followed by the definition of the variants of the pi-calculus (Section 2.1.1), the join-calculus (Section 2.1.2), and the CSP-calculus (Section 2.1.3) that are used throughout this thesis. Moreover, we discuss bisimulation in Section 2.2 as prominent proof technique to reason about process terms and introduce some standard bisimulations for the pi-calculus.

2.1. Basic Definitions

Assume a countably-infinite set $\mathcal{N}$, whose elements are called *names*. Names are the universe of elements of which the processes are constructed within (most of the) process calculi. We use lower case letters $a, b, c, \ldots, a', a_1, \ldots$ to range over names. Apart from names some process calculi utilise additional universes to represent some kind of data usually augmented with additional structure, as for instance the natural numbers $\mathbb{N}$ with their total ordering $\leq$. However, within this thesis we deliberately restrict our attention to process calculi that are constructed on top of the sole universe $\mathcal{N}$, because this simplifies several of the following considerations, as the definition of inference rules of the considered process calculi in the following sections or the introduction of type systems in Section 6.2, just to name two. There is one exception: we allow the use of indices in form of natural numbers in order to compare to the results of [Pal03] in Chapter 4. However, we do not allow to use functions or relations like $\leq$ on these indices, i.e., do not allow o benefit from additional structure. Hence, we can also consider them as special names whose syntactical representation are natural numbers instead of lower case letters.

A *process calculus* is a language $\mathcal{L} = \langle\, \mathcal{P}, \longmapsto \,\rangle$ that consists of a set of process terms $\mathcal{P}$ (its syntax) and a relation $\longmapsto \subseteq \mathcal{P} \times \mathcal{P}$ on process terms (its semantics). We often refer to process terms also simply as processes—in particular if we want to underline their character as elements of a language modelling real world objects and procedures—or as terms—in particular if we refer to their mathematical representation and underlying theory. But these intuitions are just guidelines. Indeed, we often treat the notions process term, process, and term as equivalent. We use upper case letters $P, Q, R, \ldots, P', P_1, \ldots$ to range over process terms.

The *syntax* is usually defined by a context-free grammar defining operators. An *operator* $\mathrm{op} : \mathcal{N}^n \times \mathcal{P}^m \to \mathcal{P}$ is a function from names and process terms into a process term. In this case, we say op has the *arity* m. Sometimes, process calculi also specify operators that, instead of a fixed number of arguments, accept any finite set of names and/or process terms usually indexed by a finite index set I. In this case, the arity of the operator is not a fixed value but, for a given set of arguments, is determined by the number of process terms among the arguments. An example of such an operator is given by the choice operator in the pi-calculus below. An operator of arity 0 is a *constant*. We require that each process calculus defines at least the *empty process* as constant and the *parallel operator* as binary operator. Note that, similar requirements can also be found e.g. in [VPP07, Gor10b]. Moreover, in the style of [Gor10b] we add the special process

✓ to each process calculus. Its purpose is to denote *success* which allows us to compare the abstract behaviour of terms in different process calculi as described in Section 3.3. The arguments of an operator that are again process terms are called *subterms* of P.

Definition 2.1.1 (Subterms)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathcal{P}$ a process term. The set of *subterms* of P is defined as

- if P is a constant, i.e., $P = \mathrm{op}\,(x_1, \ldots, x_n)$ for some $x_1, \ldots, x_n \in \mathcal{N}$, then the only subterm of P is P itself, and
- else, if $P = \mathrm{op}\,(x_1, \ldots, x_n, P_1, \ldots, P_m)$ with $x_1, \ldots, x_n \in \mathcal{N}$ and $P_1, \ldots, P_m \in \mathcal{P}$, then the set of subterms of P contains P and all subterms of $P_1, \ldots, P_m$, i.e., the set of subterms of P is given by $\{\, P, P' \mid \exists i \in \{\, 1, \ldots, m \,\} \;.\; P' \text{ is a subterm of } P_i \,\}$.

We assume that the *semantics* is given as *operational semantics* consisting of inference rules defined for the operators of the language [AFV01, Plo04, GMR06, MRG07]. For many process calculi, the semantics is provided in two forms, as *reduction semantics* and as *labelled semantics*. We assume that at least the reduction semantics $\longmapsto$ is given as part of the definition. The labelled semantics is considered only as an additional formulation of the behaviour, because the treatment of the reduction semantics is easier in the context of encodings as explained in Section 3.3. In the case of the pi-calculus, we introduce both, a labelled and a reduction semantics. A single application of the reduction semantics is called a reduction step or shortly a step.

Definition 2.1.2 (Step)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus and $P, P' \in \mathcal{P}$. The pair $(P, P') \in \longmapsto$ is called a *(reduction) step* of P and is written as $P \longmapsto P'$.

If $P \longmapsto P'$ is a step, we say P' is a *derivative* of P. Moreover, let $P \longmapsto$ denote the existence of a step from P, i.e., $P \longmapsto \triangleq\ \exists P' \in \mathcal{P} \;.\; P \longmapsto P'$, let $P \not\longmapsto$ denote the absence of a step from P, i.e., $P \not\longmapsto \triangleq\ \neg\,(P \longmapsto)$, and let $\Longmapsto$ denote the reflexive and transitive closure of $\longmapsto$.

Accordingly, we call an application of the labelled semantics as a *labelled step*. A sequence of reduction steps is called a reduction.

Definition 2.1.3 (Reduction)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathcal{P}$. A sequence of steps from P is called a *reduction*.

Reductions are either finite, as in $P_1 \longmapsto \ldots \longmapsto P_n$ or $P_1 \Longmapsto P_n$ for some $P_1, \ldots, P_n \in \mathcal{P}$, or they are infinite, as in $P_1 \longmapsto P_2 \longmapsto \ldots$ for some $P_1, P_2, \ldots \in \mathcal{P}$. We write $P \longmapsto^{\omega}$ if P has an infinite sequence of steps.

We also use *execution* to refer to a reduction starting from a particular term. A *maximal execution* of a process P is a reduction starting from P that cannot be further extended.

An infinite execution is an infinite sequence of finite executions. Note that an infinite execution is not necessarily maximal. We do not forbid empty reductions, which can be visualised as $P \Longmapsto P$, but of course the same notation is e.g. used to abbreviate a reduction $P \longmapsto P' \longmapsto P$.

Within this thesis, we often write inference rules in the form

$$\mathsf{Name} \quad \frac{A_1, \ldots, A_n}{C} \quad S_1, \ldots, S_m$$

where Name is the name of the rule, $A_1, \ldots, A_n$ is a set of subgoals, C is the conclusion of the rule, and $S_1, \ldots, S_m$ are side conditions. If $n = 0$ the corresponding rule is an axiom. In the case of reduction rules the conclusion C is a step $P \longmapsto P'$ there P, P' are process terms with process variables that allow the match of concrete terms against the rule. To reason about environments we use functions on process terms called contexts.

Definition 2.1.4 (Context)**.** Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus. A *context*

$$\mathcal{C}([\cdot]_1, \ldots, [\cdot]_n) : \mathcal{P}^n \to \mathcal{P}$$

with n holes is a function from n process terms into a process term, i.e., given n terms $P_1, \ldots, P_n \in \mathcal{P}$, the term $\mathcal{C}(P_1, \ldots, P_n)$ is the result of inserting the n terms $P_1, \ldots, P_n$ in that order into the n holes of $\mathcal{C}$.

Note that a context may bind some free names of its parameters.

To compare process terms, process calculi usually come with different well-studied equivalence relations (see [vG01, vG93] for an overview and a classification of the most frequent equivalences). Remember that an equivalence on a set M is a relation $\mathcal{R} \subseteq M \times M$, that is reflexive, symmetric, and transitive. A special kind of equivalence with great importance to reason about processes are congruences. A congruence is the closure of an equivalence with respect to contexts.

Definition 2.1.5 (Congruence)**.** Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus. An equivalence $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *congruence* if $(P, Q) \in \mathcal{R}$ implies $(\mathcal{C}(P), \mathcal{C}(Q)) \in \mathcal{R}$ for all terms $P, Q \in \mathcal{P}$ and all contexts $\mathcal{C}([\cdot]) : \mathcal{P} \to \mathcal{P}$.

Moreover, let C be a set of contexts such that for all $\mathcal{C} \in C$ the context $\mathcal{C}$ is of type $\mathcal{P} \to \mathcal{P}$. Then an equivalence $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *congruence with respect to* C if $(P, Q) \in \mathcal{R}$ implies $(\mathcal{C}(P), \mathcal{C}(Q)) \in \mathcal{R}$ for all terms $P, Q \in \mathcal{P}$ and all contexts $\mathcal{C} \in C$.

To prove properties of an encoding the set of considered contexts has sometimes to be restricted to contexts that respect the protocol behind the encoding (see e.g. [VP96]). We use this technique in Section 6.3.4.

Moreover, process calculi usually come with a special congruence $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ called *structural congruence*. Its main purpose is to equate syntactically different process terms that model quasi-identical behaviour. Often, this fact is captured explicitly by an inference rule of the form

$$\frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

in the reduction semantics. If the process calculus does not come with a standard structural congruence, we either use equality instead or we derive the rules of structural congruence by arguing as in the rule above. However, obtaining a structural congruence

in this way usually leads to unnecessary and very complicated rules. So, we strongly recommend to use the standard structural congruence of a calculus if available.

The meaning of operators is defined by the inference rules in the operational semantics. We distinguish between reducible (or dynamic) and static operators (compare e.g. to the static and dynamic laws in [Mil89] or the static and dynamic constructions in [Mil93a]). Intuitively, a *reducible operator* defines parts of terms that can perform steps, while *static operators* define connections between terms and side conditions on the reductions of their respective subterms, i.e., they allow for new reductions or forbid reductions of their subterms. We sometimes denote the parts of a term that are removed or *reduced* in reduction steps as *capabilities*, to illustrate that they indicate the parts of a term that can perform steps. A typical example of a reducible operator are prefix operators as $y(x).P$ in the pi-calculus or $y!z \rightarrow P$ in CSP, where the prefixes $y(x)$ and $y!z$ are capabilities of the respective calculi. Note that static operators are usually manipulated by the rules of structural congruence, whereas the reduction of reducible operators is often described by the axioms of the reduction semantics. The remaining inference rules of the reduction semantics usually describe the interplay with static operators. A typical static operator that we assume to be part of every process calculus is the parallel operator. Apart from reducible and static operators, we distinguish between operators that allow for reductions of their subterms and those that require to be reduced first. More precisely, we denote an operator as *guard* if at least one of its subterms cannot be used to perform a step before the guard itself is reduced by a reduction step. Its subterm(s) that cannot perform steps before the guard is reduced are called *guarded* subterms. The other subterms, if there are any, as well as the subterms of operators that are no guards are called *unguarded* subterms. Basically, all operators for sequential compositions of terms or mutual exclusive alternatives are guards, whereas constants are never guards, simply because they have no subterms. Note that a guard usually guards all of its subterms, as it is the case in the pi-calculus. However, there are process calculi, as the join-calculus, where a single operator combines different needs and, thus, guards only some of its subterms. A guard is called *context-sensitive* if its reducibility (in a reduction semantics) depends on a given context, i.e., if there are some contexts in which the guard can be reduced and some in which the guard cannot be reduced although it appears unguarded within the contexts. As an example in the asynchronous pi-calculus the input $y(x).P$ is context-sensitive, because it can only be reduced if it appears unguarded within a context that provides an unguarded matching output $\overline{y}\langle z\rangle$, whereas the term $\tau.P$ can reduce in any context in which it appears unguarded, i.e., the τ-prefix is a guard that is not context-sensitive.

Replication or recursion can be provided by reducible or static operators. For the pi-calculus typical forms are $y^\star(x).P$ (reducible operator) or $!P$ (static operator). Also the semantics can be given by a reduction rule as $y^\star(x).P \mid \overline{y}\langle z\rangle \longmapsto \{\, {}^{z}/_{x} \,\} P \mid y^\star(x).P$ or by a rule of structural congruence as $y^\star(x).P \equiv y(x).P \mid y^\star(x).P$. In both cases, recursion or replication distinguishes itself from other operators by the fact that (one of) its subterms can be copied within rules of structural congruence or by reduction rules while the operator itself is usually never removed during reductions. We call such operators and capabilities *recurrent*.

Another typical operator is the restriction of scopes of names. A *scope* defines an area in which a particular name is known and can be used. For several reasons, it can be useful to restrict the scope of a name. For instance to forbid interactions between two processes or with an unknown and, hence, potentially untrusted environment. Names whose scope is restricted such that they cannot be used from outside the scope are denoted as *bound names*. The remaining names are called *free names*. Accordingly, we assume three sets—the sets of names $\mathsf{n}(P)$ and its subsets of free names $\mathsf{fn}(P)$ and bound names $\mathsf{bn}(P)$—for each term P. In the case of bound names, their syntactical representation as lower case letter serves as a place holder for any fresh name, i.e., any name that does not occur elsewhere in the term. To avoid name capture or clashes, i.e., to avoid confusion between free and bound names or different bound names, bound names can be mapped to fresh names by *α-conversion*. We write $P \equiv_\alpha Q$ if P and Q differ only by α-conversion. We use σ, σ', σ_1, ... to range over substitutions. A substitution is a finite mapping from names to names defined by a set $\{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}$ of renamings, where the $x_1, \ldots, x_n$ are pairwise distinct. The application of a substitution to a term $\{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}(P)$ is defined as the result of simultaneously replacing all free occurrences of x_i by y_i for $i \in \{ 1, \ldots, n \}$, possibly applying α-conversion to avoid capture or name clashes. For all names $\mathcal{N} \setminus \{ x_1, \ldots, x_n \}$ the substitution behaves as the identity mapping. Let id denote identity, i.e., id is the empty substitution $\mathsf{id} = \emptyset$. We sometimes omit the parentheses, i.e., $\sigma(P) = \sigma P$.

Let $\mathcal{S}(M)$ denotes the set of finite sequences of elements of M. Then $\tilde{x} \in \mathcal{S}(M)$ is such a finite sequence, i.e., if $M = \mathcal{N}$, $\tilde{x} = x_1, \ldots, x_n$ is a finite sequence of names for some $n \in \mathbb{N}$ and some $x_1, \ldots, x_n \in \mathcal{N}$. The length of a sequence $\tilde{x}$ is denoted by $|\tilde{x}|$, i.e., if $\tilde{x} = x_1, \ldots, x_n$ then $|\tilde{x}| = n$. Moreover, we naturally extend substitutions to sequences of names, i.e., if $|\tilde{x}| = n = |\tilde{y}|$ and all names in $\tilde{x}$ are pairwise different then $\{ {}^{\tilde{y}}/_{\tilde{x}} \} = \{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}$.

A *network* is the parallel composition of processes. Usually, we allow that these processes can be surrounded by some static operator, for example to implement some restriction on names that should be private to the network. Moreover, note that the processes of a network can be again networks.

The semantics of process calculi is often given as *interleaving semantics*, i.e., the semantics specifies the execution of single steps and concurrency is simulated via non-deterministic interleaving. In contrast, semantics that explicitly consider the concurrent executions of steps are often denoted as "truly concurrent" or *step-semantics* (see e.g. [Pra86, Old87, DDNM88, MP95, AM96, Kie98, Lan07]). Within this thesis we restrict our attention mainly to interleaving semantics. Consequently, the semantics introduced in the following for different process calculi are interleaving semantics. However, for Section 3.4, Section 4.3, and Section 4.4 we assume the existence of a step-semantics for the considered calculi without explicitly referring to a particular such semantics.

2.1.1. The Pi-Calculus

In the following, we introduce different variants of the pi-calculus as described for instance in [MPW92, Mil99, SW01]. Later on we compare these variants with respect to

their expressive power. Thereby the centre of interest is to compare synchronous and asynchronous interactions. Note that the full pi-calculus describes synchronous interactions, whereas asynchronous interactions are described by one of its subcalculi, the asynchronous pi-calculus. As already demonstrated in [Pal03] and further exposed in Chapter 4, the most interesting operator for a comparison of the expressive power between these two calculi is mixed choice, i.e., choice between input and output capabilities. Thus, we denote the full pi-calculus by π_{m}.

Let $\tau \notin \mathcal{N}$ and $\overline{\mathcal{N}}$ the set of co-names, i.e., $\overline{\mathcal{N}} = \{\, \overline{n} \mid n \in \mathcal{N} \,\}$.

Definition 2.1.6 (π_{m})**.** The set of process terms of the *synchronous pi-calculus (with mixed choice)*, denoted by $\mathcal{P}_{\mathrm{m}}$, is given by

$$P ::= \checkmark \quad | \quad (\nu x)\, P \quad | \quad P_1 \mid P_2 \quad | \quad y^{\star}(x)\,.P \quad | \quad \sum_{i \in I} \pi_i.P_i$$

$$\pi ::= \overline{y}\langle z \rangle \quad | \quad y(x) \quad | \quad \tau$$

for some names $x, y, z \in \mathcal{N}$ and a finite index set I.

The interpretation of the defined process terms is as usual. *Restriction* $(\nu x)\, P$ restricts the scope of the name x to the definition of P. The *parallel composition* $P_1 \mid P_2$ defines the process in which P_1 and P_2 may proceed independently, possibly interacting using shared links. $y^{\star}(x)\,.P$ denotes *input-guarded replication*. It is the only recurrent operator of π_{m}. The process term $\sum_{i \in I} \pi_i.P_i$ represents *finite guarded choice*; as usual, the sum $\sum_{i \in \{\, 1,\ldots,n \,\}} \pi_i.P_i$ is sometimes written as $\pi_1.P_1 + \ldots + \pi_n.P_n$ and $\mathsf{0}$ abbreviates the empty sum, i.e., where $I = \emptyset$.

The capabilities of the pi-calculus are the (replicated) input prefix $y(x)$, the output prefix $\overline{y}\langle z \rangle$, and the prefix τ, where the capability of a choice is the conjunction of the prefixes of all its branches—considered as single capability. The input prefix $y(x)$ is used to describe the ability of receiving the value x over link y and, analogously, the output prefix $\overline{y}\langle z \rangle$ describes the ability to send a value z over link y. The prefix τ describes the ability to perform an internal, not observable action. We sometimes refer to input and output prefixes as action prefixes. Prefixes are guards and all their subterms are guarded. Hence, recursion is defined for input-guarded processes only and the branches of sums are also always guarded.

The definitions of free and bound names are completely standard, i.e., names are bound by restriction and as parameter of input or replicated input and $\mathsf{n}(P) = \mathsf{fn}(P) \cup \mathsf{bn}(P)$ for all P. We naturally extend substitutions to co-names, i.e., $\forall \overline{n} \in \overline{\mathcal{N}}\ .\ \sigma(\overline{n}) = \overline{\sigma(n)}$ for all substitutions σ.

As usual, the continuation $\mathsf{0}$ is often omitted, so $y(x).\mathsf{0}$ becomes $y(x)$. In addition, for simplicity in the presentation of examples, we sometimes omit an action's object when it does not effectively contribute to the behaviour of a term. Typically, we do this when it would be enough to use a CCS-like example, but the above definitions of the pi-calculus would force us to carry *some* object along that would never be used on a receiver side, e.g. as in $y(x)\,.\mathsf{0}$, which would be written as $y.\mathsf{0}$ or just y. Moreover, let $(\nu \tilde{x})\, P$ abbreviate the term $(\nu x_1) \ldots (\nu x_n)\, P$.

The expressive power of π_{m} is compared to two of its subcalculi: π_{s}, the pi-calculus with separate choice, and π_{a}, the asynchronous pi-calculus. In π_{s}, both output and input can be used as guards, but within a single choice term either there are no input or no output guards, i.e., we have input- and output-guarded choice, but no mixed choice.

Definition 2.1.7 (π_{s})**.** The set of process terms of the *pi-calculus with separate choice*, denoted by $\mathcal{P}_{\mathrm{s}}$, are given by

$$P ::= \checkmark \;\mid\; (\nu x)\, P \;\mid\; P_1 \mid P_2 \;\mid\; y^{\star}(x)\,.P \;\mid\; \sum_{i\in I} \pi_i^O.P_i \;\mid\; \sum_{i\in I} \pi_i^I.P_i$$

$$\pi^O ::= \overline{y}\langle z\rangle \;\mid\; \tau \qquad \pi^I ::= y(x) \;\mid\; \tau$$

for some names $x, y, z \in \mathcal{N}$ and a finite index set I.

As expected, the definitions of π_{s} and π_{m} differ in the definition of choice only.

Asynchronous variants of the pi-calculus were introduced independently by [HT91, HT92] and [Bou92]. In asynchronous communication, a process has no chance to directly determine, i.e., without a hint by another process, whether a value sent by it was already received or not. To model that fact in the asynchronous pi-calculus (π_{a}), output actions are not allowed to guard a process different from $\mathsf{0}$. Accordingly, the interpretation of output guards within a choice construct is delicate. Here, we use the standard variant of π_{a}, where choice is not allowed at all. Since π_{a} has no choice, and thus no nullary choice, we include $\mathsf{0}$ as a primitive.

Definition 2.1.8 (π_{a})**.** The set of process terms of the *asynchronous π-calculus*, denoted by $\mathcal{P}_{\mathrm{a}}$, are given by

$$P ::= \mathsf{0} \;\mid\; \checkmark \;\mid\; (\nu x)\, P \;\mid\; P_1 \mid P_2 \;\mid\; y^{\star}(x)\,.P \;\mid\; \overline{y}\langle z\rangle \;\mid\; y(x)\,.P \;\mid\; \tau.P$$

for some names $x, y, z \in \mathcal{N}$.

Since π_{a} has no choice, and thus no nullary choice, we include $\mathsf{0}$ as a primitive.

Note that in the formulations of π_{m}, π_{s}, and π_{a} so far we omit one standard operator of the pi-calculus. The *match prefix*

$$[a = b]\, P,$$

for some names $a, b \in \mathcal{N}$ and a process term P, works as a conditional guard. It can be removed if and only if a and b are equal. We use the superscript $=$ to denote process calculi that are augmented with this operator. Hence, $\pi_{\mathrm{a}}^{=}$ denotes the asynchronous pi-calculus with the match prefix, where its set of process terms $\mathcal{P}_{\mathrm{a}}^{=}$ is defined by the grammar in Definition 2.1.8 extended with $[a = b]\, P$.

Furthermore, we consider a not standard extension of the asynchronous pi-calculus denoted as *polyadic synchronisation*. Polyadic synchronisation was defined by Carbone and Maffeis in [CM03] to permit divergence-free encodings of distributed calculi. Moreover, [CM03] show that polyadic synchronisation significantly increases the expressive power of the pi-calculus and allows to simulate the match prefix. Indeed, we will use

it to obtain an encoding of mixed choice in Chapter 5, whereas—as we conjecture—a divergence-free encoding without this parameter is only possible if we allow for the match prefix in the target language. Polyadic synchronisation defines new action prefixes with links that, instead of a single name, are composed of several channel names. In principle [CM03] allow for compositions of any number of names to build a single link. However, we need only polyadic synchronisation with links composed from maximal two names. Moreover, we need this extension only for the asynchronous pi-calculus. Accordingly, we extend the definition of $\mathcal{P}_{\mathrm{a}}$ by the two operators

$$\overline{y_1 \cdot y_2}\langle z\rangle \quad \mid \quad y_1 \cdot y_2(x)\,.P,$$

where $x, y_1, y_2, z \in \mathcal{N}$, introducing output and input on channels composed of two names. Two polyadic channels are equal if and only if they have the same length and are composed of the same names in the same order. In contrast to the usual π-calculus, polyadic synchronisation allows us to restrict parts of a channel such that $((\nu y_1)\,\overline{y_1 \cdot y_2}\langle z\rangle) \mid y_1 \cdot y_2(x) \not\mapsto$. Then the set of process terms $\mathcal{P}_{\mathrm{p}}$ of of the asynchronous pi-calculus with polyadic synchronisations, denoted by π_{p}, is defined by the grammar in Definition 2.1.8 extended with $\overline{y_1 \cdot y_2}\langle z\rangle$ and $y_1 \cdot y_2(x)\,.P$.

So far, we consider only monadic variants of the pi-calculus in that links carry exactly one value. Polyadic variants are obtained from the respective monadic variants by replacing within the grammar of the respective definition

- the output prefixes $\overline{y}\langle z\rangle$ and $\overline{y_1 \cdot y_2}\langle z\rangle$ with $\overline{y}\langle z_1, \ldots, z_n\rangle$ and $\overline{y_1 \cdot y_2}\langle z_1, \ldots, z_n\rangle$, and
- the (replicated) input prefixes $y(x)$, $y^\star(x)$, and $y_1 \cdot y_2(x)$ with the (replicated) input prefixes $y(x_1, \ldots, x_n)$, $y^\star(x_1, \ldots, x_n)$, and $y_1 \cdot y_2(x_1, \ldots, x_n)$,

where n is an arbitrary natural number and $x_1, \ldots, x_n, y, y_1, y_2, z_1, \ldots, z_n \in \mathcal{N}$. Note that all names in $x_1, \ldots, x_n$ have to be pairwise different. Again, we abbreviate sequences of names $x_1, \ldots, x_n$ by $\tilde{x}$. Moreover, we use the symbol $\sim$ as superscript to denote a polyadic variant of the pi-calculus. At the end of this section we present an overview on the variants of the pi-calculus used within this thesis.

A *network* is a process $(\nu\tilde{x})\,(P_1 \mid \ldots \mid P_n)$ for some $n \in \mathbb{N}$, some terms $P_1, \ldots, P_n \in \mathcal{P}$, and $\tilde{x} \in \mathcal{S}(\mathcal{N})$, where $\mathcal{P}$ is the set of processes of one of the above defined variants of the pi-calculus. We refer to $P_1, \ldots, P_n$ as the (sub)processes of the network.

The structural congruence for all defined variants of the pi-calculus is jointly given by the rules in Figure 2.1. Note that the Rule $[a = a]\,P \equiv P$ is superfluous in variants of the pi-calculus that do not contain the match prefix. Moreover, we temporary add the Rule $y^\star(x)\,.P \equiv y(x)\,.P \mid y^\star(x)\,.P$ in Section 3.4 in order to define distributability.

For the first two variants, π_{m} and π_{s}, we define a labelled as well as a reduction semantics, because we conveniently use them for different purposes. The labelled semantics is used in the first part of Chapter 4 to review a separation result of [Pal03] and to obtain a similar result while the reduction semantics are used throughout the reset of this thesis to obtain separation as well as encodability results. We start with the labelled semantics. Let $\mathcal{A} \triangleq \{\, y\,(x)\,,\ \overline{y}\,z,\ \overline{y}\,(z) \mid x, y, z \in \mathcal{N} \,\}$ denote the set of monadic

$$P \equiv Q \text{ if } P \equiv_\alpha Q \qquad P \mid \mathsf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$[a = a]\, P \equiv P \qquad (\nu n)\, \mathsf{0} \equiv \mathsf{0} \qquad (\nu n)\,(\nu m)\, P \equiv (\nu m)\,(\nu n)\, P$$

$$P \mid (\nu n)\, Q \equiv (\nu n)\,(P \mid Q) \qquad \text{if } n \notin \mathsf{fn}(P)$$

Figure 2.1.: Structural Congruence in the Pi-Calculus.

action labels for visible actions, where $y\,(x)$ denotes *free input*, $\overline{y}\,z$ denotes *free output*, and $\overline{y}\,(z)$ denotes *bound output*, respectively. Let τ denote an internal invisible action whose label is denoted by τ as well. Let $\mathcal{A}_\tau$ be the corresponding set of *labels*, i.e., $\mathcal{A}_\tau = \mathcal{A} \cup \{\, \tau \,\}$. We use $\mu, \mu', \mu_1, \ldots$ to range over labels. Moreover, let $\mathsf{fn}(\mu)$ denote the sets of *free names* in μ, $\mathsf{bn}(\mu)$ denote the sets of *bound names* in μ, and $\mathsf{n}(\mu)$ denote the sets of all *names* occurring in μ, respectively. Their definitions are completely standard, i.e., names are bound when they are the object of input or bound output, $\mathsf{n}(\tau) = \emptyset$, and $\mathsf{n}(\mu) = \mathsf{fn}(\mu) \cup \mathsf{bn}(\mu)$ for all $\mu \in \mathcal{A}_\tau$.

To avoid confusion, we use $\xrightarrow{\mu}$ with $\mu \in \mathcal{A}_\tau$ for steps within the labelled semantics and $\longmapsto$ within the reduction semantics. Moreover, let $P \longrightarrow (P \not\longrightarrow)$ denote existence (non-existence) of a step from P, i.e., there is (no) $P' \in \mathcal{P}_\mathrm{m}$ or $P' \in \mathcal{P}_\mathrm{s}$ and (no) $\mu \in \mathcal{A}_\tau$ such that $P \xrightarrow{\mu} P'$. Moreover, we sometimes use $\Longrightarrow$ to abbreviate a sequence of τ-steps, i.e., $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. Similarly, we write $\overset{\mu}{\Longrightarrow}$ for some $\mu \in \mathcal{A}$ to abbreviate the sequence $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$, and $\overset{\hat{\mu}}{\Longrightarrow}$ for some $\mu \in \mathcal{A}_\tau$ to abbreviate $\Longrightarrow$ if $\mu = \tau$ and else $\overset{\mu}{\Longrightarrow}$. The *labelled semantics* of π_m and π_s are jointly given by the transition rules in Figure 2.2.

A *(partial) execution* of length n is a sequence of steps $P \xrightarrow{\mu_1,\ldots,\mu_n} P'$ such that $P \xrightarrow{\mu_1} H_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{n-1}} H_{n-1} \xrightarrow{\mu_n} P'$ for some $P', H_1, \ldots, H_{n-1} \in \mathcal{P}_\mathrm{m}$ or $P', H_1, \ldots, H_{n-1} \in \mathcal{P}_\mathrm{s}$ with the sequence $\mu_1, \ldots, \mu_n$ of observable and unobservable actions, i.e., $\mu_1, \ldots, \mu_n \in \mathcal{A}_\tau$. Accordingly, $P \xrightarrow{\tilde{\mu}} P' \not\longrightarrow$ denotes a finite execution from P to P' with the sequence of actions $\tilde{\mu} \in \mathcal{S}(\mathcal{A}_\tau)$. An infinite execution is an infinite sequence of finite executions. A *maximal execution* is an execution that cannot be further extended. Note that an infinite execution is not necessarily maximal.

Moreover, let $P\,(\rightsquigarrow)^n\,Q$ denote a sequence of n $\rightsquigarrow$-steps from P to Q for every kind of steps $\rightsquigarrow$; e.g. in the case of $\longmapsto$-steps $P\,(\longmapsto)^3\,Q$ denotes a sequence of three reduction steps from P to Q, i.e., $P \longmapsto\longmapsto\longmapsto Q$.

The *reduction semantics* of π_m, π_s, $\pi_\mathrm{s}^=$, π_a, $\pi_\mathrm{a}^=$, and π_p are jointly given by the transition rules in Figure 2.3, where the indices m, s, a, and p refer to rules of π_m, π_s, π_a, and π_p, respectively. Moreover, note that the presence or absence of the match prefix does not influence the reduction rules, i.e., the reduction semantics of $\pi_\mathrm{s}^=$ and $\pi_\mathrm{a}^=$ are given by the rules for π_s and π_a, respectively.

In Figure 2.5 we present an overview over all variants of the pi-calculus that are

$$\textsc{Pi-ls-O-Sum}\quad \frac{}{\sum_{i\in I}\pi_i.P_i \xrightarrow{\overline{y}\,z} P_j}\quad \pi_j = \overline{y}\langle z\rangle\,, j\in I$$

$$\textsc{Pi-ls-I-Sum}\quad \frac{}{\sum_{i\in I}\pi_i.P_i \xrightarrow{y(z)} \{\,{}^{z}/_{x}\,\}\,P_j}\quad \pi_j = y(x)\,, j\in I$$

$$\textsc{Pi-ls-Tau-Sum}\quad \frac{}{\sum_{i\in I}\pi_i.P_i \xrightarrow{\tau} P_j}\quad \pi_j = \tau, j\in I$$

$$\textsc{Pi-ls-Rep}\quad \frac{}{y^{\star}(x)\,.P \xrightarrow{y(z)} \{\,{}^{z}/_{x}\,\}\,P \mid y^{\star}(x)\,.P}$$

$$\textsc{Pi-ls-Com}\quad \frac{P \xrightarrow{y(z)} P' \quad Q \xrightarrow{\overline{y}\,z} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \textsc{Pi-ls-Open}\quad \frac{P \xrightarrow{\overline{y}\,z} P'}{(\nu z)\,P \xrightarrow{\overline{y}(z)} P'}\quad y\neq z$$

$$\textsc{Pi-ls-Close}\quad \frac{P \xrightarrow{y(z)} P' \quad Q \xrightarrow{\overline{y}(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)\,(P' \mid Q')}\quad z\notin \mathsf{fn}(P)$$

$$\textsc{Pi-ls-Par}\quad \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q}\quad \mathsf{bn}(\mu)\cap\mathsf{fn}(Q)=\emptyset$$

$$\textsc{Pi-ls-Res}\quad \frac{P \xrightarrow{\mu} P'}{(\nu x)\,P \xrightarrow{\mu} (\nu x)\,P'}\quad x\notin \mathsf{n}(\mu)$$

$$\textsc{Pi-ls-Cong}\quad \frac{P\equiv Q \quad Q \xrightarrow{\mu} Q' \quad Q'\equiv P'}{P \xrightarrow{\mu} P'}$$

Figure 2.2.: Labelled Semantics of π_{m} and π_{s}.

$$\textsc{Pi-Tau}_{\text{m,s}} \quad \frac{}{\sum_{i\in I}\pi_i.P_i \longmapsto P_j} \quad \pi_j = \tau, j \in I \qquad \textsc{Pi-Tau}_{\text{a,p}} \quad \frac{}{\tau.P \longmapsto P}$$

$$\textsc{Pi-Com}_{\text{m,s}} \quad \frac{}{\sum_{i\in I_1}\pi_i.P_i \mid \sum_{j\in I_2}\pi_j.P_j \longmapsto \{\, {}^{z}/_{x} \,\} P_k \mid P_l} \quad \begin{array}{l}\pi_k = y(x)\,, k \in I_1,\\ \pi_l = \overline{y}\langle z\rangle\,, l \in I_2\end{array}$$

$$\textsc{Pi-Com}_{\text{a,p}} \quad \frac{}{y(x)\,.P \mid \overline{y}\langle z\rangle \longmapsto \{\, {}^{z}/_{x} \,\} P}$$

$$\textsc{Pi-ComPS}_{\text{p}} \quad \frac{}{y_1 \cdot y_1(x)\,.P \mid \overline{y_1 \cdot y_2}\langle z\rangle \longmapsto \{\, {}^{z}/_{x} \,\} P}$$

$$\textsc{Pi-Rep}_{\text{m,s}} \quad \frac{}{y^{\star}(x)\,.P \mid \sum_{i\in I_1}\pi_i.P_i \longmapsto \{\, {}^{z}/_{x} \,\} P \mid y^{\star}(x)\,.P \mid P_j} \quad \pi_j = \overline{y}\langle z\rangle\,, j \in I$$

$$\textsc{Pi-Rep}_{\text{a,p}} \quad \frac{}{y^{\star}(x)\,.P \mid \overline{y}\langle z\rangle \longmapsto \{\, {}^{z}/_{x} \,\} P \mid y^{\star}(x)\,.P}$$

$$\textsc{Pi-Par}_{\text{m,s,a,p}} \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \textsc{Pi-Res}_{\text{m,s,a,p}} \quad \frac{P \longmapsto P'}{(\nu x)\,P \longmapsto (\nu x)\,P'}$$

$$\textsc{Pi-Cong}_{\text{m,s,a,p}} \quad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

Figure 2.3.: Reduction Semantics of the Monadic Variants of the Pi-Calculus.

$$\textsc{Pi-Tau}^{\sim}_{\text{a,p}} \quad \frac{}{\sum_{i\in I}\pi_i.P_i \longmapsto P_j} \quad \pi_j = \tau, j \in I \qquad \textsc{Pi-Tau}^{\sim}_{\text{a,p}} \quad \frac{}{\tau.P \longmapsto P}$$

$$\textsc{Pi-Com}^{\sim}_{\text{m,s}} \quad \frac{}{\sum_{i\in I_1}\pi_i.P_i \mid \sum_{j\in I_2}\pi_j.P_j \longmapsto \{\, \tilde{z}/\tilde{x} \,\} P_k \mid P_l} \quad \begin{array}{l} \pi_k = y(\tilde{x})\,, k \in I_1, \\ \pi_l = \overline{y}\langle\tilde{z}\rangle\,, l \in I_2, \\ |\tilde{x}| = |\tilde{z}| \end{array}$$

$$\textsc{Pi-Com}^{\sim}_{\text{a,p}} \quad \frac{}{y(\tilde{x})\,.P \mid \overline{y}\langle\tilde{z}\rangle \longmapsto \{\, \tilde{z}/\tilde{x} \,\} P} \quad |\tilde{x}| = |\tilde{z}|$$

$$\textsc{Pi-ComPS}^{\sim}_{\text{p}} \quad \frac{}{y_1 \cdot y_2(\tilde{x})\,.P \mid \overline{y_1 \cdot y_2}\langle\tilde{z}\rangle \longmapsto \{\, \tilde{z}/\tilde{x} \,\} P} \quad |\tilde{x}| = |\tilde{z}|$$

$$\textsc{Pi-Rep}^{\sim}_{\text{m,s}} \quad \frac{}{y^{\star}(\tilde{x})\,.P \mid \sum_{i\in I_1}\pi_i.P_i \longmapsto \{\, \tilde{z}/\tilde{x} \,\} P \mid y^{\star}(\tilde{x})\,.P \mid P_j} \quad \begin{array}{l} \pi_j = \overline{y}\langle\tilde{z}\rangle\,, j \in I, \\ |\tilde{x}| = |\tilde{z}| \end{array}$$

$$\textsc{Pi-Rep}^{\sim}_{\text{a,p}} \quad \frac{}{y^{\star}(\tilde{x})\,.P \mid \overline{y}\langle\tilde{z}\rangle \longmapsto \{\, \tilde{z}/\tilde{x} \,\} P \mid y^{\star}(\tilde{x})\,.P} \quad |\tilde{x}| = |\tilde{z}|$$

$$\textsc{Pi-Par}_{\text{m,s,a,p}} \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \textsc{Pi-Res}_{\text{m,s,a,p}} \quad \frac{P \longmapsto P'}{(\nu x)\,P \longmapsto (\nu x)\,P'}$$

$$\textsc{Pi-Cong}_{\text{m,s,a,p}} \quad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

Figure 2.4.: Reduction Semantics of the Polyadic Variants of the Pi-Calculus.

used within this thesis. Column 1–4 show the differences in the used operators, where $+_m$ represents mixed choice like $\overline{y}\langle x\rangle.P_1 + y(x).P_2$, $+_s$ represents separate choice like $\overline{y}\langle x\rangle.P_1 + \overline{y}\langle x\rangle.P_2$ or $y(x).P_1 + y(x).P_2$, $=$, represents a match prefix like $[a=b]\,P$, and $y_1 \cdot y_2$ represents polyadic synchronisation. In the last two coloumns a link to the Figures containing the labelled semantics ($\xrightarrow{\mu}$) and the reduction semantics ($\longmapsto$) is provided. Moreover, remember that all calculi without choice, i.e., π_{a}, $\pi_{\mathrm{a}}^{=}$, π_{p}, $\pi_{\mathrm{a}}^{\sim}$, $\pi_{\mathrm{a}}^{=,\sim}$, and $\pi_{\mathrm{p}}^{\sim}$, are asynchronous variants of the calculus, i.e., output actions are not allowed to guard a process different from $\mathsf{0}$.

Monadic Variants:

	$+_m$	$+_s$	$=$	$y_1 \cdot y_2$	$\xrightarrow{\mu}$	$\longmapsto$
π_{m}	×	×	–	–	2.2	2.3
π_{s}	–	×	–	–	2.2	2.3
$\pi_{\mathrm{s}}^{=}$	–	×	×	–	2.2	2.3
π_{a}	–	–	–	–	–	2.3
$\pi_{\mathrm{a}}^{=}$	–	–	×	–	–	2.3
π_{p}	–	–	–	×	–	2.3

Polyadic Variants:

	$+_m$	$+_s$	$=$	$y_1 \cdot y_2$	$\longmapsto$
$\pi_{\mathrm{m}}^{\sim}$	×	×	–	–	2.4
$\pi_{\mathrm{s}}^{\sim}$	–	×	–	–	2.4
$\pi_{\mathrm{a}}^{\sim}$	–	–	–	–	2.4
$\pi_{\mathrm{a}}^{=,\sim}$	–	–	×	–	2.4
$\pi_{\mathrm{p}}^{\sim}$	–	–	–	×	2.4

Figure 2.5.: Variants of the Pi-Calculus.

2.1.2. The Join-Calculus

Now, we introduce the join-calculus as described e.g. in [FG96, Fou98].

Definition 2.1.9 (J)**.** The set of process terms of the *join-calculus*, denoted by $\mathcal{P}_{\mathrm{J}}$, is given by

$$P ::= \mathsf{0} \;\mid\; y\langle z\rangle \;\mid\; P_1 \mid P_2 \;\mid\; \mathsf{def}\, D \,\mathsf{in}\, P \;\mid\; \checkmark$$
$$J ::= y(x) \;\mid\; J_1 \mid J_2 \quad \text{and} \quad D ::= J \triangleright P \;\mid\; D_1 \wedge D_2$$

for some names $x, y, z \in \mathcal{N}$.

The interpretation is again as usual. $\mathsf{0}$, $y\langle z\rangle$, and $P_1 \mid P_2$ define the empty process, an output capability, and parallel composition similar to π_{a}. A *definition* $\mathsf{def}\, D \,\mathsf{in}\, P$ defines a new receiver on fresh names, where D consists of one or several elementary definitions $J \triangleright P$ connected by $\wedge$, J potentially joins several reception patterns $y(x)$ connected by $\mid$, and P is a process. Compared to the pi-calculus, join patterns represent (recurrent) input capabilities that are matched against outputs in order to instantiate and unguard an instance of a guarded subterm. Note that the definition construct $\mathsf{def}\, D \,\mathsf{in}\, P$ unifies the concepts of restriction, input capabilities, and replication of the pi-calculus. In $\mathsf{def}\ (J_1 \triangleright P_1) \wedge \ldots \wedge (J_n \triangleright P_n) \,\mathsf{in}\, P$ the subterms $P_1, \ldots, P_n$ are guarded while P is an unguarded subterm. Again, we omit an action's object when it does not effectively contribute to the behaviour of a term, e.g. we write $\mathsf{def}\, y(x) \triangleright \mathsf{0} \,\mathsf{in}\, y\langle z\rangle$ as $\mathsf{def}\, y \triangleright \mathsf{0} \,\mathsf{in}\, y$.

The sets of *received variables* $\mathsf{rv}(\cdot)$ and *defined variables* $\mathsf{dv}(\cdot)$ are inductively defined as:

$$\begin{array}{rclcrcl} \mathsf{rv}(y\,(x)) & \triangleq & \{\,x\,\} & & \mathsf{rv}(J_1 \mid J_2) & \triangleq & \mathsf{rv}(J_1) \uplus \mathsf{rv}(J_2) \\ \mathsf{dv}(y\,(x)) & \triangleq & \{\,y\,\} & & \mathsf{dv}(J_1 \mid J_2) & \triangleq & \mathsf{dv}(J_1) \cup \mathsf{dv}(J_2) \\ \mathsf{dv}(J \triangleright P) & \triangleq & \mathsf{dv}(J) & & \mathsf{dv}(D_1 \wedge D_2) & \triangleq & \mathsf{dv}(D_1) \cup \mathsf{dv}(D_2) \end{array}$$

By convention, the received variables of composed join patterns have to be pairwise distinct. The bound names $\mathsf{bn}(P)$ of P are the union of the received and defined variables in P. The free names of P are defined by its set of *free variables*, where $\mathsf{fv}(\cdot)$:

$$\begin{array}{rcl} \mathsf{fv}(J \triangleright P) & \triangleq & \mathsf{dv}(J) \cup (\mathsf{fv}(P) \setminus \mathsf{rv}(J)) \\ \mathsf{fv}(D_1 \wedge D_2) & \triangleq & \mathsf{fv}(D_1) \cup \mathsf{fv}(D_2) \\ \mathsf{fv}(y\,\langle z\rangle) & \triangleq & \{\,y, z\,\} \\ \mathsf{fv}(\mathsf{def}\ D\ \mathsf{in}\ P) & \triangleq & (\mathsf{fv}(P) \cup \mathsf{fv}(D)) \setminus \mathsf{dv}(D) \\ \mathsf{fv}(P_1 \mid P_2) & \triangleq & \mathsf{fv}(P_1) \cup \mathsf{fv}(P_2) \end{array}$$

Moreover, [FG96, Fou98] define the *core join-calculus* cJ as a subcalculus of J that restricts definitions to the form $\mathsf{def}\ y_1\,(x_1) \mid y_2\,(x_2) \triangleright P_1\ \mathsf{in}\ P_2$, i.e., in the core join-calculus definitions consist of a single elementary definition of exactly two reception patterns.

The operational semantics of the join-calculus is given by an extension of the chemical approach in [BB90]. The rules operate on so-called solutions $\mathcal{R} \vdash \mathcal{M}$, where $\mathcal{R}$ and $\mathcal{M}$ are multisets. As done in [FG96], we only mention the elements of the multisets that participate in the rule, separated by commas. The semantics is given by the so-called heating and cooling rules

$$\begin{array}{lcccccc} \textsc{Join}_\textsc{j} & & \vdash\ P \mid Q & \rightleftharpoons & & \vdash\ P, Q \\ \textsc{And}_\textsc{j} & D \wedge E & \vdash & \rightleftharpoons & D, E & \vdash \\ \textsc{Def}_\textsc{j} & & \vdash\ \mathsf{def}\ D\ \mathsf{in}\ P & \rightleftharpoons & \sigma_{dv}(D) & \vdash\ \sigma_{dv}(P) \end{array}$$

and the reduction rule

$$\textsc{Red}_\textsc{j} \quad J \triangleright P \ \vdash\ \sigma_{rv}(J) \quad \longmapsto \quad J \triangleright P \ \vdash\ \sigma_{rv}(P)$$

where σ_{dv} instantiates the defined variables in D to distinct fresh names, and σ_{rv} substitutes the transmitted names for the distinct received variables. Note that the heating and cooling rules describe the underlying structural congruence on processes, i.e., if $P \rightleftharpoons Q$, $Q \longmapsto Q'$, and $Q' \rightleftharpoons P'$ then also $P \longmapsto P'$. In the following, we write $P \equiv Q$ if P and Q differ only by applications of the heating and cooling rules.

2.1.3. Communicating Sequential Processes (CSP)

The language CSP was introduced by Hoare [Hoa78, Hoa04]. We consider two variants of CSP that (instead of arbitrary action events) use in- and output prefixes. The first variant $\mathrm{CSP}_{\mathrm{in}}$ allows input guards in the choice construct.

Definition 2.1.10 ($\mathrm{CSP}_{\mathrm{in}}$)**.** The set of process terms of the CSP-*calculus with input guarded choice*, denoted by $\mathcal{P}_{\mathrm{in}}$, is given by

$$\begin{aligned} P &::= \mathrm{STOP} \quad | \quad P \setminus n \quad | \quad P_1 \parallel P_2 \quad | \quad y^\star?\,(x) \to P \quad | \quad \checkmark \\ &\quad | \quad y!z \to P \quad | \quad [\,C\,] \\ C &::= G \quad | \quad G \;\Box\; C \quad \text{and} \quad G ::= y?\,(x) \to P \quad | \quad \tau \to P \end{aligned}$$

for some names $n, x, y, z \in \mathcal{N}$.

$P \setminus n$ restricts the name n to P. $P_1 \parallel P_2$ places its subterms in parallel. The process $y!z \to P$ first sends a value z over y and then behaves as P. By convention, the prefix operator $\to$ is right associative. $[\,C\,]$ describes a choice whose branches are separated by $\Box$. In $\mathrm{CSP}_{\mathrm{in}}$ all branches of a choice are either guarded by an input prefix $y?\,(x)$ or the internal action τ. For simplicity, we define recursion as input guarded replication $y^\star?\,(x) \to P$ similar to the pi-calculus. However, this decision does not influence the following results.

The capabilities and guards are similar to the pi-calculus. Also the definitions of free and bound names are standard and similar to the pi-calculus. Again, we sometimes omit an action's object when it does not effectively contribute to the behaviour of a term, e.g. as in $y?\,(x) \to \mathrm{STOP}$, which would be written as $y? \to \mathrm{STOP}$.

The second variant of CSP that we consider, is a subcalculus of $\mathrm{CSP}_{\mathrm{in}}$ that allows only for internal choice.

Definition 2.1.11 ($\mathrm{CSP}_{\mathrm{no}}$)**.** The set of process terms of the CSP-*calculus with only internal choice*, denoted by $\mathcal{P}_{\mathrm{no}}$, is given by

$$\begin{aligned} P &::= \mathrm{STOP} \quad | \quad P \setminus n \quad | \quad P_1 \parallel P_2 \quad | \quad y^\star?\,(x) \to P \quad | \quad \checkmark \\ &\quad | \quad y!z \to P \quad | \quad y?\,(x) \to P \quad | \quad [\,C\,] \\ C &::= G \quad | \quad G \;\Box\; C \quad \text{and} \quad G ::= \tau \to P \end{aligned}$$

for some names $n, x, y, z \in \mathcal{N}$

The operational semantics and structural congruence of $\mathrm{CSP}_{\mathrm{in}}$ and $\mathrm{CSP}_{\mathrm{no}}$ can be derived from [Hoa04]. In contrast to communications in the pi-calculus, where communication is always between exactly one input and one output guarded process, communication steps in CSP reduce a single output guarded process and arbitrarily many input guarded terms. Moreover, to perform a communication step, all top-level parallel components have to participate in this communication. Interestingly, this communication mechanism in CSP leads to a separation result in Section 4.3.3, while [Nes00] presents a good an distributability-preserving encoding between the respective counterparts π_{s} without output guarded sums and π_{a} in the pi-calculus.

2.2. Bisimulation

In this section we introduce *bisimulation*, because it is the most studied form of behavioural equivalence for processes [San09]. Bisimulation is defined coinductively and

represents not only a widely applicable equivalence but also a proof method. Within this thesis bisimulation is in particular used to prove the quality of an encoding, i.e., to establish positive translational results. Since the considered encodability results are on variants of the pi-calculus we review some common notions of bisimulation in the pi-calculus. An introduction to bisimulations in the pi-calculus can be found e.g. in [MPW92] or [SW01].

2.2.1. Bisimulation and Coupled Simulation in the Pi-Calculus

[MPW92] introduces bisimulation as a technique to compare pi-calculus terms. Intuitively, a process *simulates* another process if it simulates all its transitions such that the resulting derivatives remain in the simulation. A simulation is then called a *bisimulation* if also its inverse is a simulation.

Definition 2.2.1 (Strong Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong simulation* if $(P, Q) \in \mathcal{S}$ implies that for all $P' \in \mathcal{P}_{\mathrm{m}}$ and $\mu \in \mathcal{A}_\tau$ such that $P \xrightarrow{\mu} P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *strongly bisimilar*, denoted as $P \sim Q$, if they are related by some strong bisimulation.

As explained in Section 2.1.1, the label τ represents an internal action. Accordingly, one may want to abstract from internal actions while comparing process terms. The resulting relation is denoted as weak bisimulation.

Definition 2.2.2 (Weak Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak simulation* if $(P, Q) \in \mathcal{S}$ implies that for all $P' \in \mathcal{P}_{\mathrm{m}}$ and $\mu \in \mathcal{A}_\tau$ such that $P \xrightarrow{\mu} P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are weak simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *weakly bisimular*, denoted as $P \approx Q$, if they are related by some weak bisimulation.

In [PS92] an alternative to bisimulation, denoted as *coupled simulation*, is proposed. In contrast to strong or weak bisimulation it allows an internal choice to be distributed onto several internal choices. [vG93, PS94] extend coupled simulation to cover also divergent processes.

Definition 2.2.3 (Coupled Simulation)**.** A *mutual simulation* is a pair $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ and $\mathcal{S}_2^{-1}$ are weak simulations. A mutual simulation $(\mathcal{S}_1, \mathcal{S}_2)$ is a *coupled simulation* if

- for all $(P, Q) \in \mathcal{S}_1$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \Longrightarrow Q'$ and $(P, Q') \in \mathcal{S}_2$, and
- for all $(P, Q) \in \mathcal{S}_2$ there is some $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \Longrightarrow P'$ and $(P', Q) \in \mathcal{S}_1$.

Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *coupled similar*, denoted as $P \leftrightarrows Q$, if they are related by both components of some coupled simulation.

Already [Nes96, NP00, Nes00] use coupled simulation to reason about encodings of choice. We do alike in Section 6.3.4, although we use coupled simulation in a much more restricted manner. In fact, due to a different setting to measure quality of an encoding, coupled simulation is not crucial for the correctness of the considered encodings.

2.2.2. Observables and Barbed Bisimulation in the Pi-Calculus

We observe that the bisimulations introduced in the last section rely on a labelled semantics. On the other side, we define a labelled semantics only for two of the considered variants of the pi-calculus, namely π_{m} and π_{s}. The reason is that the general framework of quality criteria for encodings (Section 3.3) that we use throughout this thesis relies on reduction semantics. Reduction semantics is easier in the context of encodings, because its abstraction from labels allows to compare calculi that do not agree in the nature of observable behaviour. If we change the use of labelled semantics in the definition of (strong/weak) bisimilarity above, we obtain (strong/weak) reduction bisimulation. Reduction bisimulations are for example discussed in [SW01].

Definition 2.2.4 (Strong Reduction Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong reduction simulation* if $(P, Q) \in \mathcal{S}$ implies that for all $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \longmapsto P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong reduction bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong reduction simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *strong reduction bisimilar*, denoted as $P \leftrightarrow Q$, if they are related by some strong reduction bisimulation.

Unfortunately, weak reduction bisimulation as defined below is trivial, i.e., does not distinguish any process terms.

Definition 2.2.5 (Weak Reduction Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak reduction simulation* if $(P, Q) \in \mathcal{S}$ implies that for all $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \longmapsto P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \Longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak reduction bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are weak reduction simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *weak reduction bisimilar*, denoted as $P \Leftrightarrow Q$, if they are related by some weak reduction bisimulation.

To obtain again useful notions of bisimulation we have to replace the lost information on the labels by some notion of *observable* or *barb*. The standard observables in the pi-calculus are its unguarded and not restricted output and input capabilities, where in the case of an asynchronous variant of the pi-calculus usually only output capabilities are considered [Hon92b, HY95, ACS98, SW01].

Definition 2.2.6 (Observables)**.** Let $P \in \mathcal{P}_{\mathrm{m}}$. Then P has an *output observable* y, denoted as $P \downarrow_{\overline{y}}$, if $y \in \mathsf{fn}(P)$ and P contains an unguarded output on channel y, and P has an *input observable* y, denoted as $P \downarrow_y$,if $y \in \mathsf{fn}(P)$ and P contains an unguarded input on channel y.

Moreover, let $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Then P *reaches an observable* μ, denoted as $P \Downarrow_\mu$, if there exists some $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \Longmapsto P'$ and $P' \downarrow_\mu$.

Augmenting reduction bisimulation with a requirement on the reachability of these observables result in *barbed bisimulation*. Barbed bisimulation was introduced in [MS92] and can be considered as one of the standard equivalences of the pi-calculus.

Definition 2.2.7 (Strong Barbed Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong barbed simulation* if $(P, Q) \in \mathcal{S}$ implies that

1. $P \downarrow_{\mu}$ implies $Q \downarrow_{\mu}$ for all $\mu \in \mathcal{A}$ and
2. for all $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \longmapsto P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *strong barbed bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are strong barbed simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *strong barbed bisimilar*, denoted as $P \dot{\sim} Q$, if they are related by some strong barbed bisimulation.

Weak barbed bisimulation is obtained straightforwardly.

Definition 2.2.8 (Weak Barbed Simulation and Bisimulation)**.** A relation $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak barbed simulation* if $(P, Q) \in \mathcal{S}$ implies that

1. $P \downarrow_{\mu}$ implies $Q \Downarrow_{\mu}$ for all $\mu \in \mathcal{A}$ and
2. for all $P' \in \mathcal{P}_{\mathrm{m}}$ such that $P \longmapsto P'$ there is some $Q' \in \mathcal{P}_{\mathrm{m}}$ such that $Q \Longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

A relation $\mathcal{B} \subseteq \mathcal{P}_{\mathrm{m}} \times \mathcal{P}_{\mathrm{m}}$ is a *weak barbed bisimulation* if both $\mathcal{B}$ and $\mathcal{B}^{-1}$ are weak barbed simulations. Two processes $P, Q \in \mathcal{P}_{\mathrm{m}}$ are *weak barbed bisimilar*, denoted as $P \dot{\approx} Q$, if they are related by some weak barbed bisimulation.

3. Encodings and their Quality

Language comparison by means of encodings is a wide area of research within the context of process calculi. Reasonable and meaningful encodings from one language into another shows that the latter is at least as expressive as the former, whereas the absence of such an encoding shows that the former can express some behaviour that is not expressible in the latter, i.e., reveals a difference in the expressive power of the former compared to the latter language. However, as stated several times in literature (e.g. in [Pal03, Nes06, Par08, Gor10b]), there is no agreement on what set of criteria makes an encoding reasonable and meaningful.

Sometimes it is even stated that such an agreement may not exist or may not be desirable (see e.g. [Pal03]), because many criteria result from different practical needs. Indeed it is obviously no trivial task to decide on the quality criteria of translational results. They are often derived from the main purpose of the current analysis. From a practical point of view this is meaningful. But, obviously, using different quality criteria for different results, because they were motivated by different practical problems, naturally leads to incomparable results. To circumvent this problem, [Gor10b] as well as [FL10] propose a general framework. Like them, we believe that a general framework does exist and is meaningful at least as a core framework to compare languages and to build hierarchies. Note that particularly the last requirement, i.e., the construction of hierarchies, significantly benefits from a general framework.

In Section 3.1 we shortly formalise what we understand as an encoding. Then we review common quality criteria in Section 3.2. In Section 3.3 we shortly revisit the general framework presented in [Gor10b] that is used then throughout this thesis. To cover the need for domain-specific analysis, we consider the extension of this framework in Section 3.4 by an additional domain-specific criterion, which is then used in the following chapters to obtain positive as well as negative translational results.

3.1. Encoding Functions

For us, an *encoding* is simply a function from process terms of the source language into the process terms of the target language, i.e., it is a mapping of syntax.

Definition 3.1.1 (Encoding)**.** Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, denoted as *source* and *target language*. An *encoding* from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ is a function $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ from the process terms $\mathcal{P}_{\mathrm{S}}$ of the source language $\mathcal{L}_{\mathrm{S}}$ into process terms (of a subset of) $\mathcal{P}_{\mathrm{T}}$ of the target language $\mathcal{L}_{\mathrm{T}}$.

In Chapter 5 we introduce some encoding functions between different process calculi. To distinguish between different encodings we use different super- and subscripts.

Thereby, the superscript usually refers to the source and the subscript to the target language. However, whenever we reason about arbitrary encodings as well as to define properties of arbitrary encodings we abandon super- and subscribts as in the definition above. We also use the blank version $[\![\cdot]\!]$ to prove separation results in Chapter 4, to mirror that there is no encoding function with the required properties, respectively. Moreover, we often use $S, S', S_1, \ldots$ to range over terms of the source language and $T, T', T_1, \ldots$ to range over terms of the target language of some encoding.

Note that encodings often translate single source term steps into a sequence or pomset of target term steps. We call such a sequence or pomset an *emulation* of the corresponding source term step. The co-domain of an encoding is usually smaller than the set of terms of the target language. With *target terms* we denote the set of process terms that are reachable from the encoded source terms modulo structural congruence and reduction steps.

Definition 3.1.2 (Target Terms)**.** Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, and $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ be an encoding from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$. The set of *target terms* of $[\![\cdot]\!]$, denoted by $\mathcal{P}_{\mathrm{T}}{\restriction_{[\![\cdot]\!]}}$, is the set of terms reachable from encoded source terms, i.e.,

$$\mathcal{P}_{\mathrm{T}}{\restriction_{[\![\cdot]\!]}} \triangleq \{\, T \mid \exists S \in \mathcal{P}_{\mathrm{S}} \,.\, [\![S]\!] \equiv_{\mathrm{T}} T \vee [\![S]\!] \Longmapsto_{\mathrm{T}} T \,\},$$

where $\equiv_{\mathrm{T}}$ is the structural congruence of $\mathcal{L}_{\mathrm{T}}$.

Obviously, $\mathcal{P}_{\mathrm{T}}{\restriction_{[\![\cdot]\!]}} \subseteq \mathcal{P}_{\mathrm{T}}$.

3.2. Quality Criteria

So far, an encoding function is simply a function from the processes of the source language into the processes of the target language. Obviously, such functions can always be obtained by doing some trivial mappings. For instance, between each pair of process calculi we find the encoding that translates everything to the empty process, because we assume that the empty process is part of every process calculus. Of course, such an encoding tell us nothing about the expressive power of the considered calculi. Hence, to analyse the quality of encodings and also to rule out trivial or meaningless encodings, encodings are augmented with a set of quality criteria.

Discussions of often used quality criteria can e.g. be found in [Nes96, Nes00, VPP07, Par08, Gor10b, FL10]. We shortly revisit the most common criteria in the following subsections. Note that apart from general criteria as the criteria described in the following, sometimes also domain-specific or problem-specific criteria are used. Domain-specific criteria do not consider expressivity in general but the expressive power of a language with respect to some specific domain. An example is the requirement on the homomorphic translation of the parallel operator that is used to ensure that an encoding preserves the degree of distribution of source terms [Nes00, Pal03]. We discuss this issue in Section 3.4. Problem-specific criteria are criteria designed in order to enable a separation result with respect to a particular problem. For this, usually a problem is identified

that can be solved in one but not the other language and then the least set of criteria is used that ensures that a solution of the problem in the source language is translated into a solution in the target language [VPP07]. This leads to problem-driven criteria [Gor10b]. As an example consider the side condition on the requirement of substitution preservation in [Pal03]. It ensures that leader election can be used to distinguish π_{m} and π_{s}. We discuss this result in Section 4.1.2.

Throughout the following subsections assume a source language $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$, a target language $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$, and an encoding function $[\![\, \cdot \,]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ from the source into the target language. Moreover, note that some of the quality criteria require the preservation or reflection of some condition. An encoding *preserves* some condition, e.g. some predicate $\mathsf{P}(\cdot)$, if, for all source terms that satisfy this condition, the encoded terms also satisfy this condition, i.e., $\mathsf{P}(S)$ implies $\mathsf{P}([\![\, S \,]\!])$ for all $S \in \mathcal{P}_{\mathrm{S}}$. Reflection is the corresponding counterpart of preservation for the opposite direction. Accordingly, an encoding *reflects* some condition, e.g. some predicate $\mathsf{P}(\cdot)$, if, for all target terms that satisfy this condition, the corresponding source terms also satisfy this condition, i.e., $\mathsf{P}([\![\, S \,]\!])$ implies $\mathsf{P}(S)$ for all $S \in \mathcal{P}_{\mathrm{S}}$.

3.2.1. Equivalence

The least debatable criterion is the direct comparison of the source and the target language by a behavioural equivalence. This criterion requires that

$$\forall S \in \mathcal{P}_{\mathrm{S}} \quad [\![\, S \,]\!] \asymp S$$

holds, for some behavioural equivalence $\asymp \subseteq (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}}) \times (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}})$ that is defined in exactly the same way on the source and the target language [Par08]. Intuitively, it is required that the encoding preserves and reflects the semantics of the source modulo the chosen equivalence. Obviously, the quality of the encoding directly depends on $\asymp$. A stricter such equivalence leads to stricter requirements on the encoding function. It is also very clear in this case under which circumstances two different results can be compared. If both results are proven with respect to the same equivalence then the results can be compared directly. If one of the considered equivalence is strictly weaker then the results can be compared with respect to the weaker equivalence. Else, if the equivalences are incomparable, also the results are incomparable. Hence, a language $\mathcal{L}_1$ can be considered as strictly weaker than the language $\mathcal{L}_2$ with respect to $\asymp$, if there is an encoding from $\mathcal{L}_1$ into $\mathcal{L}_2$ that satisfies the above requirement but there is no such encoding from $\mathcal{L}_2$ into $\mathcal{L}_1$.

Of course, there may be still some debate on how to choose the equivalence $\asymp$. For instance neither the identity nor $\asymp = (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}}) \times (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}})$ are intuitively meaningful choices. Moreover, as shown in [vG01, vG93] there are usually very many potential candidates. However, since the choice of the equivalence directly monitors the requirements on the encoding function, this problem is not that serious. There are, however, two serious drawbacks of this criterion. The first is that the requirement that $\asymp \subseteq (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}}) \times (\mathcal{P}_{\mathrm{S}} \cup \mathcal{P}_{\mathrm{T}})$ is defined in exactly the same way on the source and the

target language is in general a very severe requirement. Indeed, this side condition usually significantly limits the set of potential equivalences, at least with respect to the set of well-understood standard equivalences in the source and the target. Since the quality of the encoding is directly related to this equivalence, the criterion should be instantiated with the strictest candidate possible. But, because of e.g. different standard observables in the source and target language, this equivalence (if it can be constructed at all) will usually be very complex and unreadable. As a consequence, if $\asymp$ is not a standard equivalence, the direct comparison of source and target terms modulo $\asymp$ reveals very less intuition on the encoding function. The second problem is that a complex equivalence $\asymp$ leads to a hard proof of this criterion. If $\asymp$ is not a standard equivalence, most of the standard techniques that would ease such a proof may not be applicable. Moreover, since there is only one criterion, the whole complexity of the quality proof is imposed to a single proof.

Hence, this criterion is very well suited if it is proven with respect to a standard equivalence. Else, it may reveal very less intuition and very less guidance for the proof of encodability or separation as well as the design of the encoding function.

3.2.2. Full Abstraction

Whenever source and target can not be compared directly with respect to a standard equivalence, full abstraction might be a way to use nonetheless standard equivalences. *Full abstraction*—denoted as *observational correspondence* in [FL10]—is probably the most common quality criterion for language comparison. It is used for instance in [San94, Yos96, NP00, BPV05], just to name some. Full abstraction as proof method for language comparison was adapted from the use of full abstraction to show correspondence between a denotational semantics of a program and its operational semantics. An encoding $[\![\cdot]\!]$ is fully abstract if

$$\forall S_1, S_2 \in \mathcal{P}_{\mathrm{S}} \,.\quad S_1 \asymp_S S_2 \quad \text{iff} \quad [\![S_1]\!] \asymp_T [\![S_2]\!]$$

for two behavioural equivalences $\asymp_S \subseteq \mathcal{P}_{\mathrm{S}} \times \mathcal{P}_{\mathrm{S}}$ and $\asymp_T \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$, i.e., full abstraction requires that equivalent source terms have to be mapped into equivalent target terms and vice versa. Note that the direction from the left to the right is often called soundness condition and the only if part completeness condition of full abstraction. The soundness condition is usually the most demanding part. Note that some well-known and widely accepted encodings, as e.g. [Bou92, HT91, Mil92, Mil93b], do not satisfy this property with respect to a reasonable combination of standard equivalences. The main advantage of full abstraction is its wide applicability also with respect to (more or less) standard equivalences. It does e.g. not require that source and target share any notion of observable, which is a premise for the use of most of the standard equivalences in the criterion above. However, again there may be a very large number of equivalences on the source as well as equivalences on the target and the strictness of the property expressed by full abstraction strongly relies on the combination of the chosen equivalences. To reduce the strong dependence of full abstraction results on the chosen equivalences, full abstraction is often combined with operational correspondence. In [FL10] it is even stated that full

abstraction is not of much use without operational correspondence. Because of the various possibilities two choose these two equivalences, it is often not possible to compare different full abstraction results, which is a major drawback of this criterion.

3.2.3. Operational Correspondence

Intuitively, *operational correspondence* requires preservation and reflection of executions. Again, it consists of a completeness a soundness part. The completeness condition, also called adequacy, requires that for all source term steps $S \longmapsto_{\mathrm{S}} S'$ or source term executions $S \Longmapsto_{\mathrm{S}} S'$ there is one emulating execution in the target language such that $[\![S]\!] \Longmapsto_{\mathrm{T}} \asymp_T [\![S']\!]$, where $\asymp_T \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$ is some equivalences on the target language. Note that there is no difference in the consideration of single source term steps or source term executions. Intuitively, the completeness condition requires that any source term execution is emulated by the target term modulo some equivalence $\asymp_T$. Again, completeness is usually the easiest part.

For the soundness condition we basically find two formulations. The stricter formulation requires that for all executions of the target $[\![S]\!] \Longmapsto_{\mathrm{T}} T$ there exists some execution of the source $S \Longmapsto_{\mathrm{S}} S'$ such that $[\![S']\!] \asymp_T T$. Intuitively, soundness requires that whatever $[\![S]\!]$ can do is a translation of some behaviour of S modulo $\asymp_T$ [FL10]. The weaker formulation requires that for all executions of the target $[\![S]\!] \Longmapsto_{\mathrm{T}} T$ there exists some execution of the source $S \Longmapsto_{\mathrm{S}} S'$ and some execution of the target $T \Longmapsto_{\mathrm{T}} T'$ such that $[\![S']\!] \asymp_T T'$. Intuitively, it states that any execution of the target is some part of the emulation of an execution in the source modulo $\asymp_T$ [Par08, Gor10b]. The main difference is that the later formulation allows for intermediate or partial commitment states, i.e., for states that do not need to be related directly to the states of the respective source term but that have to belong to some emulation of a source term step. In this sense, an intermediate state results from the partial emulation of a source term step. We discuss this issue in Section 6.3.1.

Again different variants of operational correspondence may arise from different requirements on the assumed equivalence $\asymp_T$ on the target language. Note that [Nes96, NP00] present operational correspondence without the equivalence, i.e., require $[\![S]\!] \Longmapsto_{\mathrm{T}} [\![S']\!]$ whenever $S \longmapsto_{\mathrm{S}} S'$ and $[\![S]\!] \Longmapsto_{\mathrm{T}} T$ implies $S \Longmapsto_{\mathrm{S}} S'$ for some S' such that $T \Longmapsto_{\mathrm{T}} [\![S']\!]$, which again leads to a stricter formulation than above. They also present a stricter variant of the soundness part—$[\![S]\!] \longmapsto_{\mathrm{T}} T$ implies $S \longmapsto_{\mathrm{S}} S'$ for some S' such that $T \asymp_T [\![S']\!]$—and state that only prompt encodings can satisfy this stricter variant.

Moreover, in [FL10] labelled steps are considered instead of a reduction semantics under the assumption that there exists a mapping $\widehat{\cdot}$ from the labels of the source term into the labels of the target term. Hence, the resulting requirement—$[\![S]\!] \overset{\widehat{\lambda}}{\Longrightarrow} \asymp_T [\![S']\!]$ whenever $S \overset{\lambda}{\Longrightarrow} S'$ and $[\![S]\!] \overset{\lambda}{\Longrightarrow} T$ implies $S \overset{\lambda'}{\Longrightarrow} S'$ for some λ', S' such that $[\![S']\!] \asymp_T T$ and $\widehat{\lambda'} = \lambda$—can be considered as stricter than the above variant of operational correspondence, because also observables have to be preserved and reflected in some sense. In fact, without this strengthening to labelled semantics, operational correspondence

alone can hardly be considered as suitable criterion. Hence, the above version based on reduction semantics is usually combined with other criteria as full abstraction or some requirements on the preservation or reflection of some kind of observable.

3.2.4. Observables, Testing, and Termination

If source and target terms can not be compared directly by a standard equivalence, e.g. because not all standard observables of the source are standard observables of the target, a natural weaker requirement is to consider preservation or reflection of the remaining observables that are shared by source and target. Typical observables are links used for communication, barbs (communication capabilities), or traces [VPP07, Par08]. Moreover, the use of *termination properties* as the possibility of deadlock, livelock, or divergence are popular [Nes00, Par08, Gor10b, FL10]. Also all kind of *tests* that the process may (or must) pass, for some formal notion of test can be used to compare source and target behaviour [Par08, Gor10b].

Another kind of termination property is the above mentioned promptness condition. Intuitively, promptness ensures that an encoding does not introduce preprocessing steps.

Definition 3.2.1 (Promptness)**.** An encoding $[\![\cdot]\!]$ from $\langle \mathcal{P}_S, \longmapsto_S \rangle$ into $\langle \mathcal{P}_T, \longmapsto_T \rangle$ is *prompt*, if $S \mapsto\!\!\!\!\!\!\not\quad_S$ implies $[\![S]\!] \mapsto\!\!\!\!\!\!\not\quad_T$ for all source terms $S \in \mathcal{P}_S$.

Within this thesis we consider the reflection of deadlock as well as divergence as a good requirements for encodability and separation results. In the style of [Gor10b] we also compare source terms and their encodings with respect to the reachability of success. Since ✓ can not be further reduced and $\mathsf{n}(\checkmark) = \mathsf{fn}(\checkmark) = \mathsf{bn}(\checkmark) = \emptyset$, the semantics and structural congruence of a process calculus are not affected by this additional constant operator. The test for reachability of success is standard.

Definition 3.2.2 (Success)**.** Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus and $\equiv$ its structural congruence. A process $P \in \mathcal{P}$ *may lead to success* or *may-succeeds*, denoted as $P \Downarrow_{\checkmark}$, if it is reducible to a process containing a top-level unguarded occurrence of ✓, i.e., if $P \Longmapsto P' \wedge P' \equiv P'' \mid \checkmark$ for some P', P''.

Moreover, we write $P \Downarrow_{\checkmark!}$, if P reaches success in every finite maximal execution.

A process $P \in \mathcal{P}$ *must lead to success* or *must-succeeds*, denoted as $P \Downarrow\!\!\Downarrow_{\checkmark}$, if it reduces to a process containing a top-level unguarded occurrence of ✓ in every maximal execution.

Note that ✓ can be considered as some kind of termination property—describing successful termination in contrast to not successful termination by $\mathsf{0}$—or as the successful pass of some kind of test.

3.2.5. Structural Requirements

The above discussed criteria describe semantic requirements, i.e., requirements on the behaviour of target terms. To prove the quality of an encoding semantic criteria are often combined with structural criteria. Intuitively, the semantic criteria describe how

the encoded terms should behave with respect to the behaviour of the corresponding source term, whereas structural criteria rather describe how the encoded terms have to look like. Moreover, as stated in [Par08], structural criteria are needed in order to measure expressiveness of operators in contrast to expressiveness of terms.

The most common structural criterion is compositionality with homomorphy as a special case. Intuitively, *compositionality* states that the translation of a compound term must be defined in terms of the translation of the subterms. To mediate between translations of subterms, a context is introduced. Different manifestations result from different requirements on allowed contexts. In the strictest form, often denoted as *homomorphy*, the context has to be the original source term operator again, i.e., an encoding translates the source term operator $\mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)$ homomorphically if it ensures that $[\![\ \mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)\ ]\!] = \mathrm{op}\,(x_1, \ldots, x_n, [\![\ S_1\ ]\!], \ldots, [\![\ S_m\ ]\!])$ holds for all $x_1, \ldots, x_n \in \mathcal{N}$ and all $S_1, \ldots, S_m \in \mathcal{P}_\mathrm{S}$. Of course, this requires that the respective operator is part of the source and the target language. Because of this, homomorphy is often required only for the parallel operator, because it occurs more or less with the same meaning and comparable syntax in most of the process calculi. Here, we require (like [Gor10b]) that the parallel operator is a binary operator in every process calculus. Hence, the homomorphic translation of the parallel operator means $[\![\ S_1 \mid S_2\ ]\!] = [\![\ S_1\ ]\!] \mid [\![\ S_2\ ]\!]$ for all $S_1, S_2 \in \mathcal{P}_\mathrm{S}$, where $\mid$ is the syntactical representation of the parallel operator. Homomorphic translations of operators are e.g. used to analyse the expressive power of a single operator. To show for instance that a set of operators is not minimal the existence of an encoding is analysed that translates all operators homomorphically except for the operator that should be removed. Moreover, homomorphy is a very grateful property. As discussed in Section 6.1, the homomorphic translation of some of the source term operators significantly eases the proof of the correctness of the encoding function. Basically, the homomorphic translation of an operator ensures that for this operator nothing is to show, because in this point the encoding obviously preserves and reflects all properties of that operator. However, even in case the respective operator is part of the source as well as the target language, homomorphy is a very strict requirement. Intuitively, it states that the encoding function is not allowed to touch the respective operator and, hence, is not allowed to simulate its behaviour by some protocol. Such translations are possible only if the compared languages are very close (at least with respect to this operator). In Section 4.2.1 we present some negative results in the style of [Pal03, Gor10b] to show that not even between calculi that are so close as π_m and π_s an encoding that translates the parallel operator homomorphically exists. Instead, often compositionality is required. It does not impose additional restrictions on the introduced context. [Gor10b] even allows for the context to be parametrised on the free names of the subterms. In contrast to homomorphy, compositionality is a very natural requirement. Intuitively, it states that every occurrence of an operator in the source term is treated by the encoding function in exactly the same way, i.e., is translated into the same term modulo the translation of the respective subterms. Also note that a compositional encoding, i.e., an encoding that translates all source term operators compositionally, implies that also any source term context can be represented as a context in the target language [Par08]. Moreover, compositionality guides the design of encodings, because it describes how an encoding

has to look like.

Another structural criterion is the *preservation of substitutions*, denoted as *name invariance* in [Gor10b] and as *stability* in [FL10]. It usually requires that, for all source terms $S \in \mathcal{P}_S$ and all substitutions σ on source terms, there exists some substitution σ' on target terms such that $[\![\sigma(S)]\!] \asymp_T \sigma'([\![S]\!])$ for some equivalence $\asymp_T \subseteq \mathcal{P}_T \times \mathcal{P}_T$ on target terms. Often additional requirements on the relationship between σ and σ' or on the equivalence $\asymp_T$ are stated. The strictest case is of course that $\sigma = \sigma'$ and $\asymp_T = \equiv_\alpha$. This criterion is based on the idea that names are property-less [FL10]. Hence, the preservation of substitutions should ensure that encodings of source terms that differ only in their free names can also only differ in free names (modulo the provided equivalence).

Moreover, [VPP07] present *link independence*, a condition that prevents encodings from introducing free names. More precisely, link independence means that, for all source terms $S_1, S_2 \in \mathcal{P}_S$, $\mathsf{fn}(S_1) \cap \mathsf{fn}(S_2) = \emptyset$ implies $\mathsf{fn}([\![S_1]\!]) \cap \mathsf{fn}([\![S_1]\!]) = \emptyset$.

3.3. A General Framework

In the last section we presented some criteria from a wide field of possible general requirements for an encoding function. Together with domain-specific and problem-specific criteria, and the various variants of the above criteria that result from the instantiation of the required equivalences, the domain of requirements that may turn an encoding into a good encoding is indeed huge. Unfortunately, there is no agreement on what criteria lead to reasonable translational results [Pal03, Nes06, Par08, Gor10b]. Moreover, in order to strengthen the respective kind of results, usually minimal criteria are searched for negative results and maximal criteria for positive results. Because of that, we hardly find two results from different authors that are proven with respect to exactly the same set of criteria. Obviously, this is problematic, e.g. if we want to compare different results to construct a hierarchy. Moreover, if we want to relate the expressive power of two given languages there is no guidance on how to start or whether an obtained result is sufficiently substantiated by the chosen criteria to call it reasonable.

In order to provide a general framework, Gorla in [Gor10b] suggests five criteria well suited for language comparison, i.e., for positive as well as negative translational results. In particular, this framework allows us to generate clearly organised hierarchies of languages with respect to their expressive power (compare to Section 7.2). As claimed in [Gor10b], most of the encodings appearing in the literature satisfy this framework and several known separation results can also be derived within this framework but there are also encodings that do not satisfies this framework, i.e., the framework is not trivial. Moreover, [Gor10b] presents some new separation results proved within this framework. Finally, the set of criteria is small and handy but at the same time guides the design of encoding functions and supports the proof of translational results by separating the requirements on different intuitive criteria. Accordingly, we consider an encoding to be "good" if it satisfies Gorla's five criteria. In the following, we shortly present the criteria of his general framework as presented in [Gor08b, Gor09, Gor10b].

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection*, and (5) *success sensitiveness*. Note that for the definition of name invariance and operational correspondence a behavioural equivalence $\asymp$ for the target language is assumed. Its purpose is to describe the abstract behaviour of a target process, where abstract refers to the behaviour of the source term.

Intuitively, an encoding is compositional if the translation of an operator is the same for all occurrences of that operator in a term. Hence, the translation of that operator can be captured by a context that is allowed in [Gor10b] to be parametrised on the free names of the respective source term.

Definition 3.3.1 (Criterion 1: Compositionality)**.** The encoding $[\![\cdot]\!]$ is *compositional* if, for every operator $\mathbf{op} : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{S}}$ of $\mathcal{L}_{\mathrm{S}}$ and for every subset of names N, there exists a context $\mathcal{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{T}}$ such that, for all $x_1, \ldots, x_n \in \mathcal{N}$ and all $S_1, \ldots, S_m \in \mathcal{P}_{\mathrm{S}}$ with $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$, it holds that $[\![\mathbf{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)]\!] = \mathcal{C}^N_{\mathbf{op}}(x_1, \ldots, x_n, [\![S_1]\!], \ldots, [\![S_m]\!])$.

The second structural criterion states that the encoding should not depend on specific names used in the source term. This is important, since sometimes it is necessary to translate a source term name into a sequences of names or reserve some names for the encoding function. To ensure that there are no conflicts between these reserved names and the source term names, the encoding is equipped with a renaming policy $\varphi_{[\![ ]\!]}$, i.e., a substitution from names into sequences of names. To keep distinct names distinct, Gorla assumes that the sequences of names that result from applying a renaming policy to distinct names have no common name. Moreover, if the renaming policy translates a single name into a sequence of names then the length of such a sequence has to be the same for all names, such that the encoding can not distinguish between different source term names by the length of the sequences to which they are encoded. Obviously, no name should be translated into an infinite sequence of names.

Definition 3.3.2 (Renaming Policy)**.** A substitution $\varphi_{[\![ ]\!]} : \mathcal{N} \to \mathcal{N}^n$ from names into sequences of names is a renaming policy, if

$$\forall x, y \in \mathcal{N} \,.\, x \neq y \text{ implies } \varphi_{[\![ ]\!]}(x) \cap \varphi_{[\![ ]\!]}(y) = \emptyset,$$

where $\varphi_{[\![ ]\!]}(z)$ is simply considered as a set here.

Note that the renaming policy allows us to use the names reserved by the encoding like implicit parameters. It is for instance possible that some part of the encoding introduces a free occurrence of a reserved name within the encoding of a subterm which is bound by the surrounding part of the encoding. An example can be found in Section 5.1.2. Moreover, note that in [Gor10b] an encoding is in fact a pair $\left([\![\cdot]\!]\cdot, \varphi_{[\![ ]\!]}\cdot\right)$. All encodings presented within this thesis follow this scheme and also introduce a renaming policy.

In the following, if single names are translated into single names, we sometimes extend the notion of homomorphic translation to the use of a renaming policy. Thus, if the

encoding translates an operator $\mathbf{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)$ always into

$$\mathbf{op}\,(\varphi_{[\![\,]\!]}(x_1), \ldots, \varphi_{[\![\,]\!]}(x_n), [\![\, S_1 \,]\!], \ldots, [\![\, S_m \,]\!]),$$

as in $[\![\, (\nu x)\, P \,]\!] = \big(\nu \varphi_{[\![\,]\!]}(x)\big)\, [\![\, P \,]\!]$, we call the translation of this operator again homomorphic.

If an encoding does not need a special renaming policy we use the identity function. An encoding is then independent of specific names if it preserves all substitutions σ on source terms by a substitution σ' on target terms such that σ' respects the changes made by the renaming policy.

Definition 3.3.3 (Criterion 2: Name Invariance)**.** The encoding $[\![\, \cdot \,]\!]$ is *name invariant* if, for every $S \in \mathcal{P}_{\mathrm{S}}$ and σ, it holds that

$$[\![\, \sigma\,(S) \,]\!] \begin{cases} \equiv_\alpha \sigma'\,([\![\, S \,]\!]) & \text{if } \sigma \text{ is injective} \\ \asymp \sigma'\,([\![\, S \,]\!]) & \text{otherwise} \end{cases}$$

where σ' is such that $\varphi_{[\![\,]\!]}(\sigma\,(n)) = \sigma'\,\big(\varphi_{[\![\,]\!]}(n)\big)$ for every $n \in \mathcal{N}$.

The first semantic criterion and usually the most elaborate one to prove is operational correspondence. It consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be emulated by its translation. *Soundness* requires that every computation of a target term corresponds to some computation of the corresponding source term.

Definition 3.3.4 (Criterion 3: Operational Correspondence)**.** The encoding $[\![\, \cdot \,]\!]$ satisfies *operational correspondence* if it satisfies:

Completeness:	For all $S \Longmapsto_{\mathrm{S}} S'$, it holds $[\![\, S \,]\!] \Longmapsto_{\mathrm{T}} \asymp [\![\, S' \,]\!]$.
Soundness:	For all $[\![\, S \,]\!] \Longmapsto_{\mathrm{T}} T$, there exists an S' such that $S \Longmapsto_{\mathrm{S}} S'$ and $T \Longmapsto_{\mathrm{T}} \asymp [\![\, S' \,]\!]$.

Note that the definition of operational correspondence relies on the equivalence $\asymp$ to get rid of junks possibly left over within computations of target terms (compare to Section 6.3). Sometimes, we refer to the completeness criterion of operational correspondence as *operational completeness* and, accordingly, for the soundness criterion as *operational soundness*.

The next criterion concerns the role of infinite computations in encodings.

Definition 3.3.5 (Criterion 4: Divergence Reflection)**.** The encoding $[\![\, \cdot \,]\!]$ *reflects divergence* if, for every $S \in \mathcal{P}_{\mathrm{S}}$, $[\![\, S \,]\!] \longmapsto^{\omega}_{\mathrm{T}}$ implies $S \longmapsto^{\omega}_{\mathrm{S}}$.

The last criterion links the behaviour of source terms to the behaviour of their encodings. With Gorla [Gor10b], we assume a *success* operator $\checkmark$ as part of the syntax of both the source and the target language. An encoding preserves the abstract behaviour of the source term if it and its encoding answer the tests for success in exactly the same way.

Definition 3.3.6 (Criterion 5: Success Sensitiveness)**.** The encoding $[\![\cdot]\!]$ is *success sensitive* if, for every $S \in \mathcal{P}_S$, $S \Downarrow_{\checkmark}$ iff $[\![S]\!] \Downarrow_{\checkmark}$.

Note that we choose may-testing here. However this choice is not crucial. This criterion only links the behaviours of source terms and their literal translations, but not of their derivatives. To do so, Gorla relates success sensitiveness and operational correspondence by requiring that the equivalence on the target language never relates two processes with different success behaviours.

Definition 3.3.7 (Success Respecting)**.** Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus and $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ be an equivalence. Then $\mathcal{R}$ is *success respecting* if, for every $P, Q \in \mathcal{P}$ with $P \Downarrow_{\checkmark}$ and $Q \not\Downarrow_{\checkmark}$, it holds that $(P, Q) \notin \mathcal{R}$.
$\asymp$ is a success respecting equivalence.

By [Gor10b] a "good" equivalence $\asymp$ is often defined in the form of a barbed equivalence (as described e.g. in [MS92]) or can be derived directly from the reduction semantics (as described e.g. in [HY95]) and is often a congruence, at least with respect to parallel composition. For the separation results presented in this thesis, we require only that $\asymp$ is a success respecting reduction bisimulation.

Definition 3.3.8 (Weak Reduction Bisimulation)**.** The equivalence $\asymp$ is a weak reduction bisimulation.

In this case, a good encoding respects also the ability to reach success in all finite maximal executions.

Lemma 3.3.9. *For all success respecting reduction bisimulations $\asymp \subseteq \mathcal{P}_T \times \mathcal{P}_T$ and all terms $T_1, T_2 \in \mathcal{P}_T$ such that $T_1 \asymp T_2$, it holds $T_1 \Downarrow_{\checkmark!}$ iff $T_2 \Downarrow_{\checkmark!}$.*

Proof. Let us assume the contrary, i.e., there is some success respecting bisimulation $\asymp \subseteq \mathcal{P}_T \times \mathcal{P}_T$ and two terms $T_1, T_2 \in \mathcal{P}_T$ such that $T_1 \asymp T_2$ and $T_1 \Downarrow_{\checkmark!}$ but not $T_2 \Downarrow_{\checkmark!}$. Then, for all $T_1' \in \mathcal{P}_T$ with $T_1 \Longmapsto_T T_1'$, we have $T_1' \Downarrow_{\checkmark}$ but there exists some $T_2' \in \mathcal{P}_T$ such that $T_2 \Longmapsto_T T_2'$ and $T_2' \not\Downarrow_{\checkmark}$.

Since $\asymp$ is a weak reduction bisimulation (Definition 3.3.8), $T_1 \asymp T_2$ and $T_2 \Longmapsto_T T_2'$ imply that there exists some $T_1'' \in \mathcal{P}_T$ such that $T_1 \Longmapsto_T T_1''$ and $T_2' \asymp T_1''$. Because $\asymp$ is success respecting (Definition 3.3.7), $T_2' \asymp T_1''$ and $T_2' \not\Downarrow_{\checkmark}$ imply $T_1'' \not\Downarrow_{\checkmark}$. This violates the requirement that $T_1 \Downarrow_{\checkmark!}$, i.e., contradicts the assumption that for all $T_1' \in \mathcal{P}_T$ with $T_1 \Longmapsto_T T_1'$ we have $T_1' \Downarrow_{\checkmark}$. We conclude that $T_1 \Downarrow_{\checkmark!}$ iff $T_2 \Downarrow_{\checkmark!}$. □

Moreover, in this case success sensitiveness preserves also the ability to reach success in all finite maximal executions.

Lemma 3.3.10. *For all operationally sound and success sensitive encodings $[\![\cdot]\!]$ with respect to some success respecting equivalence $\asymp \subseteq \mathcal{P}_T \times \mathcal{P}_T$ and for all $S \in \mathcal{P}_S$, if $S \Downarrow_{\checkmark!}$ then $[\![S]\!] \Downarrow_{\checkmark!}$.*

Proof. Assume the contrary, i.e., there is an encoding that satisfies the criteria operational soundness and success sensitiveness, $\asymp$ is success respecting, and there is some $S \in \mathcal{P}_S$ such that for all $S' \in \mathcal{P}_S$ with $S \Longmapsto_S S'$ we have $S' \Downarrow_{\surd}$, i.e., $S \Downarrow_{\surd!}$, but there is some $T \in \mathcal{P}_T$ such that $[\![S]\!] \Longmapsto_T T$ and $T \not\Downarrow_{\surd}$.

Since $[\![\cdot]\!]$ is operationally sound (Definition 3.3.4), $[\![S]\!] \Longmapsto_T T$ implies that there exists some $S'' \in \mathcal{P}_S$ and some $T' \in \mathcal{P}_T$ such that $S \Longmapsto_S S''$ and $T \Longmapsto_T T' \asymp [\![S'']\!]$. By Definition 3.2.2, then $T \not\Downarrow_{\surd}$ and $T \Longmapsto_T T'$ imply $T' \not\Downarrow_{\surd}$. Since $\asymp$ respects success (Definition 3.3.7), $T' \asymp [\![S'']\!]$ and $T' \not\Downarrow_{\surd}$ imply $[\![S'']\!] \not\Downarrow_{\surd}$. Because $[\![\cdot]\!]$ is success sensitive (Definition 3.3.6), then also $S'' \not\Downarrow_{\surd}$, which contradicts the assumption that $S \Downarrow_{\surd!}$. We conclude that if $S \Downarrow_{\surd!}$ then $[\![S]\!] \Downarrow_{\surd!}$. □

3.4. Designing Quality Criteria

As mentioned before we consider the above discussed framework as a core framework for language comparison. But on the other side we also see the need for domain-specific analyses. The framework of [Gor10b] is well suited to analyse the general expressive power of languages. It allows us to construct clearly separated hierarchies, guides the design of encoding functions, separates the proof of correctness of an encoding in intuitive requirements, and provides a good base to obtain negative results by using criteria from which further proof techniques can easily be derived. Hence, it is a good starting point not only for the analysis of the general expressive power. Moreover, as we see in the following, it can easily be extended by domain-specific criteria. Domain-specific criteria, as the name suggests, are used to analyse properties of a specific domain that may in general not be interesting. Hence, it is not a good idea to overload the framework by permanently adding domain-specific criteria. Instead, we add a domain-specific criterion only if it is necessary to answer a particular kind of question. Possible domains in the context of process calculi are e.g. causality, the branching time behaviour of processes, or considerations related to some special features as failures or time constrains.

Note that an additional criterion may strengthen a positive translational result, but it weakens negative translational results. Moreover, already a single additional criterion significantly complicates the comparison of a result with other already established results that do not rely on this additional criterion. Quite often, they even lead to incomparable results; in particular if two results use different additional criteria. Hence, we strongly recommend to introduce new criteria only in case they are unavoidable to answer a specific kind of question.

3.4.1. Abstract Formulation

The first step in the design of a new criterion is to fix its purpose, i.e., to carefully describe the area we want to analyse and which kind of processes we want to distinguish and why. This consideration should result in an abstract description of the new criterion and an argumentation to explain, why—and maybe also in which settings—it is meaningful to consider this additional criterion.

We want to fix the notions of distributability and preservation of distributability in the context of process calculi. Note that, as we discuss in the following, distributability is not captured by the general framework above. Intuitively, a distribution of a process means the extraction (or separation) of its (sequential) components and their association to different locations.

Systems composed from such components are called distributed systems. Due to the interplay of synchronisation, i.e., actions in form of two or multi-way rendezvous of different components, and concurrency, i.e., actions performed independently by other components, the behaviour of distributed systems is usually hard to analyse. A comparison and classification of languages with respect to their degree of distributability can help to analyse distributed concurrent systems. In particular, it reveals the class of languages suited to express concurrency (possible with respect to some specific system requirements) and, hence, the class of languages suited for system design in this area of research. Because of that, we want to extend the general framework of Section 3.3 in order to reason about the more domain-specific problem of distributability. Then we use the extended framework in Section 4.3 and Section 4.4 to compare different process calculi. In particular, we compare different variants of the pi-calculus and the join-calculus.

Most of the existing approaches that analyse the distributability of concurrent systems use special formalisms often equipped with an explicit notion of location, e.g. the distributed pi-calculus in [Hen07], a variant of the join-calculus with explicit locations in [FLMR96], or [BD12] in Petri nets. In contrast to these approaches, we analyse (similar to [vGGS08, vGGS12]) the potential of a formalism to describe distributed systems without an explicit association of locations to processes. Instead, we abstract from a particular distribution and consider *distributability* and, thus, all possible explicitly-located variants of a calculus. We do so because we consider the expressive power of languages, not just of individual terms. Moreover, we obtain results for a larger number of process calculi.

Another way is to explicitly consider the concurrent execution of independent steps directly within an operational semantics, often called *step semantics* (e.g. [Lan07] for the case of the pi-calculus), and also in the form of dedicated behavioural equivalences, consequently denoted as *step equivalences* [vG01, vG93, vGG01]. Indeed, if we want to compare source terms and their encodings directly with respect to distributability, we would need a step semantics and a branching time equivalence. However, the quality criteria in the general framework are designed to circumvent such direct comparisons of source and target terms and we follow this line for our new criterion.

Given an extension of a process calculus with an explicit notion of distribution or location we can define the degree of distribution of a process as the number of its locations. However, instead of considering locations explicitly, we just focus on the possible divisions of a process term into components. Accordingly, a process P is *distributable into* $P_1, \ldots, P_n$, if we find some distribution that extracts $P_1, \ldots, P_n$ from within P onto different locations. Preservation of distributability then means that the target term is as least as distributable as the source term. Note that the operator of process calculi that is usually associated with distribution is the parallel operator. Accordingly, we consider

components of a term that are composed in parallel as distributable.

Hence, we basically analyse the possibilities to implement the operators of a calculus or especially its parallel operator. If it is always possible to preserve the degree of distributability in an encoding of a source language into a target language which is close to an implementation e.g. in a real world scenario, then the corresponding parallel operator can be implemented in this scenario simply as the operator of distribution, i.e., parallel source terms can be implemented in distributed real world processes. If it is not possible to obtain a distributability preserving encoding, then the source language implicitly defines side conditions on the use of the parallel operator usually induced by the defined synchronisation mechanism that forbids for such simple implementations. Thus, the implementation of parallel source terms as distributed processes may be possible only under some side conditions, which are hopefully already paraphrased by the respective separation result.

3.4.2. Comparison and Classification

As next step, the current set of used criteria and also well-known criteria from the literature should be monitored, to check whether they already capture the new criterion. It is always much easier to reuse an existing criterion than to design a new one and to validate its design against the set of criteria used by others. Moreover, a comparison with existing criteria may help to classify the new criterion with respect to existing ones. Such a classification can be of great assistance for the formalisation in the next step as well as to prepare the consistency check in the last step. At least we should analyse whether our new criterion is a structural or rather a semantic criterion.

In order to measure whether an encoding respects the degree of distribution, usually the homomorphic translation of the parallel operator, i.e., $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$, is used as a criterion (see e.g. [Pal03, CM03, LV10]). Based on this requirement [Pal03] already shows a separation result between the synchronous and the asynchronous variant of the pi-calculus that we discuss in the first part of the next chapter. The homomorphic translation of the parallel operator forbids the introduction of a *global coordinator*, i.e., a centralised control instance that resolves all conflicts but at the cost of reducing the degree of distribution. Indeed, as we will observe in Section 3.4.4, the homomorphic translation of the parallel operator implies (under certain conditions) that the respective encoding preserves distributability but we will also observe that the converse is not true, i.e., there are encodings that do not translate the parallel operator homomorphically but preserve nonetheless the distributability of all source terms. In this sense, the homomorphic translation of the parallel operator is too strict—at least for separation results. It rightly forbids the introduction of coordinators that reduce the degree of distribution. But it also forbids protocols that handle communications of parallel components without sequentialising them or reducing the degree of distribution in another sense. Moreover, the homomorphic translation of the parallel operator is not always suited to reason about distribution in process calculi as, for example, the join-calculus: there, it is not always possible to separate distributable subterms by means of a parallel operator (see the discussion in the next section).

Within his general framework, Gorla requires the compositional translation of source term operators (Section 3.3). Interestingly, this requirement already prevents from the use of global coordinators. Gorla requires that the parallel operator is binary and unique for all process calculi[1]. Compositionality requires that all occurrences of a parallel operator have to be translated basically in the same way. Hence, if such an encoding introduces a coordinator then for every parallel operator a coordinator is introduced and there is no possibility to examine which of them is the outermost or to order them, i.e., it is not possible to coordinate the coordinators such that they proceed as a centralised entity. In that view, compositionality can be seen as a minimal criterion to ensure the preservation of distributability. However, sometimes—as in the current case—compositionality alone is too weak, because it still allows for *local coordinators*, i.e., a compositional encoding may still sequentialise some parts of a source term. If we are not only interested in the expressive power with respect to the abstract behaviour of terms but additionally in how far problems can be solved exploiting at least the same degree of distributability, we must consider an additional criterion.

Consequently, the preservation of distributability ranges between compositionality with respect to the parallel operator and its homomorphic translation. Both are structural criteria. Hence, one may assume that also the preservation of distributability is a structural criterion, but in fact this is not true.

A natural first condition is to require that encoded source terms are at least as distributable as the source term itself, i.e., that the degree of distributability has to be preserved by the encoding. However, it does not suffice to reason about the degree of distributability, i.e., about the number of distributable components, without additional requirements on the components. An encoding can always trivially ensure that the encoding has at least as much distributable components by introducing new subterms without any behaviour. Thus, we require that the encodings of distributable source term parts and their corresponding parts in the encoding are related by $\asymp$. By doing so we relate the definition of the preservation of distributability to operational completeness, i.e., a semantic criterion that ensures the preservation of the behaviour of the source term (part). Hence, we require that each target term part can emulate at least all behaviour of the respective source part. As a side effect, we require, that whenever a part of a source term can solve a task independently of the other parts—i.e., it can reduce on its own—then the respective part of its encoding must also be able to emulate this reduction independently of the rest of the encoded term. This reflects our intuition that distribution adds some additional requirements on the independence of parallel terms. Accordingly, we require that not only the source term and its encoding are distributable to the same degree, but also their derivatives, i.e., we do not only consider the *initial* degree of distributability. Because of that, our new criterion has both a structural as well as a semantic component.

[1] Remember that we require the same in Section 2.1.

3.4.3. Formalisation

Now, we have to formalise the new criterion. Note that this formalisation should be as general as possible, because a formalisation that is to close to a specific process calculus may hinder the derivation of similar results for other calculi and, thus, the comparison with other results. Furthermore, it is sometimes easier to define a new criterion with respect to an existing one, but again this may shrink the possibilities to compare to other results that do not satisfy the old criterion.

Remember that we understand distribution as the separation of a process into its (sequential) components. In order to formalise the identification of sequential components, we assume for each process calculus a so-called *labelling* on the capabilities of processes. The labelling has to ensure that (1) each capability has a label (2) no label occurs more than once in a labelled term, (3) that a label disappears only when the corresponding capability is reduced in a reduction step, and (4) that, once it has disappeared, it will not appear in the execution any more. The last three conditions are called unicity, disappearence, and persistence in [CCP09] which defines a labelling method to establish such a labelling for processes of the pi-calculus. Note that such a labelling can be derived from the syntax tree of processes possibly augmented with some informations about the history of the process, as it is done in [CCP09]. However, we assume that, once the labelling of a term is fixed, the labels are preserved by the rules of structural congruence as well as by the reduction semantics of the respective calculus. Because of recurrent operators, new subterms with fresh labels for their capabilities may arise from applications of structural congruence or reduction rules. Since we need the labels only to distinguish syntactically similar components of a term, and to track them alongside reductions, we do not restrict the domain of the labels nor the method used to obtain them as long as the resulting labelling satisfies the above properties for all terms and all their derivatives in the respective calculus. Due to space constraints, and in order not to clutter the development with the details of labelling, we prefer to argue at the corresponding informal level.

Example 3.4.1. Consider the term $P = \overline{y} \mid y.\checkmark \mid \overline{y} \mid y.0$ in the pi-calculus. It contains two syntactically equivalent outputs $\overline{y}$. In order to unambiguously distinguish its subterms as well as its steps we introduce labels. A suitable labelling of this example is e.g. $P = [\overline{y}]_1 \mid [y]_2.\checkmark \mid [\overline{y}]_3 \mid [y]_4.0$. As described in the following, P is distributable into at most four terms, namely $[\overline{y}]_1$, $[y]_2.\checkmark$, $[\overline{y}]_3$, and $[y]_4.0$, but is e.g. also distributable into $[\overline{y}]_3 \mid [y]_2.\checkmark$ and $[\overline{y}]_1 \mid [y]_4.0$. But it is e.g. not allowed to distribute P into $[\overline{y}]_1 \mid [y]_2.\checkmark$ and $[\overline{y}]_1 \mid [y]_4.0$, because these two components share the same capability, i.e., both contain a capability with the same label. Moreover, P can perform four different steps—reducing the labels 1 and 2, 1 and 4, 3 and 2, and 3 and 4, respectively—but modulo structural congruence we can only distinguish between two of them.

Before we can formalise what it means to preserve distributability, we have to formalise distributability itself. The most important operator to implement distribution or distributability is the parallel operator. More precisely, we consider distributability

as a special case of parallel composition with a stricter notion of independence, which becomes visible only in calculi as the join-calculus or by comparing calculi. So, first of all, two subterms are distributable if they are parallel.

Unfortunately, the converse of that statement—two subterms are not distributable if they are not parallel—is usually not true. The main reason for this is scoping of names. Consider for example the term $(\nu x)\,(P \mid Q)$ in the pi-calculus. Although the outermost operator is not the parallel operator, the processes P and Q are nonetheless distributable. More precisely, for all variants of the pi-calculus introduced in Section 2.1.1, two subterms are distributable if they are (modulo structural congruence) composed in parallel under some restrictions; see the notion of *standard form* of the pi-calculus [Mil99]. Hence, (1) we consider distributability modulo structural congruence, and (2) we allow to remove top-level restrictions and parallel operators to separate the distributable components.

Note that recurrent operators as the replicated inputs $y^\star(x)\,.P$ in the pi-calculus represent arbitrarily many copies of its subterm(s) in parallel as visualised by the reduction rule Pi-Rep$_{\mathrm{m,s}}$ for π_{m} that generates and reduces a copy of the subterm of the recurrent operator but does not reduce the replicated input itself. Accordingly, we consider replicated inputs as distributable. We can also visualise this by the (non-standard) structural congruence rules $y^\star(x)\,.P \equiv y(x)\,.P \mid y^\star(x)\,.P$ or $y^\star(x)\,.P \equiv y^\star(x)\,.P \mid y^\star(x)\,.P$. Indeed, if we add the first of these rules to the considered structural congruence of the pi-calculus, the respective rules for replicated inputs in the reduction semantics like Pi-Rep$_{\mathrm{m,s}}$ for π_{m} become superfluous. We do not add this rule to structural congruence in Figure 2.1, because this simplifies some of the considerations in Chapter 6. But, in order to simplify the following definition, we allow for such a structural congruence rule in the definition of distributability. More precisely, we define $\equiv^\star$ as the union of the rules in Figure 2.1 and $y^\star(x)\,.P \equiv y^\star(x)\,.P \mid y^\star(x)\,.P$.

In the case of the join-calculus, the situation is worse. Again, the problematic operator is scoping of names. But, in the case of the join-calculus, scoping is realised by definitions that at the same time represent the input capabilities of the calculus. Consider the term $R = \mathsf{def}\ a \triangleright 0\ \mathsf{in}\ (\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ (a \mid b))$. It is constructed of two nested definitions. Intuitively, it represents the combination of the two processes $\mathsf{def}\ a \triangleright 0\ \mathsf{in}\ a$ and $\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ b$ but, because of $c\,\langle a\rangle$, we can not get rid of the nesting of the definitions—not even modulo structural congruence. The best we can achieve is $R \equiv \mathsf{def}\ a \triangleright 0\ \mathsf{in}\ ((\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ b) \mid a)$. Note that $\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ b$ is not guarded within R. Because of that, the cooling and heating rules, which model structural congruence of the join-calculus, allow us to derive $\vdash R \rightleftharpoons b \triangleright c\,\langle a\rangle \vdash \mathsf{def}\ a \triangleright 0\ \mathsf{in}\ a \mid b$ as well as $\vdash R \rightleftharpoons a \triangleright 0 \vdash \mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ b \mid a$. This reason is enough for us to consider $\mathsf{def}\ a \triangleright 0\ \mathsf{in}\ a$ and $\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ b$ as distributable within R. Formally, each J-term J is distributable into the terms $J_1, \ldots, J_n \in \mathcal{P}_{\mathrm{J}}$ if, for all $1 \leq i \leq n$, there exists some multisets $\mathcal{R}, \mathcal{M}$ such that $\vdash J \rightleftharpoons \mathcal{R} \vdash J_i, \mathcal{M}$ and there are no two capabilities in $J_1, \ldots, J_n$ with the same label, i.e., the $J_1, \ldots, J_n$ represent distinct parts of the original term. Note that we can define structural congruence for all process calculi by a chemical abstract machine, but that this kind of special consideration is only necessary because definitions in the join-calculus are guards that have unguarded subterms. Hence, we assume that, (at least) for all process calculi that contain a guard with unguarded subterms, structural congruence is given by a chemical

abstract machine.

Note that this example on the join-calculus illuminates that we consider distributability as an irreversible predicate. There is no possibility to restore from a given set of distributable components the original process term, because by the separation of the components we irreversibly lose their original connections. Thus, we can not beyond doubt conclude that the terms $\mathsf{def}\, a \triangleright \mathsf{0}\, \mathsf{in}\, a$ and $\mathsf{def}\, b \triangleright c\langle a\rangle\ \mathsf{in}\, b$ originally belonged to R. Similarly, we can not conclude that the terms P and Q were originally subterms of the pi-calculus term $(\nu x)\,(P \mid Q)$, because we lost the information about the restriction. However, our main goal is the formulation of the preservation of distributability and, as it turns out, for this criterion we do not need the information lost, because the preservation of the behaviour of the original term and, thus, the preservations of the connections between distributable components, are already checked by the other quality criteria.

Another important observation on the join-calculus is that, here, replication is expressed by definitions; but it should not be allowed to distribute a single definition as we do for replicated inputs in the pi-calculus. This reflects a fundamental design decision in the join-calculus, namely that the receptors of a given channel are forced to reside at the same location [FG96, Lév97]. Note that this design decision marks the main difference between the join-calculus and the asynchronous pi-calculus. Accordingly, we require that this design decision is made explicit by the existence or absence of a rule of structural congruence as $y^\star(x)\,.P \equiv^\star y^\star(x)\,.P \mid y^\star(x)\,.P$ in the pi-calculus. A recurrent operator is called distributable if such a structural congruence rule is provided; otherwise, it is not distributable, i.e., J-term definitions are not distributable.

Definition 3.4.2 (Distributability)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus, $\equiv$ be its structural congruence, and $P \in \mathcal{P}$. P is *distributable* into $P_1, \ldots, P_n \in \mathcal{P}$ if there exists $P' \in \mathcal{P}$ with $P' \equiv P$ such that

1. for all $1 \leq i \leq n$, P_i is an unguarded subterm of P' or, in case $\equiv$ is given by a chemical approach, $\vdash P' \rightleftharpoons \mathcal{R} \vdash P_i, \mathcal{M}$ for some multisets $\mathcal{R}, \mathcal{M}$, such that P_i contains at least one capability or constant different from $\mathsf{0}$,

2. in $P_1, \ldots, P_n$ there are no two occurrences of the same capability, i.e., no label occurs twice, and

3. each guarded subterm and each constant of P' is a subterm of at least one of the terms $P_1, \ldots, P_n$.

The *degree of distributability* of P is the maximal number of distributable subterms of P.

Remember that for the considered variants of the pi-calculus we replace $\equiv$ by $\equiv^\star$.

By Definition 3.4.2, we can split a process into its sequential components or larger subterms, e.g. each term is distributable into itself. This allows us to analyse the behaviour of distributable subterms. Note that we do not allow to distribute the empty process, because otherwise usually every process is distributable into infinitely many empty processes. The same holds for subterms not containing any capability or constant different

from $\mathsf{0}$, as e.g. in the term $\mathsf{0} \mid \mathsf{0}$. As a consequence, e.g. the term $\mathsf{0}$ is not distributable at all. On the other side, the above definition leads to an infinite degree of distributability for each term containing a top-level distributable recurrent operator. Note that both of these design decisions—$\mathsf{0}$ is not distributable and the degree of distributability of $y^\star(x)\,.P$ is infinite—do not influence the results of this thesis.

Now, we formalise our new criterion. As discussed above, we require that encoded source terms are at least as distributable as the source term itself and that the encodings of distributable source term parts and their corresponding parts in the encoding are related by $\asymp$. By Definition 3.3.4, operational correspondence relates the behaviour of source terms and their encodings modulo some success respecting reduction bisimulation $\asymp$. Since our formulation of the property of preserving distributability is not independent of operational correspondence, it may have some odd meaning if the considered encoding does not satisfy operational correspondence. However, all results derived on the new criterion in the next chapter are based on the general framework which includes operational correspondence.

Definition 3.4.3 (Preservation of Distributability)**.** An encoding $[\![\cdot]\!] : \mathcal{P}_\mathrm{S} \to \mathcal{P}_\mathrm{T}$ *preserves distributability* if for every $S \in \mathcal{P}_\mathrm{S}$ and for all terms $S_1, \ldots, S_n \in \mathcal{P}_\mathrm{S}$ that are distributable within S there are some $T_1, \ldots, T_n \in \mathcal{P}_\mathrm{T}$ that are distributable within $[\![S]\!]$ such that $T_i \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$.

In essence, this requirement is a distributability-enhanced adaptation of operational completeness. Whenever a source term is distributable into n terms then its encoding must again be distributable into n terms, i.e., the encoded source term is at least as distributable as the source term itself. Moreover, if some of these n terms, say S_i, can perform some execution independent of the rest then, by operational completeness, this execution has to be emulated by its translation modulo $\asymp$, i.e., $S_i \Longmapsto_\mathrm{S} S_i'$ implies $T_i \Longmapsto_\mathrm{T} \asymp [\![S_i']\!]$. Thus, our formalisation of the preservation of distributability respects both the intuition on distribution as separation on different locations—captured by the structural requirement that the encoded source term is at least as distributable as the source term itself—as well as the intuition on distribution as independence of processes and their executions—a semantic requirement implemented by the condition $T_i \asymp [\![S_i]\!]$.

Of course this is not the only way to reason about distribution or distributability. However, we find it natural and appealing to require that parallel source term steps can be emulated truly in parallel, i.e., that for each pair of independent source term steps there is at least the possibility to emulate them independently.

3.4.4. Verification

As last step, we have to verify the new criterion against the current setting of existing criteria. First, we have to ensure that the new criterion is not in conflict to the existing criteria, i.e., positive results are still possible. Second, we have to argument why the new criterion results indeed in new results and is not already subsumed by some other criteria. At least the connection to the criteria identified in the second step in Section 3.4.2 as close to the new criterion should be analysed.

To ensure that the new criterion is not in conflict with the framework of Gorla, it suffices to show the existence of encodings that satisfy all six criteria. In Chapter 4 we discuss two such encodings; the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ from π_{s} into π_{a} of [Nes00] and the intermediate encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ from π_{m} (without replicated input) into π_{p}. In Chapter 6 we show that the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ satisfy the criteria of Gorla and preserve distributability. Moreover, every good encoding between different variants of the pi-calculus that translates the parallel operator and restriction homomorphically and preserves structural congruence also preserves distributability.

Lemma 3.4.4. *Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two two variants of the pi-calculus as introduced in Section 2.1.1, and let $\equiv_{\mathrm{S}}$ and $\equiv_{\mathrm{T}}$ be the structural congruence of $\mathcal{L}_{\mathrm{S}}$ and $\mathcal{L}_{\mathrm{T}}$, respectively. Any operationally complete encoding that translates the parallel operator and restriction homomorphically and preserves structural congruence, i.e., $S_1 \equiv_{\mathrm{S}} S_2$ implies $[\![S_1]\!] \equiv_{\mathrm{T}} [\![S_2]\!]$ for all $S_1, S_2 \in \mathcal{P}_{\mathrm{S}}$, preserves distributability.*

Proof. Let us assume that $\equiv_{\mathrm{S}}$ and $\equiv_{\mathrm{T}}$ contain the structural congruence rule $y^\star(x).P \equiv y^\star(x).P \mid y^\star(x).P$ (or $y^\star(\tilde{x}).P \equiv y^\star(\tilde{x}).P \mid y^\star(\tilde{x}).P$ if the considered calculus is polyadic). Then, by Definition 3.4.2, $S \in \mathcal{P}_{\mathrm{S}}$ is distributable into the terms $S_1, \ldots, S_n \in \mathcal{P}_{\mathrm{S}}$ if $S \equiv_{\mathrm{S}} (\nu x_1, \ldots, x_m)(S_1 \mid \ldots \mid S_n)$ for some sequence of names $x_1, \ldots, x_m \in \mathcal{N}$. $S \equiv_{\mathrm{S}} (\nu x_1, \ldots, x_m)(S_1 \mid \ldots \mid S_n)$ implies $[\![S]\!] \equiv_{\mathrm{T}} [\![(\nu x_1, \ldots, x_m)(S_1 \mid \ldots \mid S_n)]\!]$, because structural congruence is preserved by the encoding. By the homomorph translation of the parallel operator and restriction, we have

$$\begin{aligned} [\![S]\!] &\equiv_{\mathrm{T}} [\![(\nu x_1, \ldots, x_m)(S_1 \mid \ldots \mid S_n)]\!] \\ &= \left(\nu \varphi_{[\![]\!]}(x_1), \ldots, \varphi_{[\![]\!]}(x_m)\right) [\![S_1 \mid \ldots \mid S_n]\!] \\ &= \left(\nu \varphi_{[\![]\!]}(x_1), \ldots, \varphi_{[\![]\!]}(x_m)\right) ([\![S_1]\!] \mid \ldots \mid [\![S_n]\!]), \end{aligned}$$

where $\varphi_{[\![]\!]}$ is the renaming policy of $[\![\cdot]\!]$. Hence, $[\![S]\!]$ is distributable into the terms $T_1, \ldots, T_n \in \mathcal{P}_{\mathrm{T}}$ with $T_i = [\![S_i]\!]$ for all $1 \leq i \leq n$. Obviously, $[\![S_i]\!] \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$. We conclude that $[\![\cdot]\!]$ preserves distributability. □

Not surprisingly, the most crucial requirement here is the homomorphic translation of the parallel operator. However, this holds only in case of process calculi as the pi-calculus, where distributable terms can be separated modulo structural congruence by parallel operators. In calculi as the join-calculus the criterion on the homomorphic translation of the parallel operator is even stricter.

Thus, the (semantic) criterion formalised in Definition 3.4.3 can be considered to be at most as hard as the (syntactic) criterion on the homomorphic translation of the parallel operator. To see that it is not an equivalent requirement, but indeed strictly weaker, we consider the intermediate encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ from π_{m} (without replicated input) into π_{p}. This encoding is good and preserves distributability but it does not translate the parallel operator homomorphically. Moreover, [CM03] proves that there is no good encoding from π_{m} into π_{p} that translates the parallel operator homomorphically; this separation result does not rely on replication, i.e., it also implies that there is no such encoding from π_{m} without replicated input into π_{p}.

3.4.5. Alternative Formalisation

As discussed above, the criterion in Definition 3.4.3 requires not only the preservation of the distributability of processes but also the preservation of the distributability of steps or executions of the respective distributable processes. In order to obtain an alternative way to prove the preservation of distributability, we make this intuition explicit. More precisely, we show that an operationally complete encoding that preserves distributability always also preserves the distributability between sequences of source term steps. To do so, we define first what it means for two steps or executions to be distributable.

If a single process—of an arbitrary process calculus—can perform two different reduction steps, i.e., steps on capabilities with different labels, then we call these steps alternative to each other. Two alternative steps can either be in conflict or not; in the latter case, it is possible to perform both of them in parallel, according to some assumed step semantics.

Definition 3.4.5 (Distributable Steps)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathcal{P}$ a process. Two alternative steps of P are in *conflict*, if performing one step disables the other step, i.e., two steps are in conflict if both reduce the same not recurrent capability. Otherwise, i.e., if all capabilities used by both steps are recurrent, the steps are called *parallel*.

Two parallel steps of P are called *distributable*, if each recurrent capability reduced by both steps is distributable. Else the steps are called *local*.

Note that the "same" in the definition above means "with the same label", i.e., in $\overline{y} \mid y.P_1 \mid \overline{y}$ the two alternative steps on y are in conflict but $\overline{y} \mid y.P_1 \mid y.P_2 \mid \overline{y}$ and $\overline{y} \mid y^\star.P_1 \mid \overline{y}$ can both perform two parallel steps on y. Moreover, the reductions on channel a and b are parallel in $\overline{a} \mid \overline{b} \mid a.P_1 \mid b.P_2$, but there are in conflict in $\overline{a} \mid \overline{b} \mid a.P_1 + b.P_2$, because choice counts as a single capability which is reduced in both steps.

Also note that in contrast to parallel steps, distributable steps can reduce the same recurrent capability only if it is distributable. In many process calculi such as π_{a}, two steps are distributable iff they are parallel, because all recurrent capabilities are also distributable. Therefore, there is often no need to distinguish these two notions. However, there are also process calculi as the join-calculus in which these notions indeed refer to quite different situations. Thus, for the comparison with these calculi as in Section 4.3.2, their intuitive distinction is useful.

In the join-calculus, two alternative steps that reduce the same definition but do not compete for some output, as e.g. the reduction of $x\,\langle u\rangle$ and $x\,\langle v\rangle$ in $\mathsf{def}\, x\,(z) \triangleright y\,\langle z\rangle$ in $(x\,\langle u\rangle \mid x\,\langle v\rangle)$, can be considered as *parallel* steps; they do not compete for the input capability, because it is recurrent. However, we can *not* consider these two steps as distributable, as this would imply that the definition itself is distributable which—by design—is not intended in J: there is always exactly one receiver for each defined name [FG96, Fou98].

With the notions of conflicting, parallel, and distributable steps in mind, we define parallel and distributable sequences of steps.

Definition 3.4.6 (Distributable Executions)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus, $P \in \mathcal{P}$, and let A and B denote two executions of P. A and B are in *conflict*, if a step of A and a step of B are in conflict. Else A and B are *parallel*.

Two parallel sequences of steps A and B are *distributable*, if each pair of a step of A and a step of B is distributable.

In π_{a}, two sequences of steps A and B of a process P are parallel iff there exists some sequence of names $\tilde{x}$ and two terms $P_1, P_2 \in \mathcal{P}_{\mathrm{a}}$ such that $P \equiv (\nu\tilde{x})\,(P_1 \mid P_2)$ and P_1 can perform A while P_2 can perform B, i.e., if $A : P \longmapsto P_{A,1} \longmapsto \ldots \longmapsto P_{A,n}$ and $B : P \longmapsto P_{B,1} \longmapsto \ldots \longmapsto P_{B,m}$ then, for all $1 \leq i \leq n$ and all $1 \leq j \leq m$, there exists $P'_{A,i}, P'_{B,j} \in \mathcal{P}$ such that $P_{A,i} \equiv (\nu\tilde{x})\left(P'_{A,i} \mid P_2\right)$ and $P_{B,j} \equiv (\nu\tilde{x})\left(P_1 \mid P'_{B,j}\right)$. Moreover, two sequences of steps are again distributable iff they are parallel. Unfortunately, in the join-calculus two processes able two perform parallel sequences of steps can not always be separated by a parallel operator in this way; even if they do not reduce the same definition. The reason is again the restriction caused by definitions. In the term $\mathsf{def}\ a \triangleright P_1\ \mathsf{in}\ (\mathsf{def}\ b \triangleright c\,\langle a\rangle\ \mathsf{in}\ (a \mid b))$ the reduction of a is completely independent of the reduction of b. Hence, we consider these two steps as parallel and even as distributable. But, because of $c\,\langle a\rangle$, we can not get rid of the nesting of these two definitions.

Although the definitions of distributable processes in Definition 3.4.2 and distributable executions in Definition 3.4.6 are quite different, they are closely related. Two executions of a term P are distributable iff P is distributable into two subterms such that each performs one of these executions.

Lemma 3.4.7. *Let $\mathcal{L} = \langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus, $P \in \mathcal{P}$, and $A_1, \ldots, A_n$ a set of executions of P. The executions $A_1, \ldots, A_n$ are pairwise distributable within P iff P is distributable into $P_1, \ldots, P_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of P_i, i.e., during A_i only capabilities of P_i are reduced.*

Proof. Let $\equiv$ be the structural congruence of $\mathcal{L}$.

Assume that the set of executions $A_1, \ldots, A_n$ are pairwise distributable in P. By Definition 3.4.6, no pair of executions A_i and A_j with $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$ reduces the same not distributable capability. Moreover, since for all $1 \leq i \leq n$ the sequence of steps A_i is an execution of P, i.e., $P \longmapsto P_{i,1} \longmapsto \ldots \longmapsto P_{i,m}$ for some $P_{i,1}, \ldots, P_{i,m} \in \mathcal{P}$, none of these executions reduces a capability produced, i.e., unguarded, by a step of one of the other executions in the set $\{\, A_1, \ldots, A_n \,\}$. Thus, whenever an execution A_i reduces some capability that was guarded in P, then A_i also reduces the guarding capability. Hence, we can choose $P_1, \ldots, P_n$ such that, for all $1 \leq i \leq n$, P_i is an unguarded subterm of P' or can be separated in P' by the chemical approach with $P' \equiv P$ and P_i contains at least all capabilities reduced in A_i. Note that to ensure that all guarded subterms and constants of P are contained in at least one of the terms $P_1, \ldots, P_n$, as it is required by the last condition of Definition 3.4.2, some of these terms may contain subterms that are not reduced by one of the executions

$A_1, \ldots, A_n$. Since different executions A_i and A_j with $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$ reduce the same capability only if it is recurrent and distributable, by Definition 3.4.2, the terms $P_1, \ldots, P_n$ are distributable in P.

Now, assume that P is distributable into n terms $P_1, \ldots, P_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of P_i, i.e., during A_i only capabilities of P_i are reduced. Then, by Definition 3.4.2, no capability with the same label occurs twice in $P_1, \ldots, P_n$. Hence, since A_i reduces only capabilities in P_i, no two executions in $A_1, \ldots, A_n$ reduces the same capability. Thus, by Definition 3.4.6, all executions in $\{ A_1, \ldots, A_n \}$ are pairwise distributable in P. □

Based on the notion of distributable sequences of steps, we prove that an operationally complete encoding is distributability-preserving only if it preserves the distributability of sequences of source term steps.

Lemma 3.4.8 (Distributability-Preservation)**.** *An operationally complete encoding* $[\![\cdot]\!] : \mathcal{P}_S \to \mathcal{P}_T$ *that preserves distributability also preserves distributability of executions, i.e., for all source terms* $S \in \mathcal{P}_S$ *and all sets of pairwise distributable executions of* S*, there exists an emulation of each execution in this set such that all these emulations are pairwise distributable in* $[\![S]\!]$.

Proof. Let $\mathcal{L}_S = \langle \mathcal{P}_S, \longmapsto_S \rangle$ and let $\mathcal{L}_T = \langle \mathcal{P}_T, \longmapsto_T \rangle$ be two process calculi.

Let us assume that the set of executions $A_1, \ldots, A_n$ is pairwise distributable within S. Then, by Lemma 3.4.7, S is distributable into n terms $S_1, \ldots, S_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of S_i, i.e., during A_i only capabilities of S_i are reduced. Because $[\![\cdot]\!]$ preserves distributability, by Definition 3.4.3, there are some $T_1, \ldots, T_n \in \mathcal{P}_T$ that are distributable within $[\![S]\!]$ such that $T_i \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$. Let us fix some arbitrary $i \in \{ 1, \ldots, n \}$. By operational completeness in Definition 3.3.4, all sequences of steps of S_i are emulated by its encoding, i.e., $S_i \Longmapsto_S S_i'$ implies $[\![S_i]\!] \Longmapsto_S \asymp [\![S_i']\!]$. Because $\asymp$ is some reduction bisimulation, $T_i \asymp [\![S_i]\!]$ implies that also T_i has to emulate the executions of S_i independently from the other encoded subterms, i.e., $[\![S_i]\!] \Longmapsto_S \asymp [\![S_i']\!]$ implies $T_i \Longmapsto_S \asymp [\![S_i']\!]$. We conclude that for all $1 \leq i \leq n$ the term T_i emulates the sequence of steps A_i. Then, again by Lemma 3.4.7, all these emulations are pairwise distributable within $[\![S]\!]$. □

3.5. Summary and Related Work

As the title suggests, the main purpose of this chapter is to introduce encoding functions and to discuss their quality. In Section 3.1 we presented a short formalisation of encoding functions as well as target terms as they are used throughout this thesis.

In Section 3.2 we then reviewed some of the most common used quality criteria. In [Nes96] also a class of rather quantitative criteria for the effectiveness or efficiency of an encoding is discussed. The efficiency of an encoding can be measured for example by a criterion to count the number of messages or steps of an emulation [BS83, AH92, Kna93]. However, within this thesis we are more focused on semantic and structural criteria.

In Section 3.3 we reviewed one general framework as core for language comparison that will guide most of the results of the thesis. Suitability of quality criteria for encodability as well as separation results were often discussed in literature (see e.g. [Nes96, Nes06, VPP07, Par08, Gor10b, FL10]), but general frameworks or discussion of such general frameworks are rare. For a long time full abstraction was accepted as general criterion for language comparison. But, since there is no agreement on the equivalences that should be used, full abstraction results are seldom comparable without some additional effort to unify the notions of the used equivalences. Hence, to our opinion, full abstraction can hardly be considered as a general framework. Apart from that many authors state that a reasonable encoding should at least satisfy this or that property. Prominent candidates—not counting candidates as the homomorphic translation of the parallel operator in [Pal03] that are obviously added in order to measure some domain-specific properties—are operational correspondence and divergence reflection (see e.g. [Nes96, Nes00]). A variant of both is contained in the framework of [Gor08b, Gor10b] presented in Section 3.3. Another attempt for such a general frameworks for language comparison can e.g. be found in [FL10]. In contrast to [Gor10b], the authors of [FL10] present a general framework based on labelled semantics. Intuitively, they combine full abstraction and operational correspondence into a bisimulation-like relation called *subbisimulation*. Subbisimulation connects labelled executions of the source and the target language. Moreover, preservation and reflection of divergence is required. The approach is then used to compare different variants of CCS and different variants of the pi-calculus. For example subbisimilarity is applied to show the independence of the operators of the pi-calculus. In comparison to the framework of Gorla the requirements induced by the formulation of subbisimilarity seem to be stricter, although a direct comparison is difficult, because of the different formulations. However, note that within this framework the positive result presented in Chapter 5, i.e., the encoding from π_{m} into π_{a}, does not hold, because [FL10] make use of a stricter variant of operational correspondence that does not allow for intermediate or partially committed states.

In Section 3.4 we showed how the general framework can be extended by an additional domain-specific criterion. To our opinion the generation of a new criterion has to follow basically the following four steps.

1. Fix the purpose of the new criterion and the setting(s) in which it should be used.

2. Compare and classify the intended criterion with respect to existing criteria. Convince yourself that the intended criterion is not already captured by a well-known criterion from the literature.

3. Formalise the criterion (possibly with respect to existing criteria of the chosen setting but) as general as possible. Discuss why the new criterion indeed captures the intended meaning.

4. Verify the new criterion against the setting of quality criteria. Ensure that the extension of the setting still allows for positive results and that new negative results are possible.

Furthermore, we presented an alternative formulation for the new criterion, i.e., already derive some results on the new criterion that do not depend on the considered pair of languages. Note that we do the same for the general framework in Lemma 3.3.9 and Lemma 3.3.10. Such results on the criteria are often very useful, because they (1) provide additional intuition on the criteria and (2) possibly allow for new proof methods to obtain translational results. Indeed, we use Lemma 3.3.9 and Lemma 3.3.10 as well as the results derived in Section 3.4.5 to obtain translational separation results in the second half of the next chapter.

As main contribution of this chapter, Section 3.4 proposes a novel criterion to reason about the degree of distributability which is better suited than the common homomorphic translation of the parallel operator. We show that this criterion allows us to formalise the difference between the asynchronous pi-calculus and the join-calculus (Section 4.3) as well as to shed more light on the difference between the expressive power of mixed and separate choice (Section 4.4).

4. Separating Languages

The main purpose of this chapter is to analyse how translational separation results are obtained in general and in particular to show separation between some synchronous and asynchronous variants of the pi-calculus. As already shown in [Pal03] and [Nes00], the difference in the expressive power of the synchronous pi-calculus (π_{m}) and the asynchronous pi-calculus (π_{a}) is a consequence of the different expressive power of mixed choice compared to separate choice.[1] We want to shed further light upon this difference. To do so we consider different translational separation results between π_{m} and π_{s} (a synchronous variant of the pi-calculus with separate choice) in particular with respect to distributability. Finally, we capture the difference within a so-called *synchronisation pattern* that describes this difference by means of distributable and conflicting steps. Moreover, we analyse the distributability of the asynchronous pi-calculus by comparing its expressive power with the join-calculus. Again we capture the difference between these two calculi within a synchronisation pattern.

Translational separation results and separation results in general show in what way two languages differ. Hence, separation results are specific to the pair of considered languages and one may assume that separation results have nothing in common. But in fact, if we compare different separation results, we observe that most of them share some kind of meta proof method (as e.g. in [Bou88, HP01, Pal03, CM03, VPP07, Gor08a, Gor09, Gor10b, FL10, LV10]). They prove first an absolute result, i.e., some condition of the source or target language, which then guides the proof of the translational separation result. This technique is particularly useful if not only a single separation result but several of them are proved with respect to the same or comparable instances of the same absolute result (see e.g. [VPP07, Gor10b]). For the common thread of this chapter we concentrate on this observation.

The absolute expressive power of a language describes what kind of behaviour or operations on behaviour are expressible in it (see [Par08, Gor10b] and even [LSZ74]). Analysing the absolute expressive power of a language usually consists of analysing which "problems" can be solved in it and which cannot. It is often difficult to identify a suitable problem instance or problem domain to properly measure the expressive power of a language. For instance, one might consider Turing-completeness to measure the computational power of a language. In fact, Turing-completeness has been used in the context of process algebras, e.g. for Linda [BGZ00]. However, many calculi are Turing-complete and can thus not be distinguished by this problem. Hence, in Section 4.1 we discuss how to obtain suitable problem instances. As examples we review two absolute results that are introduced by Palamidessi in [Pal03] in order to separate π_{m} and π_{s}. Palamidessi

[1] See e.g. [Bou92, QW00, Gor07, Gor08a] for a comparison of the choice free fragment of the synchronous pi-calculus and the asynchronous pi-calculus.

uses confluence and, inspired by Bougé [Bou88], the distributed coordination problem of leader election. Leader election refers to initially symmetric networks, where all potential leaders have equal chances and all processes run the same—read: symmetric—code. There, to solve the leader election problem, it is required that in all possible executions a leader is elected. Usually, it is argued that it is necessary—again in all possible executions—to break the initial symmetry in order to do so. On the other hand, if there is just a single execution in which the symmetry is somehow perpetually maintained or at least restored, then also leader election may fail, and thus the leader election problem is not solved. One may conclude that, at a closer look, Palamidessi's proof furthermore addresses another problem: the problem of breaking initial symmetries. Therefore, we suggest to promote "breaking symmetries" from a mere auxiliary proof technique to a proper problem of its own. It turns out that, by doing so, we can significantly weaken the definition of symmetry and at the same time provide a stronger proof applicable to problem instances different from leader election. We conclude that in the considered setting breaking symmetries is better suited as leader election to derive a translational separation result. To underpin that, we discuss under which circumstances an absolute result is suited for the derivation of a separation result.

Later on, Gorla [Gor08b, Gor10b] uses a simpler difference between π_{m} and π_{s}. Instead of leader election in symmetric networks, it employes the reducibility of "incestual" processes (mixed choices that include both enabled senders and receivers for the same channel) when running two copies in parallel. Note that Gorla does not explicitly use a notion of symmetry. Both Palamidessi and Gorla rephrased their results by stating that there is no good encoding from π_{m} into π_{s}. In Section 4.2 we review these translational separation results. In each case, the ability to break initial symmetries turns out to be essential. Furthermore, we present two new negative translational results that separate π_{m} and π_{s} with respect to distributability and causality. While doing so, we concentrate on the consequences of varying notions of uniformity and reasonableness, i.e., of varying sets of quality criteria. Moreover, we show how absolute results can be used to derive translational separation results.

In Section 4.1 we discuss how to obtain a suitable absolute result from scratch. Other ways to obtain such a result are to transfer an absolute result from another formalism or to adapt an absolute result of another translational result in the context of process calculi. We consider examples for both ways in Section 4.3 and Section 4.4. Again, we compare synchronous and asynchronous variants of the pi-calculus as running examples, but, in contrast to the first half of this chapter, Section 4.3 and Section 4.4 are more focused on the domain of distributability.

Other results on distributability can for instance be found for Petri nets in [vGGS08, vGGS09, BD12, vGGS12]. In [vGGS08] a semi-structural property called M is derived that distinguishes distributable Petri nets from those nets that may be implemented only under additional assumptions on the underlying system structure in a fully asynchronous and distributed setting. They also present a description of this property as a property of a step transition system which allows us to use this property to reason about process calculi. In Section 4.3 we show how this property can be transferred into an absolute result to analyse distributability of the pi-calculus. Intuitively, the degree

of distributability in the pi-calculus corresponds to the amount of parallel components that can act independently. However, practical experience has shown that it is not possible to implement every pi-calculus term—not even every asynchronous one—in an asynchronous setting while preserving its degree of distributability, at least not with an automatic algorithm. To overcome these problems, the join-calculus was introduced as a model of distributed computation [Lév97]. It employs a *locality* principle by ensuring that there is always exactly one immobile receiver for each communication channel. More precisely, for every name, exactly one receiver is defined at the time of the name's creation, and communication occurs only on so-defined channels [Fou98]. Apart from that, the join-calculus can be considered as an asynchronous variant of the pi-calculus. By transferring the M of Petri nets into absolute results for the join-calculus and the asynchronous pi-calculus, we are able to prove a difference in the expressive power of these two calculi with respect to distributability, elucidated by the non-existence of a good and distributability-preserving encoding from π_{a} into J. Moreover, with exemplary results in the context of CSP, we show that the presented proof method, based on *synchronisation patterns* like M, can also be applied to obtain separation results in other process calculi. Thereby we also show how translational separation results can be transferred to other source and target languages.

In Section 4.4 we adapt the absolute results of Section 4.3 to capture the difference between mixed choice and separate choice. It turns out that the difference in the expressive power of these two choice variants with respect to distributability can be described by a synchronisation pattern that is similar to the pattern used in Section 4.3 and [vGGS08] but that is more complex. Moreover, we do not only adapt the synchronisation pattern and the corresponding absolute results but also the rest of the proof method used in Section 4.3.

4.1. Absolute Results

The first section of this chapter is dedicated to the derivation of absolute results suited to obtain translational results. The most difficult task in obtaining absolute results is usually the identification of a suitable problem description. We present two ideas that can serve as starting points in this direction. Moreover, we discuss what makes an absolute result suitable to derive a particular separation result.

Often two sublanguages of the same family of process calculi that differ only by single operators or simple syntactical differences are compared in order to gain some knowledge about the expressive power of the respective operator or syntactical restriction. In Section 4.1.1 we present an absolute result of [Pal03] that is the formalisation of such a simple syntactical difference between two languages. We also show how to derive a very simple separation result on top of that absolute result.

If the two languages we want to compare are not that close, or if the absolute result derived from syntactical differences is to weak, another idea is to use standard problems for an absolute result. Therefore, in Section 4.1.2 we review such an absolute result which again was already presented in [Pal03].

In Section 4.1.3 we show how to generalise this absolute result such that it fits better to the considered setting of quality criteria. Corresponding separation results are then derived in Section 4.2.

4.1.1. Formalising the Difference of Languages

Our goal is to analyse the expressive power of mixed choice, or precisely to compare the full pi-calculus including mixed choice (π_{m}) with its subcalculus, in which only separate choice is allowed (π_{s}). The only syntactical difference between these two calculi is the restriction on the choice operator. Now, we have to describe what effects on the expressive power this syntactical restriction implies, i.e., what kinds of behaviour are expressible in π_{m} but not in π_{s}. Fortunately, this task is quite easy, because the two considered languages differ only by the characteristics of a single operator. However, whenever the languages we want to compare are close or whenever there is already some knowledge about the connection between the operators of the two languages, it is a good advice to start with the formalisation of the differences between the languages with respect to that single operators or those small syntactical differences. Of course, we do not know how Palamidessi obtained the idea to describe this difference in form of confluence, but it seems to be a good idea to start close to the operator itself.

A mixed choice is the combination of an input and an output capability within a single sum. By contrast, in separate choice no such combinations are allowed. A sum describes a process that behaves as any of its branches. As a special feature, as soon as the process decides on which branch it proceeds, all the other branches of the sum are immediately withdrawn. We observe that the reduction of no other operator of the pi-calculus can immediately withdraw alternative actions or action guarded branches. Because of that, it is possible in π_{m} that a send operation immediately withdraws an alternative input guarded subprocess, and accordingly a receive operation immediately withdraws an alternative output guarded subprocess, although no communication was performed. But both behaviours are impossible in π_{s}.

Palamidessi denotes this inability to immediately disable alternative operations of the opposite kind as *confluence* property [Pal03]. Let $\overline{x}\,[y]$ denote the label of an arbitrary output action, i.e., $\overline{x}\,[y]$ refers either to a bound output $\overline{x}\,(y)$ or an unbound output $\overline{x}\,y$.

Lemma 4.1.1 (Local Confluence)**.** *Let $P \in \mathcal{P}_{\mathrm{s}}$ be a process. If P can perform two steps $P \xrightarrow{\overline{x}[y]} Q$ and $P \xrightarrow{z(w)} R$ then there exists S such that $Q \xrightarrow{z(w)} S$ and $R \xrightarrow{\overline{x}[y]} S$.*

Confluence is visualised in Figure 4.1. The proof is by analysis of the possible rules used to derive the labelled steps and by the fact that an input and an output guarded term cannot be combined within a sum in π_{s}. We recall the proof of [Pal03].

Proof. The following proof is an adaptation of the proof of Lemma 4.1 in [Pal03] at pages 17 to 18.

There are two steps $P \xrightarrow{\overline{x}[y]} Q$ and $P \xrightarrow{z(w)} R$, i.e., these two steps can be derived from P using the rules of the labelled semantics of π_{s} in Figure 2.2. None of the Rules Pi-ls-Tau-Sum, Pi-ls-Com, or Pi-ls-Close was used to derive one of

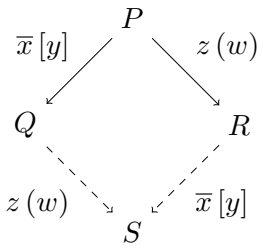


Figure 4.1.: Local Confluence [Pal03].

these steps, because they are not labelled by τ. Of the remaining rules Pi-ls-Open, Pi-ls-Par, Pi-ls-Res, and Pi-ls-Cong cannot be applied in the root of the proof trees for these two steps. Without Pi-ls-Com and Pi-ls-Close the two proof trees cannot have more than a single branch. Moreover, in the root of the proof tree for $P \xrightarrow{\overline{x}[y]} Q$ the Rule Pi-ls-O-Sum and in the root of the proof tree for $P \xrightarrow{z(w)} R$ either Pi-ls-I-Sum or Pi-ls-Rep is applied. To apply these rules, P have to contain an unguarded sum with one branch being some output guarded subterm $\overline{x}\langle y\rangle\,.P_1$ and either an unguarded sum with one branch being some input guarded subterm $z(w)\,.P_2$ or an unguarded subterm $z^\star(w)\,.P_2$ for some $P_1, P_2 \in \mathcal{P}_\mathrm{s}$. A replicated input cannot be a branch of a sum and, since mixed sums are forbidden in π_s, the input guarded term and the output guarded term cannot be combined within the same sum in P. Hence, both subterms, the sum with the output guarded branch and the sum with the input guarded branch or the replicated input, have to be combined in parallel in P. By Rule Pi-ls-Par, the parallel context of a term is left unchanged in a step. Hence, the steps $P \xrightarrow{\overline{x}[y]} Q$ and $P \xrightarrow{z(w)} R$ preserve the respective other subterm and with it the ability to perform the respective other action. By applying again the according rule Pi-ls-Par and all the other rules of the steps $P \xrightarrow{\overline{x}[y]} Q$ and $P \xrightarrow{z(w)} R$ respectively we obtain the steps $Q \xrightarrow{z(w)} S$ and $R \xrightarrow{\overline{x}[y]} S$. □

By the way, confluence is basically the opposite of conflicts. As we observe in Section 4.3 and Section 4.4, the possibility to express different kinds of conflicts and their combination with distributable steps distinguishes many different process calculi with different synchronisation mechanisms or different characteristics on the operators expressing capabilities of communication. Hence, it is always a good advice to carefully study confluence properties, in order to reason about the effect of such differences in process calculi.

Note that also π_a fulfils the confluence property, because, by Definition 2.1.7 and Definition 2.1.8, π_a is a subcalculus of π_s. To show that confluence distinguishes the absolute expressive power of π_m and π_s, consider the following example.

Example 4.1.2 (Breaking Confluence)**.** Consider the π_m-term

$$P \triangleq (\overline{x} + y)\,.$$

P can perform either an output on x or an input on y. To fulfil the confluence property, P must be able to perform an input on y after the output on x, and to perform an output on x after the input on y. But, since $P \xrightarrow{\overline{x}} \mathsf{0}$ and $P \xrightarrow{y} \mathsf{0}$, P cannot perform two subsequent steps at all.

By Definition 2.1.6 and 2.1.7, π_{s} is a subcalculus of π_{m}. So, π_{m} is at least as expressive as π_{s}. Now, by Example 4.1.2 and Lemma 4.1.1, π_{m} is strictly more expressive than π_{s} and π_{a}. Of course, it is not really surprising, that a restriction in the syntax of π_{m} leads to a less expressive subcalculus. But how big is the induced gap? To answer questions like these, translational results are well suited, because encodings, i.e., translations, abstract from the exact formalisation of behaviour in a specific calculus and allow to compare the abstract behaviour of languages, where the degree of abstraction is defined by the quality criteria on the encoding. For instance, a very naive—and obviously stupid—quality criterion of an encoding $[\![\cdot]\!]$ is to require that $[\![S]\!] = S$ holds for all source terms S. This requirement is stupid, as it allows only for the identity function to be a suitable encoding, i.e., requires syntactical equivalence of source and target.

A (little) better requirement is $[\![S]\!] \sim S$, where $\sim$ is strong bisimilarity defined similarly on both languages in Section 2.2. In this case, the abstract behaviour measured is their behaviour modulo strong bisimilarity, which is a strict but very common way to analyse the behaviour at least of two terms of a single pi-calculus variant. Based on this requirement, we can prove that there is no encoding $[\![\cdot]\!]$ from π_{m} into π_{s} such that $[\![S]\!] \sim S$ for all source terms $S \in \mathcal{P}_{\mathrm{m}}$.

Lemma 4.1.3 (Separation Result)**.** *There exists no encoding $[\![\cdot]\!]$ from π_{m} into π_{s} such that $[\![S]\!] \sim S$ for all source terms $S \in \mathcal{P}_{\mathrm{m}}$.*

Proof. Assume the contrary, i.e., assume there is an encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$ such that $[\![S]\!] \sim S$ for all source terms $S \in \mathcal{P}_{\mathrm{m}}$. Consider the term $S \triangleq (\overline{x} + y)$ of Example 4.1.2 as counterexample. The source term S can perform two steps: $S \xrightarrow{\overline{x}} \mathsf{0}$ and $S \xrightarrow{y} \mathsf{0}$, where the respective derivation $\mathsf{0}$ cannot perform any step at all. Since $[\![S]\!] \sim S$, by Definition 2.2.1, there exists two target terms $T_1, T_2 \in \mathcal{P}_{\mathrm{s}}$ such that $[\![S]\!] \xrightarrow{\overline{x}} T_1$, $[\![S]\!] \xrightarrow{y} T_2$, and $T_1 \sim \mathsf{0} \sim T_2$. Then, by confluence (Lemma 4.1.1), there exists some $T' \in \mathcal{P}_{\mathrm{s}}$ such that $T_1 \xrightarrow{y} T'$ and $T_1 \xrightarrow{\overline{x}} T'$. But, since $\mathsf{0} \not\longrightarrow$, this contradicts $T_1 \sim \mathsf{0} \sim T_2$. □

Observe that the counterexample used in the proof above is exactly the same example we use to show that π_{m} does not satisfy the confluence property. Hence, in this case, identifying a suitable counterexample is easy, mainly because our absolute separation result and the quality criterion are very close.

Again this requirement is too strict, because many interesting and accepted encodings between different calculi allow to translate single steps into sequences of steps. Hence, the source terms and their translations can usually be related at most by a weak equivalence. But the confluence property does not hold in case of weak steps as the following example illustrates.

Example 4.1.4. Consider the term $Q \triangleq (\nu l)\left(\overline{l} \mid l.\overline{x} \mid l.y\right)$ with $Q \in \mathcal{P}_{\mathrm{s}}$. Q consists of a simple restricted lock l, which guards an output on x and an input on y. Q can perform two different visible steps after removing the lock by an invisible step, i.e., $Q \stackrel{\overline{x}}{\Longrightarrow} \mathsf{0}$ and $Q \stackrel{y}{\Longrightarrow} \mathsf{0}$, but in both cases there are no further steps. Hence, Q can perform either a visible output or a visible input but not both.

Nonetheless, the Q from the example above and the P from Example 4.1.2 are not related by weak bisimilarity $\approx$. But alone with confluence, it takes some effort to prove that there is no encoding from π_{m} into π_{s} such that $[\![S]\!] \approx S$ for all source terms S. Moreover, even a direct comparison of source and target by weak bisimilarity is usually a very hard requirement. In general, and especially when the target language is not a sublanguage of the source, it is not possible to relate source and target terms by a standard equivalence. We discussed the problem of choosing the right quality criteria of an encoding in Chapter 3.

Palamidessi in [Pal03] uses the homomorphic translation of the parallel operator as one of her quality criteria, because it ensures the preservation of the degree of distribution of source terms as discussed in Section 3.4.2. As discussed later in this Chapter, this requirement can still be considered as too strict. However, for the moment let us restrict our attention to encodings that translate the parallel operator homomorphically. Unfortunately, to prove the non-existence of such an encoding is again difficult if we rely only on the confluence property. The problem is to identify a suitable counterexample based on confluence and to show that its main properties are preserved by the quality criteria. Palamidessi in [Pal03] circumvent this problem by proving another absolute result based on confluence. Inspired by the work of Bougé in [Bou88], she chooses the problem of solving leader election in symmetric networks; a famous problem often used in the context of distributed systems.

However, note that confluence is crucial for both of the following absolute results: leader election in symmetric networks in Section 4.1.2 as well as breaking symmetries in Section 4.1.3. It ensures that a communication of two processes of a network cannot immediately withdraw the possibility of all other network processes to mimic this communication, which is the basic argument for both of the other absolute results. So, why do we need an absolute result on top of confluence? The answer is, it is not necessary but it is extremely helpful to derive translational results. To use an absolute result in a translational result, we derive a term from the positive absolute result that we can use as counterexample in the translational result and ensure that the discriminating properties of this example are preserved by the required quality criteria, as it was done in the proof of Lemma 4.1.3. It is not easy to obtain such an example directly from confluence and, in case of the criteria used in Section 4.2, it is very hard and intricate to argue for the preservation of its relevant properties. Deriving more complex absolute results on top of confluence may need some effort, but it makes the derivation of translational results significantly easier.

4.1.2. Standard Problems

Another way to obtain an absolute result to distinguish two languages is to consider standard problems of the respective domain. In the area of distributed systems leader election or consensus problems in general are the most frequently used problem instances to reason about the expressive power of a language (see e.g. [LL77, Gar82, Bou88, Lyn96, VPP07]).

Leader election is an abstract formalisation of a consensus problem that often arises in practice. In general leader election consists of a network of n processes. To solve leader election, the n processes have to choose a leader among them. There are different variants of the leader election problem differing in the assumptions on the given network, existence and kind of unique identifiers for the processes of the network, or the way the processes proclaim their solution. For instance, [Lyn96] requires that the winning process changes some special status component of its state to declare itself as leader. Whereas in [Pal03] a special channel *out* is assumed to propagate the index of the winning process, i.e., the unique identifier of the leader.

We restrict our attention to leader election in symmetric networks, because else leader election can always be solved by implementing a central coordinator. There is no common definition of symmetric networks that is suitable for all kinds of domains and considered problems[2], but in the context of leader election the definition of symmetric networks usually includes at least two properties. (1) All processes of a symmetric network run basically the same abstract code. And (2) the topology of the network does not allow for trivial solutions, as to choose the root as leader in case the topology of the network is a tree. To ensure the last point, it is usually required that the links between the processes of the network are distributed symmetrically over the processes, as it holds in completely connected networks (each process can interact with each other process) or in networks whose processes form a circle.

In the following we review the absolute and the separation result given in [Pal03] as well as some previous work in the context of CSP in [Bou88]. Thus, missing proofs, formal definitions, and further explanations can be found there.

As we do in Section 2.1, [Pal03] defines a network as the parallel composition of processes with possibly some surrounding application of the restriction operator, i.e., a network is a term of the form $(\nu\tilde{x})\,(P_1 \mid \ldots \mid P_k)$ for a sequence of names $\tilde{x} \in \mathcal{S}(\mathcal{N})$ and some processes $P_1, \ldots, P_k$. However, a major difference between networks in [Pal03] and networks within this thesis is that [Pal03] explicitly allows the use of the process indices $1, \ldots, k$ as data in the processes of the network. They are used to propagate the identity of the winning process, i.e., the leader, over a special channel named *out*. More precisely, in [Pal03] the leader election problem is solved by a network iff in each of its maximal executions each process propagates the same process index over *out* and no other index is propagated.

Usually a network is considered to be symmetric if all its processes are identical modulo some renaming according to a permutation σ on their free names. Bougé denotes this kind of symmetry as *syntactic symmetry*, because the processes can be considered to run

[2] cf. Johnson and Schneider: "Symmetry means different things to different people." [JS85]

syntactically identical code up to the renaming (see [Bou88]). He also points out that the unique identifiers—the indices—of the processes can be augmented to do some cheating, i.e., they allow for solutions of leader elections in symmetric networks even though the network is purely asynchronous. The following example presents such a syntactically symmetric network solving leader election in π_{s}.

Example 4.1.5 (Asynchronous Symmetric Solution of Leader Election)**.** Consider the network

$$N \triangleq (\nu x, y)\,(P \mid \sigma(P))$$

with

$$P = \overline{x} \mid x.\overline{out}\langle 1\rangle + y.\overline{out}\langle 2\rangle \qquad \text{and} \qquad \sigma = \{\, {}^{x}\!/_{y}, {}^{y}\!/_{x} \,\}\,.$$

N is syntactically symmetric with respect to the permutation σ, i.e., $N = (\nu x, y)\,(P \mid P')$ for some P' equal to P modulo the exchange of x and y according to σ. Moreover, N solves the leader election problem, since in all maximal executions of N both subprocesses send via *out* the same index and no other index is transmitted.

To overcome these problems, i.e., to rule out such examples, Bougé recommends the use of *semantic symmetry*, i.e., syntactical symmetry augmented with a semantic component. The *semantic component* of the symmetry definition is designed to be strongly connected to the problem considered, i.e., leader election in this case. Intuitively, its purpose is to ensure that the only way to solve the leader election problem is to break the initial symmetry of the given network. Note that N does *not* break the initial syntactic symmetry to solve leader election, because e.g. in the execution

$$N \xrightarrow{\tau} P \mid \overline{out}\langle 1\rangle \xrightarrow{\tau} \overline{out}\langle 1\rangle \mid \overline{out}\langle 1\rangle \xrightarrow{\overline{out}\,1} \mathsf{0} \mid \overline{out}\langle 1\rangle \xrightarrow{\overline{out}\,1} \mathsf{0} \mid \mathsf{0} \not\longrightarrow$$

each second step results in a network that is syntactically symmetric with respect to σ. So, without an additional semantic component in the definition of symmetry, the leader election problem cannot be used to distinguish π_{m} and π_{s}.

We shortly revisit Palamidessi's notion of symmetry for the pi-calculus [Pal03]. Note that the involved definitions are based on the ones introduced by Bougé in [Bou88] for CSP.

According to [Pal03], a hypergraph is a tuple $H = \langle\, N, X, t \,\rangle$, where N and X are finite sets whose elements are called nodes and edges, and t, called type, is a function assigning to each edge the set of nodes connected by this edge. An automorphism on a hypergraph is a pair $\sigma = \langle\, \sigma_N, \sigma_X \,\rangle$ such that $\sigma_N : N \to N$ and $\sigma_X : X \to X$ are permutations which preserve the type of edges. Given a hypergraph H and σ on H the orbit of a name n is the set of nodes in which the iterations of σ map n.

A network $P \equiv (\nu\tilde{x})\,(P_1 \mid \ldots \mid P_k)$ of k processes solves the leader election problem if for every execution of P there exists an extension and an index $n \in \{\, 1, \ldots, k \,\}$ such that for each process the extended execution contains an output action of the form $\overline{out}\,n$ and no other action $\overline{out}\,m$ with $m \neq n$. The hypergraph associated to a network P is the

hypergraph $H(P) = \langle\, N, X, t \,\rangle$ with $N = \{\, 1, \ldots, k \,\}$, $X = \mathsf{fn}(P_1 \mid \ldots \mid P_k) \setminus \{\, out \,\}$, and for each $x \in X$, $t(x) = \{\, n \mid x \in \mathsf{fn}(P_n) \,\}$. Given a network P and the hypergraph $H(P)$ associated to P, an automorphism on P is any automorphism $\sigma = \langle\, \sigma_N, \sigma_X \,\rangle$ on $H(P)$ such that σ_X coincides with σ_N on $N \cap X$ and σ_X preserves the distinction between free and bound names.

A network P with the associated hypergraph $H(P) = \langle\, N, X, t \,\rangle$ and an automorphism σ on P is symmetric with respect to σ iff for each node $i \in N$, $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$[3].

The main point of the semantic component of symmetry is that the special channel *out* cannot be renamed by σ while the indices of the processes of the network must be permuted by σ. With that, the network N in Example 4.1.5 above is *not* symmetric according to [Pal03], because it does not permute the indices. This allows Palamidessi to prove that for each execution of a network in $\mathcal{P}_{\mathrm{s}}$, which is symmetric with respect to an automorphism σ, whenever there is an output $\overline{out}\, i$ there is an output $\overline{out}\, \sigma(i)$ with $\sigma(i) \neq i$ as well, which contradicts the leader election problem. This explains why in [Bou88, Pal03, VPP07] such an effort is spent to define symmetry.

To distinguish π_{m} and π_{s} Palamidessi shows that a network $P \in \mathcal{P}_{\mathrm{s}}$ which is symmetric with respect to an automorphism σ on P with only one orbit cannot solve the leader election problem while this is possible in π_{m}.

Theorem 4.1.6 ([Pal03] at page 18)**.** *Consider a network $P = (\nu \tilde{x})\,(P_1 \mid \ldots \mid P_k)$ in π_{s}, with $k > 2$, $\tilde{x} \in \mathcal{S}(\mathcal{N})$, and $P_1, \ldots, P_k \in \mathcal{P}_{\mathrm{s}}$. Assume that P has an automorphism σ with only one orbit, and that P is symmetric with respect to σ. Then P cannot solve the leader election problem.*

The existence of such a network in π_{m}, i.e., a network that is symmetric with respect to an automorphism with only one orbit, that solves the leader election problem is claimed at page 25 of [Pal03] and the construction of such networks is explained on the following pages. Based on this absolute difference between π_{m} and π_{s} Palamidessi proves the following well-known separation result.

Corollary 4.1.7 ([Pal03] at page 33)**.** *There exists no uniform encoding of π_{m} into π_{s} preserving a reasonable semantics.*

Here, uniform means that the encoding translates the parallel operator homomorphically, because that ensures that the degree of distribution of source terms is preserved by the encoding function, and that $[\![\, \sigma(P) \,]\!] = \theta([\![\, P \,]\!])$ such that $\forall i \in \mathbb{N} \,.\, \sigma(i) = \theta(i)$, i.e., the encoding preserves renamings and does not manipulate occurrences of indices of source term processes. With reasonable [Pal03] requires an encoding to be compositional on the remaining operators and the preservation of some intended semantics which is not further specified but includes at least that the encoding of a network solving leader election in the source should solve leader election in the target.

Note that this separation result was the first that pointed out the importance of the expressive power of mixed choice for a comparison of the synchronous and asynchronous variant of the pi-calculus. With this result Palamidessi clarifies a fundamental difference

[3] In [Bou88] and [VPP07] formally slightly different conditions but with the same effect are used.

between these two versions of the pi-calculus along a problem instance which is of great relevance also in practice. Together with the encodability results in [HT91, Bou92] from the choice-free fragment of the synchronous pi-calculus into the asynchronous pi-calculus and in [Nes00] from π_s into π_a it builds the basis for our analysis of synchronous interactions in the pi-calculus.

However, analysing the results in [Pal03], we observe that we can significantly ease the definition of symmetric network and at the same time improve the separation result by abandoning the requirement on the preservation of renamings, if we instead of leader election generalise the absolute result to breaking symmetries.

4.1.3. Absolute Results and Quality Criteria

If we analyse the use of absolute results to derive separation results in the previous two sections we detect the following three observations. (1) The absolute result has to be strong enough the distinguish the source and the target language. More precisely, we have to show a positive absolute result for the source and a negative result on the same problem instance for the target language. (2) In order to prove separation a term in the source language is derived that reflects the basic distinguishing properties of the absolute result. (3) If these properties are preserved by the requirements on the encoding function this example can be used as counterexample in the proof of the separation result. This proof then basically consists of showing how the distinguishing properties of the counterexample are preserved, and in an application of the absolute impossibility result to derive a conflict. Thus, the suitability of an absolute result for the derivation of a separation result depends on whether the quality criteria required in the given setting preserves the main properties of the absolute result. Moreover, the better these properties are preserved the easier the proof of separation.

To require that a good encoding should preserve renamings or substitutions is a frequently used criterion in the context of encoding functions and makes completely sense, because it ensures that an encoding function cannot be pruned against specific examples. Indeed, even the general framework of Gorla discussed in Section 3.3 contains a similar requirement. But the unimposing side condition $\forall i \in \mathbb{N} \,.\, \sigma(i) = \theta(i)$ is rather unusual but crucial for the separation result in [Pal03], because it is necessary to preserve the main properties of the counterexample. More precisely, it is necessary to ensure that the encoding of a symmetric network again satisfies the semantic component of the symmetry definition, i.e., that $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$. The same holds for the rather complicated definition of symmetry itself. As explained above the property $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$ is crucial to rule out symmetric solutions of leader elections in π_s. Without this property leader election cannot be used to distinguish π_m and π_s. Of course, this property and also the requirement $\forall i \in \mathbb{N} \,.\, \sigma(i) = \theta(i)$ are acceptable in the current setting; maybe they are even reasonable. But they are not necessary. To avoid both, it suffices to generalise the used absolute result to some result about breaking symmetries. Note that the main argument in the negative leader election result is already that the respective network is not able to break initial symmetries of the network. So, we do not present a totally new problem instance.

Before we present our next absolute result, we define symmetric networks as used within the rest of this thesis and what it exactly means to *break symmetries*. In contrast to [Pal03], we use a simple syntactic definition of symmetry that, as mentioned above, states two processes as symmetric iff they are identical modulo some renaming according to a permutation σ on their free names.

Definition 4.1.8 (Symmetry Relation)**.** A *symmetry relation of degree* n is a permutation $\sigma : \mathcal{N} \to \mathcal{N}$, such that $\sigma^n = \mathsf{id}$.

Let $\mathsf{Sym}(n, \mathcal{N})$ denote the set of symmetry relations of degree n over $\mathcal{N}$ and let $\sigma^0 = \mathsf{id}$.

Note that this definition does not require that n is the minimal degree of σ; consequently, the condition that σ is an automorphism with only one orbit is released. A symmetric network is then a network of n processes that are equal except for some renaming according to a symmetry relation σ.

Definition 4.1.9 (Symmetric Network)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a variant of the pi-calculus introduced in Section 2.1.1 and $P \in \mathcal{P}$. Let the sequence $\tilde{x} \in \mathcal{S}(\mathcal{N})$ contain only free names of P, $n \in \mathbb{N}$, σ be a symmetry relation of degree n over $\mathcal{N} \setminus \mathsf{bn}(P)$, and $\tilde{x}$ be closed under σ, i.e., $\tilde{x} = x_1, \ldots, x_m$ and $i < m$ implies $\sigma(x_i) = x_j$ for some $j < m$. Then

$$\left[P\right]_{\sigma}^{n,\tilde{x}} = (\nu \tilde{x}) \left(\sigma^0(P) \mid \ldots \mid \sigma^{n-1}(P)\right)$$

is a *symmetric network of degree* n.

In contrast to [Pal03], we consider the network N of Example 4.1.5 as symmetric network, because $\sigma = \{\, {}^{x}/_{y}, {}^{y}/_{x} \,\}$ is a symmetry relation of degree 2 and thus $N = \left[P\right]_{\sigma}^{2,x,y}$. Note that in the following proofs, we make use of the fact that names bound in P are bound in each other process of $\left[P\right]_{\sigma}^{n,\tilde{x}}$ as well, so we explicitly forbid α-conversion here. In the following, whenever we assume some symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$, we implicitly assume the respectively quantified parameters: a variant $\langle\, \mathcal{P}, \longmapsto \,\rangle$ of the pi-calculus introduced in Section 2.1.1, a process $P \in \mathcal{P}$, a sequence $\tilde{x}$ containing only free names of P, a network size $n \in \mathbb{N}$, and a symmetry relation σ of degree n over $\mathcal{N} \setminus \mathsf{bn}(P)$.

The main difference between our definition and the definition of a symmetric network in [Pal03] is that in [Pal03] the processes of a symmetric network are numbered consecutively and for each process P_i within the symmetric network $P_{\sigma(i)} \equiv \sigma(P_i)$ holds, i.e., the symmetry relation *additionally* has to permute the indices of the processes. Accordingly, to obtain a symmetric network in the sense of [Pal03] from Example 4.1.5, we have to extend σ with the permutation $\{\, {}^{1}/_{2}, {}^{2}/_{1} \,\}$. But then, of course, N does not solve leader election anymore. Thus, each symmetric network in [Pal03] is a symmetric network for our definition, but not vice versa. Our definition of symmetry is weaker.

We use an index-guided form of substitution to replace single processes within a symmetric network.

Definition 4.1.10 (Indexed Substitution)**.** Let $\left[P\right]_{\sigma}^{n,\tilde{x}}$ be a symmetric network. An *indexed substitution* of some processes within a symmetric network, denoted by

$$\{\, i_1 \mapsto Q_1, \ldots, i_m \mapsto Q_m \,\} \left[P\right]_{\sigma}^{n,\tilde{x}}$$

for some processes $Q_1, \ldots, Q_m \in \mathcal{P}$ and $i_1, \ldots, i_m \in \{ 0, \ldots, n-1 \}$ such that for all $j, k \in \{ 1, \ldots, m \}$, $j \neq k$ implies $i_j \neq i_k$, is the result of exchanging $\sigma^{i_k}(P)$ in $\left[P\right]_{\sigma}^{n,\tilde{x}}$ with Q_k for all $k \in \{ 1, \ldots, m \}$.

Obviously $\{ i_1 \mapsto Q_1, \ldots, i_m \mapsto Q_m \} \left[P\right]_{\sigma}^{n,\tilde{x}}$ is a network; in general, however, it is not symmetric with respect to σ.

We prove that in π_{s} it is not possible to break initial symmetries, i.e., starting with a symmetric network there is always at least one execution preserving the symmetry. We refer to such an execution as *symmetric execution*. Let us consider a symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$ of degree n. Of course, if only one process does a step on its own, then all the other processes of the network can mimic this step and thus restore symmetry. So, there is a symmetry preserving execution if there is no communication between the processes of the network. The most interesting case is how the symmetry is restored after a communication between two processes of the network has temporarily destroyed it. Both cases are reflected in the proof of Theorem 4.1.13.

Apart from symmetric networks, we use the notion of a symmetric sequence of actions. Similarly to symmetric networks, in which a symmetry relation is applied to processes to derive symmetric processes, a symmetric sequence of actions is the result of applying a symmetry relation to action labels. It is sometimes necessary to translate a bound output action to an according unbound output action, because a network can send a bound name several times but only the first of this outputs will be bound.

Definition 4.1.11 (Symmetric sequence of actions)**.** Let $\mu \in \mathcal{A}_\tau$ be an action label, let $\tilde{x} \in \mathcal{S}(\mathcal{N})$ be a sequence of names and σ a symmetry relation of degree $n \in \mathbb{N}$. Then $[\mu]_{\sigma}^{n,\tilde{x}}$ denotes the sequence $\mu_1, \ldots, \mu_n$ of n labels such that $\mu_1, \ldots, \mu_n \in \mathcal{A}_\tau$, $\mu_1 = \mu$ and for $i \in \{ 2, \ldots, n \}$:

$$\mu_i = \begin{cases} \tau, & \text{if } \mu = \tau \\ \sigma^i(a)\,(b)\,, & \text{if } \mu = a\,(b) \\ \overline{\sigma^i(a)}\,\sigma^i(b)\,, & \text{if } \mu = \overline{a}\,b \text{ or } \big(\mu = \overline{a}\,(b) \text{ and} \\ & \qquad \sigma^i(b) \notin \tilde{x} \setminus \{ b, \sigma(b), \ldots, \sigma^{i-1}(b) \}\big) \\ \overline{\sigma^i(a)}\,\big(\sigma^i(b)\big)\,, & \text{if } \mu = \overline{a}\,(b) \text{ and } \sigma^i(b) \in \tilde{x} \setminus \{ b, \sigma(b), \ldots, \sigma^{i-1}(b) \} \end{cases}$$

Sometimes we refer to $\mu_2, \ldots, \mu_n$ as the symmetric counterparts of μ.

Intuitively, a symmetric execution is an execution starting from a symmetric network returning to a symmetric network after any nth step, and which is either infinite or terminates in a symmetric network. Thereby, each sequence of n steps is labelled by a symmetric sequence of actions.

Definition 4.1.12 (Symmetric Execution)**.** A *symmetric execution* is either a finite execution of length $m \cdot n \in \mathbb{N}$

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \ldots \xrightarrow{[\mu_m]_{\sigma_m}^{n,\tilde{x}_{m-1}}} \left[P_m\right]_{\sigma_m}^{n,\tilde{x}_m} \not\longrightarrow$$

for some $P_1, \ldots, P_m \in \mathcal{P}$, $\mu_1, \ldots, \mu_m \in \mathcal{A}_\tau$, $\tilde{x}_1, \ldots, \tilde{x}_m \in \mathcal{S}(\mathcal{N})$ and some $\sigma_1, \ldots, \sigma_m \in \mathsf{Sym}(n, \mathcal{N})$ such that $\sigma \subseteq \sigma_1 \subseteq \ldots \subseteq \sigma_m$ or an infinite execution

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \left[P_2\right]_{\sigma_2}^{n,\tilde{x}_2} \xrightarrow{[\mu_3]_{\sigma_3}^{n,\tilde{x}_2}} \ldots.$$

for some $P_1, P_2, \ldots \in \mathcal{P}$, $\mu_1, \mu_2, \ldots \in \mathcal{A}_\tau$, $\tilde{x}_1, \tilde{x}_2, \ldots \in \mathcal{S}(\mathcal{N})$ and $\sigma_1, \sigma_2, \ldots \in \mathsf{Sym}(n, \mathcal{N})$ such that $\sigma \subseteq \sigma_1 \subseteq \sigma_2 \subseteq \ldots$.

Note that because of $\sigma \subseteq \sigma_1 \subseteq \ldots$ the symmetry relation can only increase during a symmetric execution in a way such that existing symmetries are preserved. Moreover—as shown in Lemma 4.1.14—the symmetry relation does only grow in the presence of bound output or if a bound name is transmitted from one process to another process of the network to capture the renaming done by α-conversion. In the absence of both we have $\sigma = \sigma_1 = \ldots = \sigma_m$ and $\sigma = \sigma_1 = \sigma_2 = \ldots$ respectively.

With help of the confluence property (compare to Section 4.1.1) we prove that it is not possible to break symmetries in π_{s}. Intuitively, we show that there is at least one symmetric execution by proving, that whenever there is a step destroying symmetry, we can restore it in $n-1$ more steps mimicking the first step. The respective existence relies on the standard lemma in process calculi like the pi-calculus that transitions are preserved under substitution. As conclusion, it is not possible in π_{s} to break an initial symmetry in all executions.

Theorem 4.1.13 (Absolute Result)**.** *No symmetric network in $\mathcal{P}_{\mathrm{s}}$ can break its symmetry within a single step, i.e., every symmetric network in $\mathcal{P}_{\mathrm{s}}$ has at least one symmetric execution.*

Proof of Theorem 4.1.13. First we prove that for every symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$ in $\mathcal{P}_{\mathrm{s}}$, whenever $\left[P\right]_{\sigma}^{n,\tilde{x}}$ can perform a step then there are exactly $n-1$ more steps that restore symmetry, i.e., that lead to a symmetric network again and the corresponding n steps are labelled by a sequence of symmetric actions. Note that the main line of argumentation of this lemma is very similar to the proof of Theorem 4.2 in [Pal03] at pages 18 to 23, although we prove a completely different statement. Nevertheless, due to the different formulations of the statements, also the proofs differ in technical details.

Lemma 4.1.14.

$$\begin{aligned}
&\forall n \in \mathbb{N} \,.\, \forall \tilde{x} \in \mathcal{S}(\mathcal{N}) \,.\, \forall P \in \mathcal{P}_{\mathrm{s}} \,.\, \forall \sigma \in \mathsf{Sym}(n, \mathcal{N} \setminus \mathsf{bn}(P)) \,.\, \forall \mu \in \mathcal{A}_\tau \,.\\
&\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P} \text{ implies}\\
&\quad \exists P' \in \mathcal{P}_{\mathrm{s}} \,.\, \exists \tilde{x}' \in \mathcal{S}(\mathcal{N}) \,.\, \exists \mu_2, \ldots, \mu_n \in \mathcal{A}_\tau \,.\, \exists \sigma' \in \mathsf{Sym}(n, \mathcal{N}) \,.\\
&\quad \widehat{P} \xrightarrow{\mu_2, \ldots, \mu_n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'} \text{ and } \mu, \mu_2, \ldots, \mu_n = [\mu]_{\sigma'}^{n,\tilde{x}} \text{ and } \sigma \subseteq \sigma'
\end{aligned}$$

Proof of Lemma 4.1.14. Without loss of generality, let us assume that there are no name clashes in P, i.e., $\mathsf{bn}(P) \cap \mathsf{fn}(P) = \emptyset$ and no syntactical representation of a name is bound twice in P. Then, all name clashes in $\left[P\right]_{\sigma}^{n,\tilde{x}}$ are caused by symmetry, i.e., the syntactical representation of a name if bound in P is bound in every process of the network.

$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P}$ can be the result of either an internal μ-step of one process of the network, i.e., it can be produced without the rules Pi-ls-Com or Pi-ls-Close, or of a communication between two processes of the network, i.e., be produced by one of the rules Pi-ls-Com or Pi-ls-Close. In the first case, only one process performs a step and the rest of the network remains equal, i.e.,

$$\begin{aligned} &\exists i \in \{\, 0,\ldots,n{-}1 \,\} \;.\; \exists H \in \mathcal{P}_{\mathrm{s}} \;.\; \exists \tilde{x}_1 \in \mathcal{S}(\mathcal{N}) \;.\; \sigma^i(P) \xrightarrow{\mu} H \\ &\qquad \text{and } \widehat{P} \equiv \{\, i \mapsto H \,\} \left[P\right]_{\sigma}^{n,\tilde{x}_1} \end{aligned} \tag{C1}$$

In the second case, $\mu = \tau$ and two processes of the network change, i.e.,

$$\begin{aligned} &\exists i,j \in \{\, 0,\ldots,n{-}1 \,\} \;.\; \exists H_1,H_2 \in \mathcal{P}_{\mathrm{s}} \;.\; \exists z,z' \in \mathcal{N} \;.\; i \neq j \\ &\quad \text{and } \Big(\sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} H_1 \mid H_2 \;\text{ or }\; \sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} (\nu z,z')\,(H_1 \mid H_2)\Big) \\ &\quad \text{and } \widehat{P} \equiv \{\, i \mapsto H_1, j \mapsto H_2 \,\} \left[P\right]_{\sigma'}^{n,\tilde{x}'} \end{aligned} \tag{C2}$$

We proceed with a case split.

Case (C1): Within the symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$, all processes $\sigma^i(P)$ for $0 \leq i \leq n{-}1$ are equal except for some renaming of free names according to σ. Thus, whenever a process $\sigma^i(P)$ can perform a step $\xrightarrow{\mu}$ then each other process $\sigma^k(P)$ of the network can mimic this step by $\xrightarrow{\mu'}$, where μ' is the result of applying σ^{k-i+n} to μ possibly by changing bound output to unbound output as described in Definition 4.1.11.[4] The case of a bound output action μ is slightly tricky, so we consider the other cases first.

If μ is no bound output, then we can choose the labels $\mu_2,\ldots,\mu_n$ such that $\mu,\mu_2,\ldots,\mu_n = [\mu]_{\sigma}^{n,\tilde{x}}$. Moreover, by symmetry $\sigma^i(P) \xrightarrow{\mu} H$ implies $\sigma^k(P) \xrightarrow{\mu'} \sigma^{k-i+n}(H)$ for a $H \in \mathcal{P}_{\mathrm{s}}$. With it, we can restore symmetry by mimicking the μ-step of process $\sigma^i(P)$ by the $n{-}1$ steps $\sigma^{i+1}(P) \xrightarrow{\mu_2} \sigma(H)$, …, $\sigma^{n-1}(P) \xrightarrow{\mu_{n-i}} \sigma^{n-1-i}(H)$, $\sigma^0(P) \xrightarrow{\mu_{n-i+1}} \sigma^{n-i}(H)$, …, $\sigma^{i-1}(P) \xrightarrow{\mu_n} \sigma^{n-1}(H)$. These n steps build the chain

$$\begin{aligned}
\left[P\right]_{\sigma}^{n,\tilde{x}} \quad &\xrightarrow{\mu} \quad (\nu\tilde{x}_1)\,\big(\sigma^0(P) \mid \ldots \mid \sigma^{i-1}(P) \mid \sigma^0(H) \mid \sigma^{i+1}(P) \mid \ldots \\
&\qquad\qquad \mid \sigma^{n-1}(P)\big) \\
&\;\vdots \\
&\xrightarrow{\mu_{n-i}} \quad (\nu\tilde{x}_{n-i})\,\big(\sigma^0(P) \mid \ldots \mid \sigma^{i-1}(P) \mid \sigma^0(H) \mid \ldots \\
&\qquad\qquad \mid \sigma^{n-1-i}(H)\big) \\
&\xrightarrow{\mu_{n-i+1}} \quad (\nu\tilde{x}_{n-i+1})\,\big(\sigma^{n-i}(H) \mid \sigma(P) \mid \ldots \mid \sigma^{i-1}(P) \mid \sigma^0(H) \\
&\qquad\qquad \mid \ldots \mid \sigma^{n-1-i}(H)\big) \\
&\;\vdots \\
&\xrightarrow{\mu_n} \quad (\nu\tilde{x}_n)\,\big(\sigma^{n-i}(H) \mid \ldots \mid \sigma^{n-1}(H) \mid \sigma^0(H) \mid \ldots \\
&\qquad\qquad \mid \sigma^{n-1-i}(H)\big)
\end{aligned}$$

[4] Note that n is added in $k{-}i{+}n$ and $k{-}j{+}n$ just to ensure that both values are positive. Because $\sigma^n = \mathsf{id}$ if $k{-}i \geq 0$ we have $\sigma^{k-i+n} = \sigma^{k-i}$.

with $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathcal{S}(\mathcal{N})$ and $\tilde{x}' = \tilde{x}_n$. Because of $\sigma^n = \mathsf{id}$ after the last step, we result in a network which is again symmetric with respect to σ, i.e., we choose $\sigma' = \sigma$. Hence, we can choose $P' = \sigma^{n-i}(H)$ such that $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\mu} \widehat{P} \xrightarrow{\mu_2,\ldots,\mu_n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'}$.

If μ is equal to τ, an input, or an unbound output action, then so are its symmetric counterparts $\mu_2, \ldots, \mu_n$. We choose $\tilde{x}' = \tilde{x}_1 = \ldots = \tilde{x}_n = \tilde{x}$ and are done.

If μ is a bound output action $\overline{y}\,(z)$, then we have to consider two cases.

Case $z \notin \mathsf{bn}\big(\sigma^i(P)\big)$**:** Here, $z \in \mathsf{fn}\big(\sigma^i(P)\big)$ and because μ is a bound output z must be in $\tilde{x}$. So we have to choose $\tilde{x}_1 = \tilde{x} \setminus \{\, z \,\}$. Then, by Definition 4.1.11 some of the actions $\mu_2, \ldots, \mu_n$ might be bound and some might be unbound outputs depending on whether $\sigma^{j-1}(z)$ was already the subject of an earlier bound output of this sequence or not. If $\sigma^{j-1}(z)$ of μ_j was already the subject of a bound output within $\mu, \mu_2, \ldots, \mu_{j-1}$, then μ_j is an unbound output and we choose $\tilde{x}_j = \tilde{x}_{j-1}$, else μ_j is a bound output and we choose $\tilde{x}_j = \tilde{x}_{j-1} \setminus \sigma^{j-1}(z)$ for all $j \in \{\, 2, \ldots, n \,\}$. Again, we can choose $\sigma' = \sigma$ and $P' = \sigma^{n-i}(H)$ and proceed as in the case where μ is not a bound output.

Case $z \in \mathsf{bn}\big(\sigma^i(P)\big)$**:** Here, by symmetry, $\sigma^j(z) = z$ is bound in $\sigma^{i+j}(P)$ for all $j \in \{\, 0, \ldots, n-1 \,\}$. By the above assumption that there are no name clashes (except for the duplicate binding of names), we conclude $z \notin \tilde{x}$. Then, by Definition 4.1.11, $\mu, \mu_2, \ldots, \mu_n$ is a sequence of n bound output actions. Each of these actions μ_j changes the scope of $\sigma^{i+j-1}(P)$ (in a symmetric way to the other processes) but the scope of the network is left unchanged. So, we can again choose $\tilde{x}' = \tilde{x}_1 = \ldots = \tilde{x}_n = \tilde{x}$. The crux is that performing the first bound output with label μ may force an α-conversion to avoid name clashes to the other bound instances of z in the other processes of the network such that the symmetry is destroyed. To illustrate this problem, let us consider an example:

Example 4.1.15. Let

$$N \triangleq (\nu x)\,\overline{a}\langle x\rangle\,.\overline{x} \mid (\nu x)\,\overline{a}\langle x\rangle\,.\overline{x} = \left[\,(\nu x)\,\overline{a}\langle x\rangle\,.\overline{x}\right]_{\mathsf{id}}^{2}.$$

N can perform two bound outputs $\overline{a}\,(x)$. To avoid name capture we have to apply α-conversion such that we have $N \xrightarrow{\overline{a}(x)} \overline{x} \mid (\nu x')\,\overline{a}\langle x'\rangle\,.\overline{x'} \xrightarrow{\overline{a}(x')} \overline{x} \mid \overline{x'}$. Because of this α-conversion, we result in a network which is not symmetric with respect to id. Nevertheless, the first step is mimicked by the second step and, thus, both parts of the network behave symmetrically. As consequence to our intuition, the resulting network should be again considered as symmetric network. To overcome this problem, we record the renaming done by α-conversion in $\sigma' = \{\, {}^{x}/_{x'}, {}^{x'}/_{x} \,\}$ such that $\overline{x} \mid \overline{x'} = \left[\overline{x}\right]_{\sigma'}^{2}$. Note that because of this, σ' can only increase by adding permutations on formerly bound names and fresh names.

That is why we have to increase the symmetry relation in this case to keep track of the renaming done by α-conversion. Thereto, we enforce the α-conversion after the first bound output to rename all instances of z (except the

first one) to a different fresh name for each process of the network and add the respective permutations of z to σ in order to obtain σ' such that $\sigma \subseteq \sigma'$. Afterwards, we can choose $\mu_2, \ldots, \mu_n$ such that $\mu, \mu_2, \ldots, \mu_n = [\mu]^{n,\tilde{x}}_{\sigma'}$ and $P' = \sigma'^{n-i}(H)$ and proceed as in the case where μ is not a bound output.

Case (C2): In this case, there is a communication between $\sigma^i(P)$ and $\sigma^j(P)$ as result of one of the rules Pi-ls-Com or Pi-ls-Close. Without loss of generality, let us assume that $\sigma^i(P)$ is the sender and $\sigma^j(P)$ is the receiver of this communication, i.e., there are $y, z_1, z_2 \in \mathcal{N}$ such that $\sigma^i(P) \xrightarrow{\overline{y}\, z_1} H_1$ (or $\sigma^i(P) \xrightarrow{\overline{y}(z_1)} H_1$) and $\sigma^j(P) \xrightarrow{y(z_2)} \{\, {}^{z_2}/_{z_1} \,\}(H_2)$. Because of symmetry, each process $\sigma^k(P)$ for $0 \leq k \leq n{-}1$ can perform an output action $\mu_{\mathsf{out},k} = \overline{\sigma^{k-i+n}(y)}\, \sigma^{k-i+n}(z_1)$ (or $\mu_{\mathsf{out},k} = \overline{\sigma^{k-i+n}(y)}\, (\sigma^{k-i+n}(z_1))$) and an input action $\mu_{\mathsf{in},k} = \sigma^{k-j+n}(y)\, z_2$ such that $\sigma^k(P) \xrightarrow{\mu_{\mathsf{out},k}} \sigma^{k-i+n}(H_1)$ and $\sigma^k(P) \xrightarrow{\mu_{\mathsf{in},k}} \sigma^{k-j+n}(\{\, {}^{z_2}/_{z_1} \,\}(H_2))$. Because of the confluence Lemma 4.1.1, i.e., without mixed-choice an output action cannot block an alternative input action (within one step) and vice versa[5], as depicted in Figure 4.2 (case of unbound output) process $\sigma^k(P)$ must be able to perform both actions consecutively in arbitrary order resulting in the same term which we denote by Q_k. To restore symmetry, we build a chain of n steps such that

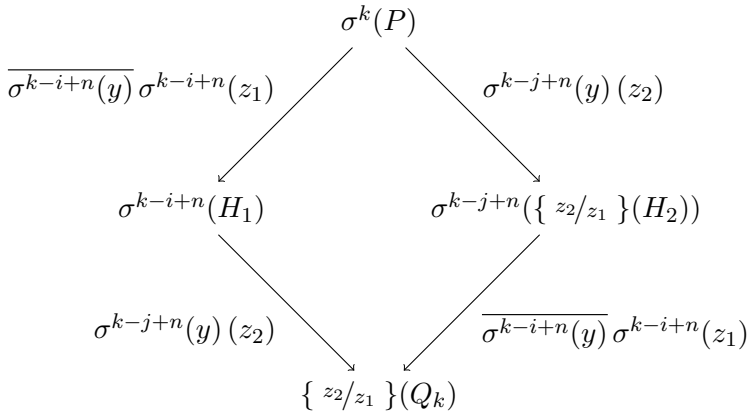


Figure 4.2.: Local confluence of receiving and sending actions.

each process $\sigma^k(P)$ performs the output action $\mu_{\mathsf{out},k}$ in step $((k{-}i{+}n) \bmod n){+}1$ and the input action $\mu_{\mathsf{in},k}$ in step $((k{-}j{+}n) \bmod n){+}1$, i.e., each process is once a sender and once a receiver and $\mu = \mu_2 = \ldots = \mu_n = \tau$ and with that $\mu, \mu_2, \ldots, \mu_n = [\mu]^{n,\tilde{x}}_{\sigma}$. Again, we consider the case of unbound outputs by first. Then, we have $\left[P\right]^{n,\tilde{x}}_{\sigma} \xrightarrow{\tau} \{\, i \mapsto H_1, j \mapsto H_2 \,\} \left[P\right]^{n,\tilde{x}}_{\sigma}$ as first step with $\sigma^i(P) \xrightarrow{\overline{y}\, z_1}$

[5] Indeed without mixed choice the only possibility for $\sigma^k(P)$ to be able to perform both actions is that these two actions are composed in parallel, so $\sigma^k(P)$ can perform both actions in an arbitrary order and it is not possible that performing one of these actions alone prevents $\sigma^k(P)$ from performing the other one next.

H_1, $\sigma^j(P) \xrightarrow{y(z_2)} \{\, {}^{z_2}/_{z_1} \,\}(H_2)$ and $\sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} H_1 \mid H_2$. Depending on the values of i and j, some of the processes perform the corresponding input action first while others perform at first the corresponding output action. Because of Lemma 4.1.1, both is possible. We let each process perform exactly these two actions (compare to Figure 4.2). We choose $P' = Q_0$ and $\tilde{x}' = \tilde{x}$. We start with a symmetric network and all processes behave symmetrically, i.e., each process mimic the behaviour of its neighbour, so we have $Q_k = \sigma^k(Q_0)$ for all k with $0 \leq k \leq n{-}1$ such that can choose $\sigma' = \sigma$ and have

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\tau}{}^{n} \left[P'\right]_{\sigma'}^{n,\tilde{x}'}.$$

Now we consider the case of bound outputs. Note that $\sigma^i(P)$ and $\sigma^j(P)$ perform a communication step *within* the network, so if $\sigma^i(P)$ performs a bound output z_1 must be bound in $\sigma^i(P)$, i.e., $z_1 \notin \tilde{x}$. By symmetry $\sigma^l(z_1) = z_1$ is bound in $\sigma^{i+l}(P)$ for all $l \in \{\, 0, \ldots, n{-}1 \,\}$. With that either all output action are bound or all are unbound. In case of bound output we have $\sigma^i(P) \mid \sigma^j(P) \xrightarrow{\tau} (\nu z, z')\,(H_1 \mid H_2)$, because first we have to apply α-conversion to rename the instance of z_1 bound in $\sigma^j(P)$ and then the bound output by $\sigma^i(P)$ leads to a scope extrusion such that $z = z_1$ and z' is the renaming of z_1 in $\sigma^j(P)$. Again we use α-conversion after the first communication step to rename all instances of z_1 (except the first) to a different fresh name for each process of the network and add the respective permutations of z_1 to σ in order to obtain σ' such that $\sigma \subseteq \sigma'$. Let $z_{1,2}, \ldots, z_{1,n}$ denote the sequence of names used to rename z_1 according to σ'. We proceed as in the case of unbound outputs with the $n{-}1$ communication steps as described above. Of course we have to replace the processes $\sigma^k(P)$ by $\{\, {}^{z_{1,2}}/_{z_1}, \ldots, {}^{z_1}/_{z_{1,n}} \,\}^{k-i}(P)$ and $\mu_{\mathsf{out},k}$ by $\overline{\sigma^{k-i+n}(y)}\left[\{\, {}^{z_{1,2}}/_{z_1}, \ldots, {}^{z_1}/_{z_{1,n}} \,\}^{k-i}(z_1)\right]$ for $0 \leq k \leq n{-}1$. After completing these n communication steps the names $z_1, z_{1,2}, \ldots, z_{1,n}$ are pulled outwards by scope extrusion, i.e., we have

$$\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{\tau} \{\, i \mapsto H_1, j \mapsto H_2 \,\} \left[P\right]_{\sigma}^{n,\tilde{x},z_1,z_{1,2}} \xrightarrow{\tau}{}^{n-1} \left[R\right]_{\sigma'}^{n,\tilde{x}'},$$

where $\tilde{x}' = \tilde{x}, z_1, z_{1,2}, \ldots, z_{1,n}$ and $P \xrightarrow{\overline{\sigma^{n-i}(y)}(z_1),\sigma^{n-j}(y)(z_2)} R$. With that we can choose $P' = R$ and are done.

□

With Lemma 4.1.14, we can now construct the symmetric execution. We start with an arbitrary symmetric network $\left[P\right]_{\sigma}^{n,\tilde{x}}$. If $\left[P\right]_{\sigma}^{n,\tilde{x}} \not\rightarrow$ we have a symmetric execution of length 0. Otherwise, if $\left[P\right]_{\sigma}^{n,\tilde{x}}$ can perform a step labelled by μ_1 by Lemma 4.1.14 we can perform $n{-}1$ more steps such that $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1}$. Now we can proceed alike with $\left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1}$ and result either in a finite symmetric execution of length n or we have $\left[P\right]_{\sigma}^{n,\tilde{x}} \xrightarrow{[\mu_1]_{\sigma_1}^{n,\tilde{x}}} \left[P_1\right]_{\sigma_1}^{n,\tilde{x}_1} \xrightarrow{[\mu_2]_{\sigma_2}^{n,\tilde{x}_1}} \left[P_2\right]_{\sigma_2}^{n,\tilde{x}_2}$. By recursively repeating this argument, we either get a finite or an infinite symmetric execution. □

Note that Theorem 4.1.13 does not state anything about encodability and it does not need a notion of reasonableness either. Instead, it just states without any precondition that every symmetric network in $\mathcal{P}_{\mathrm{s}}$ has at least one symmetric execution. In contrast, there are symmetric networks in $\mathcal{P}_{\mathrm{m}}$ without such a symmetric execution, as the following example shows.

Example 4.1.16. Consider the network

$$(\nu x, y)\,(P \mid \sigma(P)) \quad \text{with} \quad P = \overline{x}.\overline{1} + y.\overline{2} \quad \text{and} \quad \sigma = \{\, {}^{x}/_{y}, {}^{y}/_{x}, {}^{1}/_{2}, {}^{2}/_{1} \,\}$$

with $\sigma^2 = \mathsf{id}$, i.e., $(\nu x, y)\,(P \mid \sigma(P))$ is a symmetric network in $\mathcal{P}_{\mathrm{m}}$. It has, modulo structural congruence, exactly the two following executions

$$(\nu x, y)\,(P \mid \sigma(P)) \xrightarrow{\tau} \overline{1} \mid \overline{1} \xrightarrow{\overline{1}} \overline{1} \xrightarrow{\overline{1}} 0$$
$$(\nu x, y)\,(P \mid \sigma(P)) \xrightarrow{\tau} \overline{2} \mid \overline{2} \xrightarrow{\overline{2}} \overline{2} \xrightarrow{\overline{2}} 0$$

and even none of them is symmetric; the initial symmetry is broken.

So Theorem 4.1.13 proves a difference in the absolute expressive power between π_{m} and π_{s}.

We conclude that the absolute result used by [Pal03] and reviewed in Section 4.1.2 is stricter than necessary. The weaker absolute result augmenting the ability to break initial symmetries is already sufficient to prove a fundamental difference between π_{m} and π_{s}. Moreover, we can abandon the additional property $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$ on symmetric networks and present a very simple definition instead. As shown in the next section, this absolute result is not only suited to derive a separation result similar to [Pal03] but we can also abandon one of the requirements on encoding functions and, thus, obtain a stronger separation result. The complicated definition of symmetry necessary to use leader election as distinguishing problem description induces the necessity of an additional requirement on the quality of encodings, namely that $[\![\, \sigma(P) \,]\!] = \theta([\![\, P \,]\!])$ such that $\forall i \in \mathbb{N} \,.\, \sigma(i) = \theta(i)$. Such a requirement is not necessary to preserve the pure syntactic definition of symmetry which allows to distinguish π_{m} and π_{s} by their ability to break initial symmetries. Moreover, simpler, i.e., more general, absolute results usually lead to simpler proofs of separation results, because it is often easier to show the preservation of the respective distinguishing properties of the counterexample with respect to the weaker set of quality criteria. This shows how a separation result can benefit from the choice of a better suited absolute result. However, obtaining suitable problem descriptions is often not an easy task. Usually, it requires an exhaustive study of the source and the target language and their differences. Moreover, choosing a suitable problem instance is a balancing act between absolute results strong enough to distinguish the source and target language and to allow for the derivation of a suitable counterexample, and results so weak that the main properties of the counterexample are preserved with respect to the quality criteria in the current setting. However, absolute results do not have to be optimal to derive interesting and important results and they always provide further insights in the differences between the languages.

4.2. Separation and Quality Criteria

The main purpose of this section is to show how separation results with respect to different sets of quality criteria can be derived from a single absolute result, namely Theorem 4.1.13. In all separation results of this section we compare again π_{m} and π_{s}. The first part in Section 4.2.1 considers sets of quality criteria that share the requirement on the homomorphic translation of the parallel operator. Since the homomorphic translation of the parallel operator naturally preserves symmetry of source term networks, the absolute result is well-suited for all set of criteria in the first part, which leads to considerable short and simple proofs of the separation results. They all follow the same line of argumentation which is typical for separation results on top of absolute results.

In the second part in Section 4.2.2 the homomorphic translation of the parallel operator is replaced by the weaker requirement on compositionality and the preservation of distributability. In this sense, we strengthen the results of the first part. However, we replace the different settings in the first part by the general framework of Section 3.3, whose criteria are stricter than the requirements in the different sets of the first part. Hence, the results of the first and the second part are in fact incomparable. Unfortunately, the absolute result is not well-suited for the criteria in the general framework, because they do not preserve symmetry of source terms. Hence, proving separation becomes much more complicated and is less intuitive. However, the absolute result still allows for some helpful arguments in the proof. Moreover, in Section 4.4 we present an absolute result better suited in the context of the required criteria and a separation result based on it that sheds more light on the difference between π_{m} and π_{s}.

In Section 4.2.2 two separation results with respect to different domains are derived. First we consider distributability and then causality. In order to prove results for both domains, we show first a condition of all good encodings from π_{m} into π_{s} that is then used in the separation results. Note that we choose a very simple example of separation with respect to different domains here, because under the presented setting and with respect to the exploited properties of good encodings between π_{m} and π_{s} the two domains are closely related.

4.2.1. Different Sets of Quality Criteria

Palamidessi in [Pal03] proves that there exists no uniform and reasonable encoding from π_{m} into π_{s} based on an absolute result concerning the ability of these languages to solve leader election in symmetric networks (see Section 4.1.2). Later on Gorla in [Gor10b] proves a similar result with a considerable simpler proof with respect to his general framework as reviewed in Section 3.3 and the additionally requirements that $\asymp$ is exact, i.e., if $T \asymp T'$ then $T \xrightarrow{\mu}$ implies $T' \stackrel{\mu}{\Longrightarrow}$ for all $T, T' \in \mathcal{P}_{\mathrm{s}}$ and $\mu \in \mathcal{A}$, and that the parallel operator is translated homomorphically. Moreover, he proves separation between π_{m} and π_{s} in two more settings that do not require the homomorphic translation of the parallel operator. Both settings are based again on the criteria of the general framework. The second setting requires additionally that $\asymp$ is reduction sensitive, i.e., if $T \asymp T'$ then $T \longmapsto$ implies $T' \longmapsto$ for all $T, T' \in \mathcal{P}_{\mathrm{s}}$, which leads to prompt encodings. Whereas the

third setting uses a stricter form of operational correspondence requiring that all leftovers of former emulations, i.e., junk, in target terms are composed in parallel to the encoding of the respective continuation.

Similarly, we will also consider the separation between π_{m} and π_{s} in three different settings, but require in all three settings that the parallel operator is translated homomorphically. Moreover, we base all three separation results on the absolute result of Theorem 4.1.13. It is no real surprise that this absolute result leads to differences in the translational expressiveness of the languages. Because the homomorphic translation of the parallel operator preserves the symmetric nature of π_{m}-terms, the absolute result directly leads to counterexamples for respective encodings. Hence, the non-existence of a reasonable encoding from π_{m} into π_{s} that translates the parallel operator homomorphically is a natural consequence of the difference in their absolute expressiveness. Unfortunately, there is no agreement on the minimal requirements of a reasonable encoding, so we cannot formally prove this result in general, although we believe that it holds for any meaningful definition of reasonableness. Instead, to underpin our assertion, we prove it in the settings of [Pal03] and (the first setting of) [Gor10b].

According to [Pal03], an encoding is "uniform" if it translates the parallel operator homomorphically and preserves renamings, i.e., for all permutations of names σ there exists a permutation of names θ such that $[\![\, \sigma(P) \,]\!] = \theta([\![\, P \,]\!])$ that also fixes indexes. Vigliotti et al. [VPP07] additionally require that the permutations σ and θ are compatible on observables. Gorla [Gor10b] does not use the notion of uniformity, but in his first setting the separation result between π_{m} and π_{s} does also assume the homomorphic translation of the parallel operator. Moreover, he specifies name invariance as a criterion for a good encoding, which is a more complex condition than Palamidessi's second condition, but comes without the strict side condition $\forall i \in \mathbb{N} \,.\, \sigma(i) = \theta(i)$. It turns out that in our setting we do not need a second condition like renaming preservation or name invariance, because we base our counterexamples in the following separation results on symmetric networks of the form $P \mid P$ as already Gorla did in [Gor10b]. For us, an encoding is uniform iff it translates the parallel operator homomorphically.

Definition 4.2.1 (Uniform encoding)**.** An encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ is a *uniform encoding* if, for all $P, Q \in \mathcal{P}_{\mathrm{S}}$

$$[\![\, P \mid Q \,]\!] = [\![\, P \,]\!] \mid [\![\, Q \,]\!] \tag{U}$$

Actually, Theorem 4.1.13 should suffice to prove that there cannot be a uniform and reasonable encoding from π_{m} into π_{s}, because such encodings preserve symmetry and it is possible to break symmetries in π_{m} while this is not possible in π_{s}. The problem is that there is no commonly accepted notion of reasonableness. For separation results, we seek a definition of reasonableness that is as weak as possible. But, without any notion of reasonableness, the theorem would not hold. For instance, we could simply translate everything to $\mathsf{0}$ (modulo $\equiv$). Of course such an encoding makes no sense and so hardly anyone would call it reasonable. Usually, an encoding is called reasonable if it preserves some kind of behaviour or the ability to solve some kind of problem so to ensure that the

purpose of the original term is preserved. In the following, we consider three different notions of reasonableness.

Version 1. For Palamidessi, an encoding is reasonable if it preserves the relevant observables and termination properties [Pal03]. Implicitly, she requires that a reasonable encoding should at least preserve the ability to solve leader election. We do alike but with a different interpretation of what it means to solve leader election that is more closely related to the definition used by Bougé [Bou88]: A network is said to solve leader election iff in each execution exactly one process propagates itself as leader while all the other processes propagate themselves as slaves. We assume the existence of two different predetermined output actions, one to propagate as leader (μ_l) and the other to propagate as slave (μ_s). Moreover, we require that for both output actions neither the channel names nor the sent values are bound within the network[6].

Definition 4.2.2 (Solving Leader Election)**.** Let $\mu_l, \mu_s \in \mathcal{A}$ be two different output action labels, i.e., $\mu_l \neq \mu_s$. A network N of size $n > 1$ *solves leader election* if every maximal execution of N contains exactly one step labelled by μ_l and $n-1$ steps labelled by μ_s and all names of μ_l and μ_s are free in N.

The main difference to the definition of leader election used in [Pal03] is that here the slaves do not have to know the identity, i.e., the index, of the leader. So, this definition is usually considered as a weaker notion of the leader election problem. An encoding is now said to be reasonable iff it preserves the ability to solve the leader election problem.

Definition 4.2.3 (1-Reasonableness)**.** An encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$ is *1-reasonable*, if $[\![P]\!]$ solves leader election iff P solves leader election, for all $P \in \mathcal{P}_{\mathrm{m}}$.

To prove that there is no uniform and reasonable encoding, we force our encoding to lead to a network of two processes that is symmetric with respect to identity. By Theorem 4.1.13, this network has at least one symmetric execution. Because we use the identity as symmetry relation, in the symmetric execution both processes behave exactly the same; in particular if one of them propagates himself as leader then the other one does alike, which contradicts leader election.

Theorem 4.2.4 (Separation Result)**.** *There is no uniform and 1-reasonable encoding from π_{m} into π_{s}.*

Proof. Let us assume the contrary, i.e., there is a uniform and 1-reasonable encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P = a.\overline{slave} + \overline{a}.\overline{leader}$$

Obviously $\sigma = \mathsf{id}$ is a symmetry relation of degree 2. Because of that, the network $N = \left[a.\overline{slave} + \overline{a}.\overline{leader}\right]^2_\sigma$ is a symmetric network. Moreover N solves leader election, because the leader sends an empty message over channel *leader* and all slaves send

[6] Note that if we allow bound names in these output actions, we could hardly predetermine them.

an empty message over channel *slave*. Since $[\![\cdot]\!]$ is uniform, we have $[\![P \mid P]\!] \overset{(U)}{=} [\![P]\!] \mid [\![P]\!] = \left[[\![P]\!] \right]^2_{\mathsf{id}}$, i.e., $[\![N]\!]$ is again a symmetric network of degree 2 with id as symmetry relation. By Theorem 4.1.13, $[\![N]\!]$ has at least one symmetric execution and by reasonableness $[\![N]\!]$ must solve leader election, i.e., in every maximal execution there is exactly one process that propagates itself as leader by an output action. Let μ_l denote this send action. By Definition 4.1.12, a symmetric execution has symmetric sequences of actions, i.e., the action μ_l is coupled to its symmetric counterpart building the sequence $[\mu_l]^{2,\tilde{z}'}_{\sigma'}$ for some $\tilde{z}' \in \mathcal{S}(\mathcal{N})$ and $\sigma' \in \mathsf{Sym}(2,\mathcal{N})$. By construction in the proof of Lemma 4.1.14, and because we start with id, we know that σ' consists of (permutations of) names that are bound in $[\![N]\!]$ or fresh. Because, by definition, μ_l can neither contain fresh nor bound names, we conclude $[\mu_l]^{2,\tilde{z}'}_{\sigma'} = \mu_l, \mu_l$, i.e., the output action appears twice in the symmetric execution. So two processes propagate themselves as leader, which is a contradiction. □

Note that in contrast to the proof of Palamidessi [Pal03, VPP07] we do not have to assume that the encoding preserves renamings.

Version 2. For the second (and also the third setting) we introduce a technical lemma. Intuitively, it states that the symmetric execution of a symmetric network of degree n, where n is *not* the minimal degree of the corresponding symmetry relation, can be subdivided into symmetric executions on symmetric subnetworks of the original network.

Lemma 4.2.5 (Absolute Result)**.** *Let $\left[P_0\right]^{n,\tilde{x}}_{\sigma}$ be a symmetric network in $\mathcal{P}_{\mathrm{s}}$. If the degree of σ is not minimal, i.e., if there is a $n' \in \mathbb{N}$ with $0 < n' < n$ such that $\sigma^{n'} = \mathsf{id}$, then $\left[P_0\right]^{n,\tilde{x}}_{\sigma}$ has a finite or an infinite symmetric execution*

$$\left[P_0\right]^{n,\tilde{x}}_{\sigma} \xrightarrow{[\mu_1]^{n,\tilde{x}}_{\sigma_1}} \left[P_1\right]^{n,\tilde{x}_1}_{\sigma_1} \xrightarrow{[\mu_2]^{n,\tilde{x}_1}_{\sigma_2}} \dots \xrightarrow{[\mu_m]^{n,\tilde{x}_{m-1}}_{\sigma_m}} \left[P_m\right]^{n,\tilde{x}_m}_{\sigma_m} \not\longrightarrow$$
$$\text{or} \quad \left[P_0\right]^{n,\tilde{x}}_{\sigma} \xrightarrow{[\mu_1]^{n,\tilde{x}}_{\sigma_1}} \left[P_1\right]^{n,\tilde{x}_1}_{\sigma_1} \xrightarrow{[\mu_2]^{n,\tilde{x}_1}_{\sigma_2}} \dots$$

for some $m \in \mathbb{N}$, $P_1, \dots, P_m \in \mathcal{P}_{\mathrm{s}}$, $\sigma_1, \dots, \sigma_m \in \mathsf{Sym}(n,\mathcal{N})$ with $\sigma \subseteq \sigma_1 \subseteq \dots \subseteq \sigma_m$, $\tilde{x}_1, \dots, \tilde{x}_m \in \mathcal{S}(\mathcal{N})$ and $\mu_1, \dots, \mu_m \in \mathcal{A}_\tau$ or some $P_1, P_2, \dots \in \mathcal{P}_{\mathrm{s}}$, $\sigma_1, \sigma_2, \dots \in \mathsf{Sym}(n,\mathcal{N})$ with $\sigma \subseteq \sigma_1 \subseteq \sigma_2 \subseteq \dots$, some $\tilde{x}_1, \tilde{x}_2, \dots \in \mathcal{S}(\mathcal{N})$ and $\mu_1, \mu_2, \dots \in \mathcal{A}_\tau$, respectively, such that $\left[P_0\right]^{n',\tilde{x}}_{\sigma}$ has the finite or infinite symmetric execution

$$\left[P_0\right]^{n',\tilde{x}'}_{\sigma} \xrightarrow{[\mu'_1]^{n',\tilde{x}'}_{\sigma'_1}} \left[P_1\right]^{n',\tilde{x}'_1}_{\sigma'_1} \xrightarrow{[\mu'_2]^{n',\tilde{x}'_1}_{\sigma'_2}} \dots \xrightarrow{[\mu'_m]^{n',\tilde{x}'_{m-1}}_{\sigma'_m}} \left[P_m\right]^{n',\tilde{x}'_m}_{\sigma'_m} \not\longrightarrow$$
$$\text{or} \quad \left[P_0\right]^{n',\tilde{x}}_{\sigma} \xrightarrow{[\mu'_1]^{n',\tilde{x}}_{\sigma'_1}} \left[P_1\right]^{n',\tilde{x}'_1}_{\sigma'_1} \xrightarrow{[\mu'_2]^{n',\tilde{x}'_1}_{\sigma'_2}} \dots$$

for some $\tilde{x}'_1, \dots, \tilde{x}'_m \in \mathcal{S}(\mathcal{N})$, $\mu'_1, \dots, \mu'_m \in \mathcal{A}_\tau$ and $\sigma'_1, \dots, \sigma'_m \in \mathsf{Sym}(n',\mathcal{N})$ with $\sigma \subseteq \sigma'_1 \subseteq \dots \subseteq \sigma'_m$ or some $\tilde{x}'_1, \tilde{x}'_2, \dots \in \mathcal{S}(\mathcal{N})$, $\mu'_1, \mu'_2, \dots \in \mathcal{A}_\tau$ and $\sigma'_1, \sigma'_2, \dots \in \mathsf{Sym}(n',\mathcal{N})$ with $\sigma \subseteq \sigma'_1 \subseteq \sigma'_2 \subseteq \dots$ respectively such that $\tilde{x}'$ is a subsequence of $\tilde{x}$, $\tilde{x}'_i$ is a subsequence

of $\tilde{x}_i$ *and either* μ'_i *or if* μ'_i *is a bound output its unbound variant is in* $[\mu_i]^{n,\tilde{x}_{i-1}}_{\sigma_i}$ *for all* $i \in \{ 1, \ldots, m \}$ *or* $i \in \mathbb{N}$ *respectively.*

Note that, like Theorem 4.1.13, this result is absolute in the sense that it holds independently of any notion of uniformity or reasonableness.

The proof is based on the following observation: every network of degree n that is symmetric with respect to a symmetry relation σ such that n is *not* the minimal degree of σ can be subdivided into several identical symmetric networks with respect to σ. Then, an induction on the number of sequences of n steps from a symmetric network to a symmetric network is performed. The inductive step is proved by a case analysis on whether the first step of such a sequence is due to an action of only one process of the network or to a communication between two processes.

Proof. Assume there is a $0 < n' < n$ such that $\sigma^{n'} = \mathsf{id}$. Then because $\sigma^n = \mathsf{id}$ there must be a $k \in \mathbb{N}$ such that $n = k * n'$. Because $\sigma^0 = \sigma^{n'} = \sigma^{i*n'}$ for each $i \in \{ 1, \ldots, k \}$ we have $\sigma^j = \sigma^{j+n'}$. So, for each $P' \in \mathcal{P}_{\mathrm{s}}$ and each $\mu' \in \mathcal{A}_\tau$, $\left[P'\right]^{n,\tilde{x}}_{\sigma}$ can be divided into k identical symmetric networks such that $\left[P'\right]^{n,\tilde{x}}_{\sigma} = (\nu\tilde{x})\left(\left[P'\right]^{n'}_{\sigma} \mid \ldots \mid \left[P'\right]^{n'}_{\sigma}\right)$ and $[\mu']^{n,\tilde{x}}_{\sigma}$ can be divided in k sequences such that $[\mu']^{n,\tilde{x}}_{\sigma}$ and $[\mu']^{n',\tilde{x}}_{\sigma}, \ldots, [\mu']^{n',\tilde{x}}_{\sigma}$ differ only by replacing some unbound outputs on a name in $\tilde{x}$ by the respective bound outputs if the respective step is a bound output for each subnetwork $\left[P'\right]^{n',\tilde{x}}_{\sigma}$.

If $\left[P_0\right]^{n,\tilde{x}}_{\sigma}$ has a symmetric execution of length 0, i.e., $\left[P_0\right]^{n,\tilde{x}}_{\sigma} \not\longrightarrow$, then of course we have $\left[P_0\right]^{n',\tilde{x}}_{\sigma} \not\longrightarrow$ as well and so $\left[P_0\right]^{n',\tilde{x}}_{\sigma}$ has a symmetric execution of length 0.

Else we consider an arbitrary sequence of n steps

$$\left[P_k\right]^{n,\tilde{x}_k}_{\sigma_k} \xrightarrow{[\mu_{k+1}]^{n,\tilde{x}_k}_{\sigma_{k+1}}} \left[P_{k+1}\right]^{n,\tilde{x}_{k+1}}_{\sigma_{k+1}}$$

of the given symmetric execution for $k \in \{ 0, \ldots, m \}$ in the case of a finite symmetric execution and $k \in \mathbb{N}$ for an infinite symmetric execution. As constructed in Theorem 4.1.13 P_{k+1} is either the result of a step of $\sigma^i_k(P_k)$ realised without the rules Pi-ls-Com and Pi-ls-Close or it is the result of two communications of $\sigma^i_k(P_k)$ and $\sigma^j_k(P_k)$ realised by one of the rules Pi-ls-Com or Pi-ls-Close. We proceed with a case split.

Case without Pi-ls-Com **and** Pi-ls-Close **:** Let $\sigma^i_k(P_k)$ with $i \in \{ 0, \ldots, n-1 \}$ be the process which performs the first of the n steps labelled μ_{k+1}. We choose μ'_{k+1} as the $n-i$th action in $[\mu_{k+1}]^{n,\tilde{x}_k}_{\sigma_{k+1}}$, i.e., we choose the label of the action performed by process P_k. If μ_{k+1} is a bound output and μ'_{k+1} is not then we choose the bound output variant of μ'_{k+1}. By construction in the proof of Lemma 4.1.14 there are n' steps performed by the processes $\sigma^0_k(P_k)$, …, $\sigma^{n'-1}_k(P_k)$ and labelled by the first n' labels of $\left[\mu'_{k+1}\right]^{n,\tilde{x}_k}_{\sigma_{k+1}}$. Note that because σ'_k differs from σ_k only on permutations on formerly bound names we can perform these steps by ${\sigma'_k}^0(P_k)$, …, ${\sigma'_k}^{n'-1}(P_k)$, too. If μ_k is not a bound output we can choose $\tilde{x}'_{k+1} = \tilde{x}'_k$ and $\sigma'_{k+1} = \sigma'_k$ and are done. Else if $\mu_k = \overline{y}\,(z)$ and $z \notin \mathsf{bn}\big(\sigma^i_k(P_k)\big)$ we can choose $\sigma'_{k+1} = \sigma'_k$ and $\tilde{x}'_{k+1}$ as

the sequence of names in $\tilde{x}'_k, z_1, \ldots, z_l$, where $z_1, \ldots, z_l$ are the values of the bound outputs in $\left[\mu'_{k+1}\right]^{n',\tilde{x}_k}_{\sigma'_{k+1}}$. Else if $z \in \mathsf{bn}\big(\sigma^i_k(P_k)\big)$ we can choose $\tilde{x}'_{k+1} = \tilde{x}'_k$ and we add the permutations of z done by α-conversion as described in Lemma 4.1.14 to σ'_k to obtain σ'_{k+1}. Again by construction in Lemma 4.1.14 performing these n' steps we have $\left[P_k\right]^{n',\tilde{x}'_k}_{\sigma'_k} \xrightarrow{\left[\mu'_{k+1}\right]^{n',\tilde{x}'_k}_{\sigma'_{k+1}}} \left[P_{k+1}\right]^{n',\tilde{x}'_{k+1}}_{\sigma'_{k+1}}$.

Case with Pi-ls-Com **or** Pi-ls-Close **:** Then $[\mu_{k+1}]^{n,\tilde{x}_k}_{\sigma_{k+1}}$ is a sequence of n times τ. We choose $\mu'_{k+1} = \mu_{k+1} = \tau$ and $\left[\mu'_{k+1}\right]^{n',\tilde{x}_k}_{\sigma_k}$ is a sequence of n' times τ. Let $\sigma^i_k(P_k)$ and $\sigma^j_k(P_k)$ with $i, j \in \{\, 0, \ldots, n{-}1 \,\}$ be the processes which perform the first of the n steps. Without loss of generality let $\sigma^i_k(P_k)$ be the sender and $\sigma^j_k(P_k)$ be the receiver, i.e., $\sigma^i_k(P_k)$ performs an output action $\overline{\gamma}$ and $\sigma^j_k(P_k)$ performs the complementary receiving action γ. By construction in the proof of Lemma 4.1.14 the first n' steps within $\left[P_k\right]^{n,\tilde{x}_k}_{\sigma_k} \xrightarrow{[\mu_{k+1}]^{n,\tilde{x}_k}_{\sigma_{k+1}}} \left[P_{k+1}\right]^{n,\tilde{x}_{k+1}}_{\sigma_{k+1}}$ are performed by the senders $\sigma^i_k(P_k)$, …, $\sigma^{i+n'-1}_k(P_k)$ in this order sending $[\overline{\gamma}]^{n',\tilde{x}_k}_{\sigma_{k+1}}$ respectively and by the receivers $\sigma^j_k(P_k)$, …, $\sigma^{j+n'-1}_k(P_k)$ in this order receiving $[\gamma]^{n',\tilde{x}_k}_{\sigma_{k+1}}$. Now because of $\sigma^{n'} = \mathsf{id}$ and σ'_k differs from σ only by formerly bound names and their renamings according to α-conversion for each $g \in \{\, i, \ldots, i{+}n'{-}1 \,\}$ and for each $h \in \{\, j, \ldots, j{+}n'{-}1 \,\}$ we have $\sigma^g_k(P_k)$ and $\sigma'^{\,g \bmod n'}_k(P_k)$, and $\sigma^h_k(P_k)$ and $\sigma'^{\,h \bmod n'}_k(P_k)$ respectively are equal modulo the renaming performed formerly by α-conversion. With that we can again close the cycle as in the proof of Lemma 4.1.14 leading to $\left[P_k\right]^{n',\tilde{x}'_k}_{\sigma_k} \xrightarrow{\left[\mu'_{k+1}\right]^{n',\tilde{x}'_k}_{\sigma'_{k+1}}} \left[P_{k+1}\right]^{n',\tilde{x}'_{k+1}}_{\sigma'_{k+1}}$, where $\tilde{x}'_{k+1}$ and σ'_{k+1} are obtained from $\tilde{x}'_k$ and σ'_k as described in Lemma 4.1.14.

Because we can subdivide an arbitrary sequence of n steps we can subdivide each such sequence in the symmetric execution and with it the symmetric execution. □

Gorla [Gor10b] defines the reasonableness of an encoding by operational correspondence, divergence reflection, and success sensitiveness. We use just the last of his properties instantiated with must testing. So we implicitly require divergence reflection. According to [Gor10b], success is represented by a process ✓ that is part of the source and the target language of the encoding and always appears unbound. More precisely, a process must-succeeds if it reduces to a process containing a top-level unguarded occurrence of ✓ in every maximal execution. The fact that P must-succeeds is denoted by $P \Downarrow_{\checkmark}$. With it, an encoding is reasonable if the encoding of a term must-succeeds iff the term itself must-succeeds.

Definition 4.2.6 (2-Reasonableness)**.** An encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$ is *2-reasonable*, if $P \Downarrow_{\checkmark}$ iff $[\![P]\!] \Downarrow_{\checkmark}$ for all $P \in \mathcal{P}_{\mathrm{m}}$.

Again, we choose a term such that the encoding results in a network of the form $Q \mid Q$ in $\mathcal{P}_{\mathrm{s}}$ that is symmetric with respect to identity. In this case, we take advantage of the fact that the minimal degree of id is less than the degree of the network such that we can use Lemma 4.2.5 to subdivide the symmetric execution. With it already Q can perform the same sequence of steps as each process in $Q \mid Q$ performs in the symmetric execution.

Theorem 4.2.7 (Separation Result)**.** *There is no uniform and 2-reasonable encoding from π_{m} into π_{s}.*

Proof. Let us assume the contrary, i.e., there is a uniform and 2-reasonable encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P = a.0 + \overline{a}.\checkmark$$

Obviously, $\sigma = \mathsf{id}$ is a symmetry relation of degree 2 and so $N = \left[a.0 + \overline{a}.\checkmark\right]^2_\sigma$ is a symmetric network. Moreover, we have $N \Downarrow_\checkmark$ but $P \not\Downarrow_\checkmark$. We have $[\![P \mid P]\!] \overset{(U)}{=} [\![P]\!] \mid [\![P]\!] = \left[[\![P]\!] \right]^2_{\mathsf{id}}$, i.e., $[\![N]\!]$ is again a symmetric network of degree 2 with id as symmetry relation. By Theorem 4.1.13, $[\![N]\!]$ has at least one symmetric execution and by Definition 4.2.6 and must testing, $[\![N]\!]$ must reduce to a process containing a top-level unguarded occurrence of $\checkmark$ within this symmetric execution, i.e., there is a sequence of actions $\tilde{\mu} \in \mathcal{S}(\mathcal{A}_\tau)$, a process $P' \in \mathcal{P}_{\mathrm{s}}$, a $\sigma' \in \mathsf{Sym}(2, \mathcal{N})$ and a sequence of names $\tilde{x}$ such that $[\![P]\!] \mid [\![P]\!] \xrightarrow{\tilde{\mu}} \left[P'\right]^{2,\tilde{x}}_{\sigma'}$ and P' or $\sigma'(P')$ contain a top-level unguarded occurrence of $\checkmark$. Then, by symmetry, both processes of $\left[P'\right]^{2,\tilde{x}}_{\sigma'}$ contain a top-level unguarded occurrence of $\checkmark$. By Lemma 4.2.5, there is a sequence of actions $\tilde{\mu}' \in \mathcal{S}(\mathcal{A}_\tau)$ and an execution $[\![P]\!] \xrightarrow{\tilde{\mu}'} (\nu \tilde{x}')\, P'$ for a subsequence $\tilde{x}'$ of $\tilde{x}$. Since every maximal execution of $[\![P]\!]$ leads to a symmetric execution of $[\![P]\!] \mid [\![P]\!]$, we have $[\![P]\!] \Downarrow_\checkmark$, and with Definition 4.2.6 $P \Downarrow_\checkmark$, which is a contradiction. □

Note that, reconsidering the proofs of this separation result in [Gor10b], we managed to omit one of Gorla's additional assumptions. Namely, we do not need the assumption that $\asymp$ is exact (first setting) or reduction sensitive (second setting) and we do not need to assume the stronger version of operational correspondence in the third setting. On the other side Gorla does not need to assume the homomorphic translation of $\mid$ in his second and third setting. He uses the weaker notion of compositional translation of $\mid$ instead. But there is an encoding from π_{m} into $\pi^{=}_{\mathrm{a}}$ for this weaker structural assumption presented in Chapter 5 and proved good in Chapter 6. Summarising, this separation result is weaker than the result in the first setting of Gorla but incomparable to the results in the other two settings. In contrast to [Pal03] we can apply our absolute result to problem instances different from leader election, because we focus on breaking symmetries instead of leader election.

Also note that we obtain the same result if we replace Definition 4.2.6 by operational correspondence, divergence reflection, and success sensitiveness. Hence, this result holds in the general framework of Gorla presented in Section 3.3.

Lemma 4.2.8 (Separation Result)**.** *There is no good and uniform encoding from* π_{m} *into* π_{s}*.*

Proof. Let us assume the contrary, i.e., there is a good and uniform encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P = a.0 + \overline{a}.\checkmark$$

Obviously, $\sigma = \mathsf{id}$ is a symmetry relation of degree 2 and so $N = \left[a.0 + \overline{a}.\checkmark\right]^2_\sigma$ is a symmetric network. Moreover, we have $N \Downarrow_{\checkmark!}$ but $P \not\Downarrow_{\checkmark}$. We have $[\![P \mid P]\!] \overset{(U)}{=} [\![P]\!] \mid [\![P]\!] = \left[[\![P]\!] \right]^2_{\mathsf{id}}$, i.e., $[\![N]\!]$ is again a symmetric network of degree 2 with id as symmetry relation. Since N has no infinite execution and because of divergence reflection (Definition 3.3.5), $[\![N]\!]$ has no infinite execution. Moreover, by Lemma 3.3.10, $N \Downarrow_{\checkmark!}$ implies $[\![N]\!] \Downarrow_{\checkmark!}$. By Theorem 4.1.13, $[\![N]\!]$ has at least one symmetric execution. Then this symmetric execution is finite and success is reached, i.e., there is a process $P' \in \mathcal{P}_{\mathrm{s}}$, a $\sigma' \in \mathsf{Sym}(2, \mathcal{N})$ and a sequence of names $\tilde{x}$ such that $[\![P]\!] \mid [\![P]\!] \Longmapsto \left[P'\right]^{2,\tilde{x}}_{\sigma'}$ and P' or $\sigma'(P')$ contain a top-level unguarded occurrence of $\checkmark$. Then, by symmetry, both processes of $\left[P'\right]^{2,\tilde{x}}_{\sigma'}$ contain a top-level unguarded occurrence of $\checkmark$. By Lemma 4.2.5, there is an execution $[\![P]\!] \Longmapsto (\nu \tilde{x}')\, P'$ for a subsequence $\tilde{x}'$ of $\tilde{x}$. With it, $[\![P]\!] \Downarrow_{\checkmark}$, and with success sensitiveness $P \Downarrow_{\checkmark}$, which is a contradiction. □

Version 3. In the second setting of [Gor10b], Gorla considers prompt encodings (see Definition 3.2.1). Its separation results relies on the observation that there are terms $P \in \mathcal{P}_{\mathrm{m}}$ such that $P \not\longmapsto$, $P \not\Downarrow_{\checkmark}$ and $(P \mid P) \Downarrow_{\checkmark}$, but there are no such terms in $\mathcal{P}_{\mathrm{s}}$. Note that $P \not\Downarrow_{\checkmark}$ and $(P \mid P) \Downarrow_{\checkmark}$ implies $P \mid P \longmapsto$ and that there are no terms P in $\mathcal{P}_{\mathrm{s}}$ such that $P \not\longmapsto$ and $P \mid P \longmapsto$ which follows directly by Lemma 4.2.5. By using a slightly stronger variant of promptness, we do not need any notion of testing or preservation of behaviour to prove the separation result.

Definition 4.2.9 (3-Reasonableness)**.** An encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$ is *3-reasonable* if $P \longmapsto$ iff $[\![P]\!] \longmapsto$ for all $P \in \mathcal{P}_{\mathrm{m}}$.

To our knowledge, only few intuitively reasonable encodings are not also 3-reasonable. Note, however, that the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ from π_{m} into $\pi^{=}_{\mathrm{a}}$ introduced in Chapter 5 is neither prompt nor 3-reasonable.

Theorem 4.2.10 (Separation Result)**.** *There is no uniform and 3-reasonable encoding from* π_{m} *into* π_{s}*.*

Again, for the separation proof, we enforce that the encoding results in a symmetric network $Q \mid Q$. By subdividing the symmetric execution of this network, we prove that $Q \longmapsto$ iff $Q \mid Q \longmapsto$, which does not necessarily hold in π_{m}.

Proof. Let us assume the contrary, i.e., there is a uniform and 3-reasonable encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$. Consider the network:

$$N \triangleq P \mid P \quad \text{with} \quad P \triangleq a + \overline{a}$$

Obviously, $\sigma = \mathsf{id}$ is a symmetry relation of degree 2 and so $N = \left[a + \overline{a}\right]^2_\sigma$ is a symmetric network. Moreover, we have $N \longmapsto$ but $P \not\longmapsto$. We have $[\![P \mid P]\!] \overset{(U)}{=} [\![P]\!] \mid [\![P]\!] = \left[[\![P]\!] \right]^2_{\mathsf{id}}$, i.e., $[\![N]\!]$ is again a symmetric network of degree 2 with id as symmetry relation. By Theorem 4.1.13, $[\![N]\!]$ has at least one symmetric execution and by 3-reasonableness we have $[\![P]\!] \mid [\![P]\!] \longmapsto$ and $[\![P]\!] \not\longmapsto$ and thus $[\![P]\!] \mid [\![P]\!] \xrightarrow{\tau}$ and $[\![P]\!] \not\xrightarrow{\overline{\tau}}$. By Lemma 4.1.14, $[\![P]\!] \mid [\![P]\!] \xrightarrow{\tau}$ implies that there is at least one step in the symmetric execution, i.e., there is a process $P' \in \mathcal{P}_{\mathrm{s}}$, a symmetry relation $\sigma' \in \mathsf{Sym}(2, \mathcal{N})$, and a sequence of names $\tilde{x} \in \mathcal{S}(\mathcal{N})$ such that $[\![P]\!] \mid [\![P]\!] \xrightarrow{\tau,\tau} \left[P'\right]^{2,\tilde{x}}_{\sigma'}$. Then, by Lemma 4.2.5, there is an execution $[\![P]\!] \xrightarrow{\tau} (\nu \tilde{x}')\, P'$ for a subsequence $\tilde{x}'$ of $\tilde{x}$, i.e., $[\![P]\!] \longmapsto$, which is a contradiction. □

Noteworthy, we do not even assume divergence reflection in this argumentation. Moreover, the counterexample consists of two very simple mixed choices combined in parallel. This shows the relevance of mixed choice for the non-existence of uniform encodings.

Summary. Our three translational separation results, i.e., the proofs of the non-existence of a uniform and reasonable encoding for different definitions of reasonableness, follow similar lines of argument. The proofs argue by contradiction. First, a symmetric network of the form $P \mid P$ in $\mathcal{P}_{\mathrm{m}}$ with special features—that are connected to our absolute result—is presented. Second, we use the fact that uniformity, i.e., the homomorphic translation of the parallel operator, preserves essential parts of the symmetric nature of $P \mid P$. Third, we apply Theorem 4.1.13 to conclude with the existence of a symmetric execution. In two proofs we then apply Lemma 4.2.5 to subdivide this symmetric execution. At last we derive a contradiction between the additional information provided by the symmetric execution (and its subdivision) and the respective definition of reasonableness.

Note that we prove the absolute result without any precondition and that we use different definitions of reasonableness for the translational results. The only constant precondition of the separation results is the homomorphic translation of the parallel operator. This condition is crucial. Without it, we could not apply our absolute separation result. To the best of our knowledge, only Gorla (second and third setting of [Gor10b]) and [FL10] (with respect to a stricter formulation of operational correspondence) ever managed to prove such a separation result between π_{m} and π_{s} without the homomorphic translation of the parallel operator. There is another separation result in [CCP07] via must-testing; but they require $[\![P \mid o]\!] = [\![P]\!] \mid [\![o]\!]$ for all processes P and observers o. Note that we present two more separation results with respect to a new criterion, namely the preservation of distributability, in Section 4.2.2 and Section 4.4. However, by the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ from π_{m} into $\pi^{=}_{\mathrm{a}}$ in Chapter 5 and its validation in Chapter 6, we show that separation does not hold for the general framework as presented in Section 3.3. On the other side, we discuss some drawbacks of our encoding in the next two chapters that, to our knowledge, cannot be avoided. Thus, we believe that there is no good encoding from π_{m} into π_{s}, if either we do not allow for matching in the target language (as it is allowed in [Gor10b]), or if we forbid for observable junk (see Section 6.3.5), e.g. by restricting $\asymp$ to be exact.

We may also turn the non-existence of a uniform and reasonable encoding around and rephrase it as a weakened existence statement. Recall that any uniform encoding from π_{m} into π_{s} preserves symmetries. While it is possible to break such symmetries in π_{m}, this is not possible in π_{s}. Note that Theorem 4.1.13 talks only about the existence of a symmetric execution, but does not tell anything about their maximal length. Hence, it is possible to "hide" in an encoding the symmetric execution of target terms within divergent executions. Of course, in divergence reflecting encodings this is not allowed. Thus, should there be a *non-uniform* but divergence reflecting and reasonable encoding from π_{m} into π_{s}, then it would have to be the encoding itself to break symmetries. We will use this insight as starting point for our encoding in Chapter 5.

4.2.2. Different Domains

In Section 3.4.3 we explain why the homomorphic translation of the parallel operator is too strict for translational separation results with respect to distributability. Instead we formalise a new criterion, namely preservation of distributability, for this purpose and combine it with the criteria of the general framework in Section 3.3. For the existence of a good encoding $[\![\cdot]\!]$ from π_{m} into π_{s}, preservation of distributability means that for all source terms $S \in \mathcal{P}_{\mathrm{m}}$, if $S \equiv (\nu \tilde{x})\,(S_1 \mid \ldots \mid S_n)$ then $[\![S]\!] \equiv (\nu \tilde{x}')\,(T_1 \mid \ldots \mid T_n)$ such that $T_i \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$. Unfortunately, the requirement $T_i \asymp [\![S_i]\!]$ is not strong enough to ensure that $(\nu \tilde{x}')\,(T_1 \mid \ldots \mid T_n)$ is a symmetric network if $(\nu \tilde{x})\,(S_1 \mid \ldots \mid S_n)$ is a symmetric network. Thus, in contrast to the homomorphic translation of the parallel operator, the preservation of distributability does not preserve source term symmetries. That is a problem for our absolute result in Theorem 4.1.13. But it turns out that, although we need another argument to complete a separation result, we can nonetheless use our absolute result to reason about the encoding function, which provides further insights also with respect to the positive result in Chapter 5.

Therefore, we analyse the context introduced by compositionality to encode the parallel operator and examine how this context has to interact with the encodings of its parameters to allow for an emulation of a source term step. We start with some observations concerning the three process terms

$$P \triangleq \overline{a} + a \mid \overline{b} + b.\checkmark, \quad Q \triangleq P \mid P, \quad \text{and} \quad R \triangleq \overline{a} + b + b.\checkmark \mid \overline{b} + a + a.\checkmark,$$

which are used in the following lemmata as counterexamples. Note that we choose Q and R such that each of them is a symmetric network of degree 2 with either $\sigma = \{\, {}^{a}\!/_{b}, {}^{b}\!/_{a} \,\}$ or id as symmetry relation. Moreover, to fix the context used to encode the parallel operator we choose P, Q, and R such that $\mathsf{fn}(P) = \mathsf{fn}(Q) = \mathsf{fn}(R) = \{\, a, b \,\}$. Hence by compositionality for each of these three terms the outermost parallel operator is translated by exactly the same context $\mathcal{C}_{|}^{\{\, a,b \,\}}([\cdot]_1, [\cdot]_2)$. Note that for all of the following lemmata and observations we silently assume that $[\![\cdot]\!] : \mathcal{P}_{\mathrm{m}} \to \mathcal{P}_{\mathrm{s}}$ is a good encoding from π_{m} into π_{s}, i.e., $[\![\cdot]\!]$ satisfies the criteria introduced in Section 3.3.

Observation 4.2.11. There exists a context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2) : \mathcal{P}_\mathrm{s} \times \mathcal{P}_\mathrm{s} \to \mathcal{P}_\mathrm{s}$ such that

$$\begin{aligned}
[\![\, P \,]\!] &= \mathcal{C}_{|}^{\{\,a,b\,\}}([\![\, \overline{a} + a \,]\!], [\![\, \overline{b} + b.\checkmark \,]\!])\,, \\
[\![\, Q \,]\!] &= \mathcal{C}_{|}^{\{\,a,b\,\}}([\![\, P \,]\!], [\![\, P \,]\!])\,, \text{ and} \\
[\![\, R \,]\!] &= \mathcal{C}_{|}^{\{\,a,b\,\}}([\![\, \overline{a} + b + b.\checkmark \,]\!], [\![\, \overline{b} + a + a.\checkmark \,]\!])\,.
\end{aligned}$$

We choose P such that none of its executions lead to success, i.e., $P \mapsto\!\!\!\!\!\!/\;\;$ and $P \Downarrow\!\!\!\!\!\!/_{\checkmark}$. Because $[\![\, \cdot \,]\!]$ satisfies the criteria of Section 3.3, this implies that also the encoding of P does not reach success.

Lemma 4.2.12. *For all $T_P \in \mathcal{P}_\mathrm{s}$, $[\![\, P \,]\!] \Longmapsto T_P$ implies $T_P \Downarrow\!\!\!\!\!\!/_{\checkmark}$.*

Proof. $P \mapsto\!\!\!\!\!\!/\;\;$ implies, by operational soundness, that $[\![\, P \,]\!]$ cannot perform a step that changes its state modulo $\asymp$, i.e., $[\![\, P \,]\!] \Longmapsto T_P$ implies $T_P \Longmapsto\!\!\!\!\!\!/\;\;\asymp [\![\, P \,]\!]$ for all $T_P \in \mathcal{P}_\mathrm{s}$. By success sensitiveness, $P \Downarrow\!\!\!\!\!\!/_{\checkmark}$ implies $[\![\, P \,]\!] \Downarrow\!\!\!\!\!\!/_{\checkmark}$. Because of that and since $\asymp$ is success respecting we have $T_P \Downarrow\!\!\!\!\!\!/_{\checkmark}$ for all $T_P \in \mathcal{P}_\mathrm{s}$ such that $[\![\, P \,]\!] \Longmapsto T_P$. □

Hence, any occurrence of ✓—if there is any—in the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ is guarded and the context cannot remove such a guard on its own. In contrast to P, we choose Q such that Q reaches an unguarded occurrence of success in any of its maximal executions. Again this has to be reflected by the encoding function.

Lemma 4.2.13. *For all $T_Q \in \mathcal{P}_\mathrm{s}$, $[\![\, Q \,]\!] \Longmapsto T_Q$ implies $T_Q \Downarrow_{\checkmark}$.*

Proof. By operational completeness, any execution $Q \longmapsto Q_1 \longmapsto Q_2 \mapsto\!\!\!\!\!\!/\;\;$ of Q can be emulated by its encoding, i.e., $[\![\, Q \,]\!] \Longmapsto Q'_1$, $[\![\, Q \,]\!] \Longmapsto Q'_2$, and $[\![\, Q_1 \,]\!] \Longmapsto Q''_2$, where $Q'_i \asymp [\![\, Q_i \,]\!]$ for $i \in \{\, 1,2 \,\}$ and $Q''_2 \asymp [\![\, Q_2 \,]\!]$. Note that any maximal execution of Q is such that $Q \longmapsto Q_1 \longmapsto Q_2 \mapsto\!\!\!\!\!\!/\;\;$ for some $Q_1, Q_2 \in \mathcal{P}_\mathrm{m}$. By operational soundness for each $T_Q \in \mathcal{P}_\mathrm{s}$ such that $[\![\, Q \,]\!] \Longmapsto T_Q$, there is some $Q' \in \mathcal{P}_\mathrm{m}$ such that $Q \Longmapsto Q'$ and $T_Q \Longmapsto\!\!\!\!\!\!/\;\;\asymp [\![\, Q' \,]\!]$, i.e., there is some $T'_Q \in \mathcal{P}_\mathrm{a}$ such that $T_Q \Longmapsto T'_Q$ and $T'_Q \asymp [\![\, Q' \,]\!]$. By success sensitiveness, $[\![\, Q' \,]\!] \Downarrow_{\checkmark}$ and since $\asymp$ is success respecting, we have $T'_Q \Downarrow_{\checkmark}$. Thus, by Definition 3.2.2, we have $T_Q \Downarrow_{\checkmark}$ for all $T_Q \in \mathcal{P}_\mathrm{s}$ with $[\![\, Q \,]\!] \Longmapsto T_Q$. □

At last we choose R such that some of its executions lead to success while some do not.

Lemma 4.2.14. *There are $T_{R,1}, T_{R,2} \in \mathcal{P}_\mathrm{s}$ with $[\![\, R \,]\!] \Longmapsto T_{R,1} \wedge [\![\, R \,]\!] \Longmapsto T_{R,2} \wedge T_{R,1} \Downarrow_{\checkmark} \wedge T_{R,2} \Downarrow\!\!\!\!\!\!/_{\checkmark}$.*

Proof. R can reduce either to ✓ or to $\mathsf{0}$. By operational completeness, $[\![\, R \,]\!]$ can emulate both steps, i.e., $[\![\, R \,]\!] \Longmapsto\!\!\!\!\!\!/\;\;\asymp [\![\, \checkmark \,]\!]$ and $[\![\, R \,]\!] \Longmapsto\!\!\!\!\!\!/\;\;\asymp [\![\, \mathsf{0} \,]\!]$. By success sensitiveness, we have $[\![\, \checkmark \,]\!] \Downarrow_{\checkmark}$ and $[\![\, \mathsf{0} \,]\!] \Downarrow\!\!\!\!\!\!/_{\checkmark}$. Then, since $\asymp$ is success respecting, we have $[\![\, \checkmark \,]\!] \not\asymp [\![\, \mathsf{0} \,]\!]$. Thus, there are at least two different target terms $T_{R,1}, T_{R,2} \in \mathcal{P}_\mathrm{s}$ such that $[\![\, R \,]\!] \Longmapsto T_{R,1}$, $[\![\, R \,]\!] \Longmapsto T_{R,2}$, $T_{R,1} \Downarrow_{\checkmark}$, and $T_{R,2} \Downarrow\!\!\!\!\!\!/_{\checkmark}$. □

Our last observation concerns the structure of the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$. The context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ has to place its parameters in parallel, because this is the only binary operator for processes different from (separate) choice. Placing them within a choice would not allow to use the encodings of both parameters to emulate target term steps. Consequently, there must be some $\mathcal{P}_{\mathrm{s}}$-contexts $\mathcal{C}_0([\cdot])$, $\mathcal{C}_1([\cdot])$, $\mathcal{C}_2([\cdot])$ with $[\![\, S_1 \mid S_2 \,]\!] \equiv \mathcal{C}_{|}^{\{\,a,b\,\}}([\![\, S_1 \,]\!],[\![\, S_2 \,]\!]) \equiv \mathcal{C}_0(\mathcal{C}_1([\![\, S_1 \,]\!]) \mid \mathcal{C}_2([\![\, S_2 \,]\!]))$, for all source terms $S_1, S_2 \in \mathcal{P}_{\mathrm{m}}$ with $\mathsf{fn}(S_1 \mid S_2) = \{\, a, b \,\}$.

Observation 4.2.15. There are some contexts $\mathcal{C}_0([\cdot]),\mathcal{C}_1([\cdot]),\mathcal{C}_2([\cdot]) : \mathcal{P}_{\mathrm{s}} \to \mathcal{P}_{\mathrm{s}}$ such that $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2) \equiv \mathcal{C}_0(\mathcal{C}_1([\cdot]_1) \mid \mathcal{C}_2([\cdot]_2))$.

Learning from the separation results in Section 4.2.1, we know that any good encoding from π_{m} into π_{s} must break source term symmetries. To do so, we show that the context introduced by the encoding of the parallel operator (which is allowed in weak compositionality as opposed to homomorphic translations) must interact with the encodings of its parameters.

Lemma 4.2.16. *To emulate a source term step, $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ and the encodings of its parameters have to interact.*

Intuitively we show, that if there is no such interaction, then since Q is a symmetric network its encoding also behaves as a symmetric network. Since any execution of $[\![\, Q \,]\!]$ leads to an unguarded occurrence of success, by symmetry and by Lemma 4.2.5, there is an execution of $[\![\, P \,]\!]$ leading to an unguarded occurrence of success, which contradicts Lemma 4.2.12.

Proof. Assume the contrary, i.e., assume the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ is such that possibly after some preprocessing steps of the context on its own, e.g. to unguard the parameters, the source term steps can be emulated without any interaction with the context. In this case, we have

$$\begin{aligned}[\![\, Q \,]\!] &\overset{4.2.11}{=} \mathcal{C}_{|}^{\{\,a,b\,\}}([\![\, P \,]\!],[\![\, P \,]\!]) \overset{4.2.15}{\equiv} \mathcal{C}_0(\mathcal{C}_1([\![\, P \,]\!]) \mid \mathcal{C}_2([\![\, P \,]\!])) \\ &\longmapsto (\nu\tilde{y})\,(\sigma_1([\![\, P \,]\!]) \mid \sigma_2([\![\, P \,]\!]) \mid T_C)\end{aligned}$$

for some constant term T_C, a sequence of names $\tilde{y}$, and two substitutions σ_1 and σ_2. Note that σ_1 and σ_2 capture renaming, due to α-conversion that is possibly necessary to move restrictions outwards. Since there is no need for an interaction, i.e., for a communication, with T_C to emulate source term steps, we can ignore it.

If $\sigma_1 = \sigma_2$ then, since these substitutions result from α-conversion, $\sigma_1 = \sigma_2 = \mathsf{id}$. Then $[\![\, P \,]\!] \mid [\![\, P \,]\!]$ is a symmetric network of degree 2 with id as symmetry relation. By Theorem 4.1.13, $[\![\, P \,]\!] \mid [\![\, P \,]\!]$ has a symmetric execution. By Lemma 4.2.13, $[\![\, Q \,]\!]$ reaches an unguarded occurrence of success in any of its executions. Since the context, and with it T_C, cannot reach success on its own and there is no interaction, $[\![\, P \,]\!] \mid [\![\, P \,]\!]$ reaches success in its symmetric execution. Then there is some $T''_Q \in \mathcal{P}_{\mathrm{s}}$ such that

$[\![P]\!] \mid [\![P]\!] \Longmapsto (\nu\tilde{x}) \left(T''_Q \mid \sigma_3\left(T''_Q\right)\right)$ is a symmetric execution for some sequence of names $\tilde{x}$ and some symmetry relation σ_3 of degree 2 and $(\nu\tilde{x}) \left(T''_Q \mid \sigma_3\left(T''_Q\right)\right)$ has an unguarded occurrence of success. By symmetry and since $\mathsf{n}(\checkmark) = \emptyset$, this implies that T''_Q as well as $\sigma_3\left(T''_Q\right)$ has an unguarded occurrence of success. Since 2 is not the minimal degree of the identity, by Lemma 4.2.5, this symmetric execution can be subdivided such that $[\![P]\!] \Longmapsto (\nu\tilde{x}')\, T''_Q$ for some sequence of names $\tilde{x}'$. Then $[\![P]\!] \Downarrow_{\checkmark}$, because of the unguarded occurrence of $\checkmark$ in T''_Q. This contradicts Lemma 4.2.12.

The argumentation for $\sigma_1 \neq \sigma_2$ is similar, but more difficult. In this case $\sigma_1([\![P]\!]) \mid \sigma_2([\![P]\!])$ is still a symmetric network whose symmetric execution leads to an unguarded occurrence of success. But since its symmetry relation is not id we cannot apply Lemma 4.2.5. However, because σ_1 and σ_2 result from α-conversion, they rename free names of $[\![P]\!]$ to fresh names. If $\sigma_1([\![P]\!])$ and $\sigma_2([\![P]\!])$ want to interact on such a fresh name, then they have first to exchange this fresh name over a channel known to both. Let us denote this channel by z. So either $\sigma_1([\![P]\!])$ receives a fresh name from $\sigma_2([\![P]\!])$ over z or vice versa. By symmetry, both terms have an unguarded input as well as an unguarded output on z, so—instead of a communication between these two processes—$\sigma_1([\![P]\!])$ can as well reduce on its own. Adding this observation to the argumentation in the proof of Lemma 4.2.5 we can prove again that the symmetric execution of $\sigma_1([\![P]\!]) \mid \sigma_2([\![P]\!])$ can be subdivided such that $\sigma_1([\![P]\!]) \Downarrow_{\checkmark}$. Because $\mathsf{n}(\checkmark) = \emptyset$, this implies $[\![P]\!] \Downarrow_{\checkmark}$. Again, this contradicts Lemma 4.2.12.

Hence, to emulate a source term step, the context necessarily has to interact with its parameters. □

As induced by this lemma, the encoding function $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ from π_{m} into $\pi^{=}_{\mathrm{a}}$ introduced in Chapter 5 implements a protocol within the context allowed by compositionality that controls the interactions between the encoded parameters of a parallel operator.

Note that the only possibility for the context to interact with its parameters is by communication. So the context contains at least one capability, i.e., input or output prefix, that needs to be consumed to emulate a source term communication between the subterms of a parallel composition. Without loss of generality, let us assume that indeed only a single capability needs to be consumed to emulate a step, i.e., a single communication step of the context with (one of) its parameters suffices to enable the emulation of a source term communication between the parameters of a parallel operator. The argumentation for a couple of necessary communication steps is similar. Let us denote this capability by μ.[7]

Next we show that either it is not possible to emulate two different source term steps between the parameters of $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1, [\cdot]_2)$ at the same time or $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1, [\cdot]_2)$ has to sequentialise parts of different emulations.

[7] In the case of a sequence of necessary steps, choose μ such that it denotes the capability that is the last to be consumed in this sequence. In the case where there are different possibilities to enable the emulation of a step, consider a set of those capabilities with one μ_i for each such possibility.

Lemma 4.2.17. *The context $\mathcal{C}_{|}^{\{\ a,b\ \}}([\cdot]_1,[\cdot]_2)$ either has to restrict the number of source term steps that can be emulated simultaneously, or it has to sequentialise two steps of different emulations.*

Here we use R as a counterexample. R can reduce either to $\mathsf{0}$ or $\checkmark$, so the choice operator introduces mutual exclusion. Without choice mutual exclusion is hard to implement, because of its ability to immediately block an alternative reduction. We show that the emulation of the respective blocking behaviour introduces either deadlock or divergence.

Proof. Assume the contrary, i.e., assume that the context $\mathcal{C}_{|}^{\{\ a,b\ \}}([\cdot]_1,[\cdot]_2)$ does enable the emulation of different alternative source term steps concurrently, i.e., provides e.g. by replication several instances of μ, and does not sequentialise emulations. Consider the source term R. Since $\{\ {}^{a}\!/_{b},{}^{b}\!/_{a}\ \}(\overline{a}+b+b.\checkmark)=\overline{b}+a+a.\checkmark$, by name invariance, there is some substitution σ' such that $\sigma'(\llbracket\ \overline{a}+b+b.\checkmark\ \rrbracket)\equiv_\alpha \llbracket\ \overline{b}+a+a.\checkmark\ \rrbracket$, i.e., these two terms are equal except to some renamings of free names.

Moreover, since both sums have the same free names, compositionality requires, that we translate both by the same context. Without mixed choice, we either have to split each of these sums into an input and an output guarded sum, or we have to convert each of them into a single sum with only separate choice. In the second case, by compositionality, both parameters have to be encoded in exactly the same way; then, we either result in two input-guarded or two output-guarded sums. But then we cannot emulate a communication between these two sums within a single step and, moreover, we cannot decide within a single step whether a considered capability can be used to successfully emulate a source term step. Unfortunately, the first step used to check whether we can use this capability to emulate a source term step removes all the other encoded capabilities of that sum, which violates operational correspondence. Therefore, at least the output and input parts of the sums have to be placed somehow in parallel.

Note that R can perform either a step on channel a or b. Because we place the encodings of the output and input capabilities of the sums $\overline{a}+b+b.\checkmark$ and $\overline{b}+a+a.\checkmark$ in parallel, the emulation of the source term step on a does not immediately withdraw the encodings of the capabilities on b and vice versa. Thus, since the emulation of both steps of R are enabled concurrently, there is some moment in the emulation of one source term step that disables the completion of the emulation of the respective other source term step. Therefore, one emulation has to consume some capability that is necessary to complete the other emulation. Remember that we assume that the only capability of the context $\mathcal{C}_{|}^{\{\ a,b\ \}}([\cdot]_1,[\cdot]_2)$ necessary to be consumed to emulate a source term step is μ. Hence, to allow the emulation of one step of R, to disable the emulation of the respective other step of R, and since $\sigma'(\llbracket\ \overline{a}+b+b.\checkmark\ \rrbracket)\equiv_\alpha \llbracket\ \overline{b}+a+a.\checkmark\ \rrbracket$, there is some capability in $\llbracket\ \overline{a}+b+b.\checkmark\ \rrbracket$ as well as in $\llbracket\ \overline{b}+a+a.\checkmark\ \rrbracket$ and, to emulate a source term step, both of these capabilities have to be consumed. Moreover, since $\sigma'(\llbracket\ \overline{a}+b+b.\checkmark\ \rrbracket)\equiv_\alpha \llbracket\ \overline{b}+a+a.\checkmark\ \rrbracket$, both capabilities are of the same kind, i.e., both are either input prefixes or both are output prefixes. Note that there is no possibility in π_{s} to consume two capabilities of the same kind within the same target term step except

they are parts of the same sum, which is not the case here, because one comes from the left and the other from the right sum and by Observation 4.2.15 the left and the right side of a parallel operator are composed in parallel. Then either both emulations agree on which capability they have to consume first (Case 2) or it cannot be avoided that for each of the emulations of the two steps exactly one of these two capabilities is consumed (Case 1).

In Case 1, since both capabilities are already consumed, none of the emulations can be completed, i.e., there is some local deadlock. Note that the possibility of such a local deadlock violates the combination of success sensitiveness and operational correspondence. Considering for example $[\![Q]\!]$, such a deadlock leads to a term T_Q with $[\![Q]\!] \Longrightarrow T_Q$ and, since none of the source term steps is emulated, no unguarded occurrence of $\checkmark$ is reached, i.e., $T_Q \not\Downarrow_{\checkmark}$. This contradicts Lemma 4.2.13.

The only way to circumvent this deadlock is that one of these capabilities is released by one of these emulations. To complete the emulation of this step later on, it has to be possible that the released capability is consumed again. But then it cannot be avoided that this is done before the other emulation is finished, i.e., that leads back to the situation before. Then we introduce divergence, which contradicts divergence reflection. Thus, in this case, it is not possible that the emulation of alternative source term steps is enabled concurrently, i.e., the context $\mathcal{C}_{|}^{\{ a,b \}}([\cdot]_1, [\cdot]_2)$ has to introduce some mechanism (e.g. a lock) to ensure, that no two source term steps are emulated at the same time over $\mathcal{C}_{|}^{\{ a,b \}}([\cdot]_1, [\cdot]_2)$. Since μ is the only capability necessary to emulate a source term step, this implies that there cannot be more than one instance of μ.

Let us consider the Case 2. Since the encoding is compositional we cannot simply assume a total ordering of these capabilities before we start the encoding function, because this would require global knowledge of the source term, which is not available in a compositional encoding. Thus, the encoding function must decide at run time which of the capabilities it consumes first and this decision has to be made consistently for different emulation attempts. We observe furthermore that the algorithm to implement this decision has to be placed into the encoding of a parallel operator, because that is the only operator that surely surrounds the encodings of two communication partners and with it the respective capabilities we have to order.

In order to emulate a source term step, the encodings of the respective communication partners send a request to the surrounding parallel operator encoding, which decides which of the capabilities has to be consumed first. Note that we do not restrict the number of emulations that can be performed simultaneously, because else we result in Case 1 again. So there may be several requests originating from different pairs of communication partners that arrive at the same parallel operator encoding. Note there may even be several left and several right requests induced by communication attempts on the same source term channel. So it is sometimes necessary to combine a left request with several right requests or the other way around. However, the parallel operator encoding cannot combine the same left and the same right request infinitely often, because this would introduce divergence. So, it must keep track of the pairs of requests it has already combined. But to do so it cannot process these pairs in parallel but only in sequence.

The algorithm cannot combine all possible pairs at the same time, because that does not allow to keep track of already checked pairs. Note that the sequential processing of these pairs does not allow that the decision of the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ on which of the respective capabilities has to be consumed first is made truly in parallel for concurrent emulations on the same parallel operator. Thus, the $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ sequentialise two steps of different emulations in this case. If these emulations refer to distributable source term steps this sequentialisation reduces the degree of distributability. □

Obviously both—the temporary blocking of emulations as well as the sequentialisation of some steps of different emulations —violates the preservation of distributability.

Theorem 4.2.18 (Separation Result)**.** *There exists no good encoding from* π_{m} *into* π_{s} *that preserves distributability.*

Proof. By Lemma 4.2.17, for all source term steps that are communications between the parameters of a parallel operator, either the number of such communications that can be emulated simultaneously has to be reduced or steps of the emulations of different such source term steps have to be processed in sequence. Q can perform two parallel steps; one reducing a and the other reducing b. For both step an interaction of the left and the right side of the outermost parallel operator is necessary. By Lemma 4.2.17 either one emulation of these two steps is temporary blocked during the emulation of the other, or two steps of both emulations are sequentialised, i.e., are not parallel and thus not distributable. In both cases, the emulations of the distributable steps in Q are not distributable in $[\![\, Q \,]\!]$. Hence, by Lemma 3.4.8, the requirement of the preservation of distributability is violated. □

Moreover, Lemma 4.2.17 has direct consequences for another research area in the context of distributed systems, namely for the ability to preserve the causal semantics of source terms. Causal relations describe a special form of dependencies. For the pi-calculus usually two kinds of causal dependencies are distinguished (see [Pri96, BS98]). The first one, called structural or subject dependencies, originates from the nesting of prefixes, i.e., from the structure of processes. A typical example of such a dependency is given by $(\nu b)\left(\overline{a}.\overline{b} \mid b.\overline{c}\right) \mid a \mid c \longmapsto (\nu b)\left(\overline{b} \mid b.\overline{c}\right) \mid c \longmapsto \overline{c} \mid c \longmapsto \mathsf{0}$. The second step on channel b is causally dependent on the first step, because it unguards $\overline{b}$. So b is causally dependent on a. Similarly, c is causally dependent on b, and by transitivity c is causally dependent on a. The other kind of dependencies are called link or object dependencies and originate from the binding mechanisms on names. Here a typical example is $(\nu x)\,(\overline{y}\langle x\rangle \mid \overline{x})$. In a labelled semantics the output on x is causally dependent on the extrusion of x by an output on y, i.e., x is causally dependent on y.

Note that the areas of causality and distributability are connected, at least in one direction. Two distributable steps are by definition independent to a degree that implies also causal independence. In fact causality is often defined as the opposite of concurrency, i.e., two actions are concurrent only if they are not causally dependent [CMT96, Pri96, BS98]. Hence, if an encoding introduces additional causal dependencies between the emulations of distributable steps, as induced by Lemma 4.2.17, then this directly violates

the preservation of distributability. Apart from that, the analysis of causal relations is interesting in itself.

We observe that causal dependencies are defined as a condition between actions or names of actions. In the context of encodings this view is problematic, because steps are often translated into pomsets of steps and names may be translated into sequences of names. Moreover a pomset of steps simulating a single source term step may be interleaved with another such pomset or some target term steps used to prepare the simulation of another source term step, whose simulation may never be completed. So, what precisely does it mean for an encoding to preserve or respect causal dependencies? If source term names are translated into sequences of names, should one consider the causal dependencies between all such translated names or only between some of them? Moreover, how should an encoding handle names reserved for some special purposes of the encoding function, i.e., target term names that do not result from the translation of a source term name?

We have no final answer to these questions yet. However, the following separation result does not require a thorough answer to the questions above. Instead we use a definition of causal dependencies that is based only on direct subject dependencies. So within this paper, a step b is considered causally dependent on a previous step a, if b depends on the availability of a capability produced by a. More precisely, step b is causally dependent on step a, if a unguards some capability, i.e., some input or output prefix, which is consumed by step b. An encoding preserves causal dependencies, if for any causal dependency between two steps of the source term there is a causal dependency between some steps of their emulations, and an encoding reflects causal dependencies, if for any causal dependency between two steps of different (completed) emulations there is a causal dependency of the corresponding source term steps.

Theorem 4.2.19 (Separation Result)**.** *There exists no good encoding from* π_{m} *into* π_{s} *that reflects causal dependencies.*

Proof. By Lemma 4.2.17 we have to distinguish two cases.

In order to reduce the number of emulations of source term communications the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ provides only a single instance of μ. Let us consider Q once more. By operational completeness, $[\![\, Q \,]\!] \Longmapsto Q_1'$, $[\![\, Q \,]\!] \Longmapsto Q_2'$, and $[\![\, Q_1 \,]\!] \Longmapsto Q_2''$, where $Q_i' \asymp [\![\, Q_i \,]\!]$ for $i \in \{\, 1,2 \,\}$ and $Q_2'' \asymp [\![\, Q_2 \,]\!]$, i.e., $[\![\, Q \,]\!]$ emulates two subsequent source term steps. Note that for both step of Q an interaction of the left and the right side of the outermost parallel operator is necessary, i.e., both emulations have to be enabled by the context $\mathcal{C}_{|}^{\{\,a,b\,\}}([\cdot]_1,[\cdot]_2)$ by an instance of μ. Hence, the instance of μ consumed by the emulation of the first source term step has to be restored during this first emulation such that the second step can be emulated. Thus, the emulation of the second step has to consume some capability μ produced by the emulation of the first step. Then the emulation of the second step causally depends on the emulation of the first step.

In the second case $[\![\,\cdot\,]\!]$ sequentialises two steps of different emulations. To forbid that these two steps are performed in parallel the encoding has to introduce a causal dependency between them. Again this leads to a causal dependency of the respective emulations.

Because any pair of subsequent steps of Q is causally independent, we conclude that in each case the encoding function adds additional causal dependencies, i.e., does not reflect the causal dependencies of the source term. □

4.3. Transferring Absolute Results

If neither the analysis of single operators nor standard problems lead to suitable absolute results, a third method is to reuse the main idea of separation results in other settings. Another attempt to compare the expressive power of synchronous and asynchronous interaction mechanisms can be found in [vGGS08]. It analyses the possibility to implement a (synchronous) Petri net specification within an asynchronous setting by an automatic algorithm. They find a semi-structural property called M that distinguishes distributable Petri nets from those nets that may be implemented only under additional assumptions on the underlying system structure in a fully asynchronous and distributed setting.

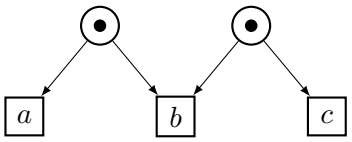


Figure 4.3.: A fully reachable pure M in Petri nets.

An M, as visualised in Figure 4.3, describes a part of a Petri net that consist of two parallel transitions and one transitions that is in conflict with both of the former. In other words it describes a situation where either two parts of the net can proceed independently or they synchronise each other in order to perform a single transition together. Thus, an M describes a special situation of synchronisation. Hence, we denote an M as synchronisation pattern. [vGGS08] states that a Petri net specification can be implemented in an asynchronous, fully distributed setting if it does not contain a fully reachable pure M. Accordingly, they denote such Petri nets as distributable. They also present a description of a fully reachable pure M as a property of a step transition system which allows us to directly use this pattern to reason about process calculi.

A first analysis shows that we find the M also in the asynchronous pi-calculus (see Example 4.3.2 below). This reflects earlier observations in [Lév97, Fou98]: it is not possible to implement the pi-calculus and even its asynchronous fragment in an asynchronous and fully distributed setting. To overcome these problems the join-calculus was introduced as a model of distributed computation [FG96, Lév97, Fou98]. Mutual encodings between the (core) join-calculus and the asynchronous pi-calculus have shown that they have the same abstract expressive power [Fou98]. Here, we show a difference with respect to the degree of distributability. Hence, we explain what exactly distinguishes both calculi. It turns out that this distinction is well described by the synchronisation pattern M, i.e., what distinguishes the asynchronous pi-calculus and the join-calculus is the ability to express conflicts between distributable steps. This lack in expressiveness in turn allows

fully distributed implementations of the join-calculus.

In Section 4.3.1 the M on Petri nets is transferred into a new absolute result that distinguishes π_a and J. In Section 4.3.2 the result of [vGGS08] is adapted to provide a proof method for the separation of process calculi and the non-existence of a good and distributability-preserving encoding between π_a and J is shown. Section 4.3.3 then shows how this proof method is applied and adapted to show results for other process calculi.

4.3.1. The Absolute Result

If we compare the asynchronous pi-calculus and the join-calculus, the most obvious difference is that in J any input channel can appear only once. As a consequence, two conflicting steps in the join-calculus can only compete for different output messages but not for different input capabilities, as it is the case in π_a. Repeating this argument, all steps of a chain of conflicting steps in the join-calculus are tied to the same definition, i.e., are not distributable.

Lemma 4.3.1 (Absolute Result)**.** *For all $P \in \mathcal{P}_J$ and all lists $S = [s_1, \ldots, s_n]$ of steps of P such that for all $1 \leq i < n$ the step s_i is in conflict with the step s_{i+1}, all steps in S are pairwise local and reduce the same definition.*

Proof. Two steps are in conflict, if performing one step disables the other step. To do so the first step has to consume something necessary to perform the other step. In the join-calculus, it is not possible to consume input capabilities, i.e., definitions. Hence, the only way for a step to disable a former alternative step is to consume one of its necessary outputs. In the join-calculus, communication is allowed only on defined variables, i.e., to consume an output message the channel of that message has to be defined in a definition. Note that defining the syntactical representation of a name twice in different definitions results in two different names. Thus, if the first step consumes an output necessary to perform the second step, then both steps share a defined variable. Because of that, both steps must use the same definition, i.e., are not distributable. We conclude that for each list of alternative steps $S = [s_1, \ldots, s_n]$, where for all $1 \leq i < n$ the step s_i is in conflict with the step s_{i+1}, all steps in S use exactly the same definition. Thus, all pairs of steps in the set $S = \{ s_1, \ldots, s_n \}$ are pairwise local. □

In contrast, in π_a, it is very easy to find such a list of conflicting steps of which some are distributable, by combining conflicts on outputs and inputs.

Example 4.3.2. Consider $P = \overline{y}\langle u \rangle \mid y(x).P_1 \mid \overline{y}\langle v \rangle \mid y(x).P_2$ with $P \in \mathcal{P}_a$. P can perform four different alternative steps modulo structural congruence:

$$P \longmapsto \{\, {}^{u}/_{x} \,\} P_1 \mid \overline{y}\langle v \rangle \mid y(x).P_2 \qquad (s_1)$$

$$P \longmapsto y(x).P_1 \mid \overline{y}\langle v \rangle \mid \{\, {}^{u}/_{x} \,\} P_2 \qquad (s_2)$$

$$P \longmapsto \overline{y}\langle u \rangle \mid y(x).P_1 \mid \{\, {}^{v}/_{x} \,\} P_2 \qquad (s_3)$$

$$P \longmapsto \overline{y}\langle u \rangle \mid \{\, {}^{v}/_{x} \,\} P_1 \mid y(x).P_2 \qquad (s_4)$$

The step s_1 is in conflict with step s_2, since both compete for the first output $\overline{y}\langle u\rangle$. Similarly, steps s_2 and s_3 compete for the second input $y(x).P_2$, and step s_3 and step s_4 compete for the second output, i.e., P has a chain $S = [s_1, \ldots, s_4]$ of conflicting steps. But s_1 and s_3 as well as s_2 and s_4 are distributable in P.

Thus, the ability to express distributable conflicts separates the asynchronous pi-calculus from the join-calculus. However, the preservation of distributability in Definition 3.4.3 does not require to preserve the distributability of conflicts but only of processes and their executions, i.e., this difference alone is not enough to prove the non-existence of a distributability-preserving encoding. On the other side, the structure used in [vGGS08] to identify distributable Petri nets strongly relies on the notion of conflict. More precisely, an M arises from the combination of two parallel steps and a third step that is in conflict with both of the former.

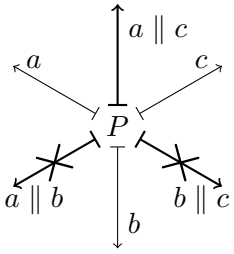


Figure 4.4.: Visualisation of the Synchronisation Pattern M.

Definition 4.3.3 (Synchronisation Pattern M)**.** Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus and $P \in \mathcal{P}$ such that:

1. P can perform at least three alternative reduction steps $a : P \longmapsto P_a$, $b : P \longmapsto P_b$, and $c : P \longmapsto P_c$ such that P_a, P_b, and P_c are pairwise different.

2. Moreover, the steps a and c are parallel in P.

3. But b is in conflict with both a and c.

In this case, we denote the process P as M. The synchronisation pattern M is visualised in Figure 4.4.[8] If the steps a and c are distributable in P, then we call the M *non-local*. Otherwise, the M is called *local*.

We observe, that the P of Example 4.3.2 represents an M in π_a, because we can choose the step s_1 as a, s_2 as b, and s_3 as c. Since s_1 and s_3 are distributable steps, P is a non-local M. The following example visualizes an M in the join-calculus.

[8] Note that a, b, and c are not labels. They serve just to distinguish different steps. Moreover, $x \parallel y$ refer to the parallel execution of x and y, given a step semantics.

Example 4.3.4 (Local M in the join-calculus)**.** Consider the J-term

$$P = \mathsf{def}\ x\,(z) \mid y\,(z') \triangleright z\,\langle z'\rangle\ \mathsf{in}\ (x\,\langle u\rangle \mid x\,\langle v\rangle \mid y\,\langle u\rangle \mid y\,\langle v\rangle)\,.$$

To show that P is an M, we can for example choose:

- $a : P \longmapsto u\,\langle u\rangle \mid \mathsf{def}\ x\,(z) \mid y\,(z') \triangleright z\,\langle z'\rangle\ \mathsf{in}\ (x\,\langle v\rangle \mid y\,\langle v\rangle)$,
- $b : P \longmapsto u\,\langle v\rangle \mid \mathsf{def}\ x\,(z) \mid y\,(z') \triangleright z\,\langle z'\rangle\ \mathsf{in}\ (x\,\langle v\rangle \mid y\,\langle u\rangle)$, and
- $c : P \longmapsto v\,\langle v\rangle \mid \mathsf{def}\ x\,(z) \mid y\,(z') \triangleright z\,\langle z'\rangle\ \mathsf{in}\ (x\,\langle u\rangle \mid y\,\langle u\rangle)$.

We observe that $u\,\langle u\rangle$, $u\,\langle v\rangle$, and $v\,\langle v\rangle$ are pairwise different. Moreover, the steps a and c are parallel, but b disables a as well as c, because it consumes $x\,\langle u\rangle$ necessary for a and $y\,\langle v\rangle$ necessary for c. Since P does only contain a single definition, all its steps are local. Hence, P is a local M in the join-calculus. And, since P is in fact a cJ-term, it is also a local M in the core join-calculus.

In contrast to Example 4.3.2, the M above is local. Indeed, all M in the join-calculus are local, because, by Lemma 4.3.1, the step b forces its conflicting counterparts to reduce the same definition.

Lemma 4.3.5 (Absolute Result)**.** *All* M *in the join-calculus are local.*

Proof. Assume the contrary, i.e., assume there is a non-local M in the join-calculus. Let us denote the corresponding J-term as P. By Definition 4.3.3, P can perform three alternative steps a, b, and c such that a and c are distributable but b is in conflict with both a and c. By Lemma 4.3.1, all conflicts in the join-calculus are local. Thus, all three steps a, b, and c are pairwise local, which contradicts the assumption that a and c are distributable. □

Thus, the asynchronous pi-calculus and the join-calculus do also differ by the ability to express a non-local M. As described in [vGGS08], a language that cannot express a non-local M can be considered as distributable. Accordingly, as intended by its design, the join-calculus is distributable. In the following we show that the pi-calculus is not distributable—not even in its asynchronous and choice-free fragment.

4.3.2. A new Separation Result

To show that the examined difference forbids distributability-preserving encodings, we have to show that it is not possible to express the abstract behaviour of all non-local M in the join-calculus with respect to our requirements on good and distributability-preserving encodings. We use the M of Example 4.3.2 as running counterexample. In the framework of Gorla, source terms and their encodings are compared by their ability to reach success. To distinguish the conflicting step $b = s_2$ from the parallel steps $a = s_1$ and $c = s_3$, we instantiate P_1 with $\overline{x}$, P_2 with $\overline{x} \mid \overline{x}$, and place the observer $O = u.v.v.\checkmark$ in parallel to P.

Example 4.3.6 (Running Counterexample)**.** The non-local M

$$S = (\overline{y}\langle u\rangle \mid y(x)\,.\overline{x}) \mid (\overline{y}\langle v\rangle \mid y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark) \tag{E1}$$

reaches success iff P performs both of the distributable steps a and c, where

Step a**:** $S \longmapsto S_a$ with $S_a = \overline{u} \mid \overline{y}\langle v\rangle \mid y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$ and $S_a \Downarrow_{\checkmark!}$,

Step b**:** $S \longmapsto S_b$ with $S_b = y(x)\,.\overline{x} \mid \overline{y}\langle v\rangle \mid \overline{u} \mid \overline{u} \mid u.v.v.\checkmark$ and $S_b \not\Downarrow_{\checkmark}$, and

Step c**:** $S \longmapsto S_c$ with $S_c = \overline{y}\langle u\rangle \mid y(x)\,.\overline{x} \mid \overline{v} \mid \overline{v} \mid u.v.v.\checkmark$ and $S_c \Downarrow_{\checkmark!}$.

To show that no good and distributability-preserving encoding can emulate E1, we use the fact that two distributable reductions in the join-calculus cannot reduce the same defined variable.

Lemma 4.3.7. *Let $P \in \mathcal{P}_{\mathrm{J}}$ and let A and C denote two distributable executions of P. Then the set of links of all outputs reduced in A and all outputs reduced in C are disjoint.*

Proof. Without loss of generality let us assume that there are no name clashes in P. Let D_A denote the set of defined variables of all outputs reduced in A, and let D_C denote the corresponding set for C. Let us assume A and C are distributable but $D_A \cap D_C \neq \emptyset$. Then there is some defined variable y such that an output on channel y is reduced in one step s_a of A, and an output on channel y is reduced in one step s_c of C. Since for each defined variable there is exactly one definition in the join-calculus, there is exactly one definition defining y. Because each step that reduces an output on channel y in the join-calculus has to use this definition, by Definition 3.4.5, s_a and s_c are not distributable. Hence, by Definition 3.4.6, A and C are not distributable, which contradicts our assumption. □

Note that any good encoding that preserves distributability has to translate E1 such that the emulations of the steps a and c are again distributable. However, the encoding can translate these two steps into sequences of steps, which allows to emulate the conflicts with the emulation of b by two different distributable steps. Hence, we show next that every distributability-preserving encoding has to distribute b and, afterwards, that this distribution of b violates the criteria for a good encoding.

Lemma 4.3.8. *Every encoding $[\![\,\cdot\,]\!] : \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{J}}$ that is good (except for compositionality) and distributability-preserving has to split up the conflict in S given by E1 of b with a and c such that there exists a maximal execution in $[\![\,S\,]\!]$ in which a is emulated but not c, and vice versa.*

Proof. By operational completeness (Definition 3.3.4), all three steps of S have to be emulated in $[\![\,S\,]\!]$, i.e., there exists $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ such that $[\![\,S\,]\!] \Longmapsto T_a \asymp [\![\,S_a\,]\!]$, $[\![\,S\,]\!] \Longmapsto T_b \asymp [\![\,S_b\,]\!]$, and $[\![\,S\,]\!] \Longmapsto T_c \asymp [\![\,S_c\,]\!]$. Because S has no infinite executions and $[\![\,\cdot\,]\!]$ reflects divergence (Definition 3.3.5), $[\![\,S\,]\!]$ has no infinite executions. By success sensitiveness (Definition 3.3.6), Lemma 3.3.9 and 3.3.10, and because $\asymp$ is success respecting (Definition 3.3.7), we have $T_a \Downarrow_{\checkmark!}$, $T_b \not\Downarrow_{\checkmark}$, $T_c \Downarrow_{\checkmark!}$, and $T_a \not\asymp T_b \not\asymp T_c$. We

conclude that, for all $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ such that $T_a \asymp [\![S_a]\!]$, $T_b \asymp [\![S_b]\!]$, and $T_c \asymp [\![S_c]\!]$ and for all sequences $A : [\![S]\!] \Longmapsto T_a$, $B : [\![S]\!] \Longmapsto T_b$, and $C : [\![S]\!] \Longmapsto T_c$, there is a conflict between a step of A and a step of B, and there is a conflict between a step of B and a step of C. Note, that since $T_b \not\Downarrow_{\checkmark}$ but $T_a \Downarrow_{\checkmark !}$ and $T_c \Downarrow_{\checkmark !}$, the conflict between a and b (or b and c) has to be translated into a conflict of A and B (or B and C). It is not possible that the emulation of b disables all ways to reach success after T_a or T_c is reached.

Because $[\![\cdot]\!]$ preserves distributability (Definition 3.4.3) and because of Lemma 3.4.8, the distributable steps a and c of S have to be translated into distributable executions, i.e., there is at least one A and one C such that these two executions are distributable. By Lemma 3.4.7, this implies that $[\![S]\!]$ is distributable into $T_1, T_2 \in \mathcal{P}_{\mathrm{J}}$ such that A is an execution of T_1 and C is an execution of T_2. By Lemma 4.3.1, the conflicts between A, B, and C are such that B and A as well as B and C compete for some output but, by Lemma 4.3.7, A and C do not reduce the same outputs. Hence, the two conflicts cannot be ruled out in a single step. Moreover, the reduction steps of A that lead to the conflicting step with B and the reduction steps of C that lead to the conflicting step with B are distributable, because A and C are distributable. We conclude, that there is at least one emulation of b, i.e., one execution $B : [\![S]\!] \Longmapsto T_b \asymp [\![S_b]\!]$, starting with two distributable executions such that one is (in its last step) in conflict with the emulation of a in $A : [\![S]\!] \Longmapsto T_a \asymp [\![S_a]\!]$ and the other one is in conflict with the emulation of c in $C : [\![S]\!] \Longmapsto T_c \asymp [\![S_c]\!]$. In particular this means that also the two steps of B that are in conflict with a step in A and a step in C are distributable.

Hence, there is no possibility to ensure that these two conflicts are decided consistently, i.e., there is a maximal execution of $[\![S]\!]$ that emulates A but not C as well as a maximal execution of $[\![S]\!]$ that emulates C but not A. □

Note that Lemma 4.3.8 describes a partial deadlock. If the emulation of b and with it the conflicts to the emulation of a and c are distributed, the encoded term can make the wrong decision and, thus, result in one successful emulation (of a or c) but two deadlocked emulation attempts of the respective other two steps. Since there is no execution of $E1$ with a but not c (or vice versa), such an encoding cannot be considered as a good encoding. Unfortunately, in the setting used so far, we cannot observe the difference in the abstract behaviour of $E1$ and $[\![E1]\!]$. One reason for that are the weak requirements on $\asymp$. A success respecting bisimulation, in its simplest case, cannot distinguish between more than three different cases: success is not reachable, success is reachable in every execution, and success is reachable in some but not all executions.

To prove non-existence of distribution-preserving encodings, it suffices to require that $\asymp$ is not trivial, e.g. by requiring that it distinguishes more than two observables. In this case, we have to modify $E1$, i.e., choose a suitable instantiation of P_1, P_2, and the observer, such that $[\![S_a]\!]$, $[\![S_b]\!]$, $[\![S_c]\!]$, and $[\![S_{ac}]\!]$ are pairwise distinguished by $\asymp$, where S_{ac} is the result of performing a and c in S. Then, the maximal execution that emulates a but not c contradicts operational correspondence.

Another way to show that there is no distribution-preserving encoding, is to make use of compositionality. Note that the best known encoding from the asynchronous pi-calcu-

lus into the (core) join-calculus in [FG96, Fou98] is not compositional, but consists of an inner, compositional encoding surrounded by a fixed context—the implementation of so-called firewalls—that is parameterised on the free names of the source term. Actually, it is this surrounding context that reduces the degree of distributability, because different steps on the same channel name have to synchronise on a firewall. The following result captures this and similar encodings.

Theorem 4.3.9 (Separation Result)**.** *There exists no good and distributability-preserving encoding from* π_{a} *into* J *and also no distributability-preserving encoding from* π_{a} *into* J *that is good except for compositionality but consists of an inner compositional encoding surrounded by a fixed context parametrised on the free names of the source term.*

Proof. Assume the contrary. Then there is a good and distributability-preserving encoding of the S given by E1. By the proof of Lemma 4.3.8, there is a maximal execution of $[\![S]\!]$ in that a but not c is emulated and success is reached, i.e., there is an execution such that the emulation of a leads to success without the emulation of c.

For encodings as described above, there exists a context $\mathcal{C} : \mathcal{P}_{\mathrm{J}}^2 \to \mathcal{P}_{\mathrm{J}}$—the combination of the surrounding context and the context introduced by compositionality (Definition 3.3.1)—such that $[\![S]\!] = \mathcal{C}([\![S_1]\!], [\![S_2]\!])$, where $S_1 = \overline{y}\langle u\rangle \mid y(x).\overline{x}$ and $S_2 = \overline{y}\langle v\rangle \mid y(x).(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$. Let $S_2' = y(x).(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$. Since $\mathsf{fn}(S_2) = \mathsf{fn}(S_2')$, also $S_1 \mid S_2'$ has to be translated by the same context, i.e., $[\![S_1 \mid S_2']\!] = \mathcal{C}([\![S_1]\!], [\![S_2']\!])$. Note that S and $S_1 \mid S_2'$ differ only by a capability necessary for step c, but step a and b are still possible. We conclude that, if $\mathcal{C}([\![S_1]\!], [\![S_2]\!])$ reaches some $T_a \Downarrow_{\checkmark !}$ without the emulation of c, then $\mathcal{C}([\![S_1]\!], [\![S_2']\!])$ reaches at least some state T_a' such that $T_a' \Downarrow_{\checkmark}$. Hence, $[\![S_1 \mid S_2']\!] \Downarrow_{\checkmark}$ but $(S_1 \mid S_2') \not\Downarrow_{\checkmark}$ which contradicts success sensitiveness. □

4.3.3. Transferring Separation Results

In the following we show how the proof method behind the above separation result can be transferred to other process calculi. Above, first an absolute result, i.e., a result that does refer to the properties of a single language, is derived in Lemma 4.3.1. It clarifies which property distinguishes the source and the target language, i.e., the reason why the target language does not contain the synchronisation pattern M. Then, the existence of the M in the source language is shown by an example which is used as counterexample in the following. Lemma 4.3.8 uses properties of the target language—basically the absolute result in Lemma 4.3.1—to show that any encoding has to split the conflict in the counterexample. Finally, Theorem 4.3.9 reasons about some properties of the source language to show that the split of the conflict in the encoded counterexample violates the criteria of a good encoding. This argumentation provides a guideline for similar considerations in other languages.

Accordingly, we consider two variants of CSP introduced in Section 2.1.3. First we replace the source language π_{a} by $\mathrm{CSP}_{\mathrm{in}}$—a variant of CSP with input and output guards and input guarded choice. Afterwards we replace the target language J by $\mathrm{CSP}_{\mathrm{no}}$—a subcalculus of $\mathrm{CSP}_{\mathrm{in}}$, where choice is only internally branching. Note that these two languages were already compared in [Bou88]. Here, we use them rather to explain how

the separation result above is transferred than to prove new results. For simplicity, we consider only compositional encodings in the following, but the results hold as well for combinations of an inner compositional encoding surrounded by a fixed context parametrised on the free names of the source terms as considered by Theorem 4.3.9.

Often, changing the source language is the easiest task, because it usually suffices to show that the new source language is expressive enough to provide the counterexample with the properties required by the absolute result. In the present case we have to show that $\mathrm{CSP}_{\mathrm{in}}$ contains a M similar to E1 and to recycle the argumentation in the proof of Theorem 4.3.9, thereby adapting it to the new source language. We gain the absolute result and Lemma 4.3.8 for free, because its proofs does not use any information about the source language except that it provides E1.

Example 4.3.10 (Non-Local M in $\mathrm{CSP}_{\mathrm{in}}$)**.** Consider

$$S = S_1 \parallel (S_2 \parallel S_3) \tag{E2}$$

with $S, S_1, S_2, S_3 \in \mathcal{P}_{\mathrm{in}}$, where $S_1 = [\,(\tau \to \mathrm{STOP}) \;\Box\; (b? \to \mathrm{STOP})\,]$, $S_2 = b! \to \mathrm{STOP}$, and $S_3 = [\,(b? \to \mathrm{STOP}) \;\Box\; (\tau \to \checkmark)\,]$. P can perform three different alternative steps modulo structural congruence:

Step a**:** $S \longmapsto S_a$ with $S_a = \mathrm{STOP} \parallel (S_2 \parallel S_3)$

Step b**:** $S \longmapsto S_b$ with $S_b = \mathrm{STOP} \parallel (\mathrm{STOP} \parallel \mathrm{STOP})$

Step c**:** $S \longmapsto S_c$ with $S_c = S_1 \parallel (S_2 \parallel \checkmark)$

If the first step is either a or c then S can perform the respective other step as second step. Moreover, the steps a and c are parallel and distributable but b is in conflict with a and c. If b is not performed, any maximal execution of S has two steps and leads to success. Hence, $S_a \Downarrow_{\checkmark!}$, $S_b \not\Downarrow_{\checkmark}$, and $S_c \Downarrow_{\checkmark!}$.

Since E2 and E1 have the same properties, we can show a separation result between CSP and J similar to Theorem 4.3.9.

Theorem 4.3.11 (Separation Result)**.** *There exists no good and distributability-preserving encoding from* $\mathrm{CSP}_{\mathrm{in}}$ *into* J*.*

Proof. Assume the contrary. Because S of Example 4.3.10 and E1 have the same properties, Lemma 4.3.8 holds also for S. Thus, there is a good and distributability-preserving encoding of S and there is a maximal execution of $[\![\, S \,]\!]$ in which a but not c is emulated and success is reached, i.e., there is an execution such that the emulation of a leads to success without the emulation of c.

Let $S_3' = [\,(b? \to \mathrm{STOP}) \;\Box\; (\tau \to \mathrm{STOP})\,]$. Because of compositionality (Definition 3.3.1) and since $\mathsf{fn}(S_3) = \mathsf{fn}(S_3')$, the terms $[\![\, S \,]\!]$ and $[\![\, S_1 \parallel (S_2 \parallel S_3') \,]\!]$ differ only by the encoding of S_3. Note that $S \Downarrow_{\checkmark}$ and $(S_1 \parallel (S_2 \parallel S_3')) \not\Downarrow_{\checkmark}$, but the possibilities to perform the steps a, b, and c remain unchanged. We conclude that, if $[\![\, S \,]\!]$ reaches some $T_a \Downarrow_{\checkmark!}$ without the emulation of c, then $[\![\, S_1 \parallel (S_2 \parallel S_3') \,]\!]$ reaches as least some state T_a' such that $T_a' \Downarrow_{\checkmark}$. Hence, $[\![\, S_1 \parallel (S_2 \parallel S_3') \,]\!] \Downarrow_{\checkmark}$ but $(S_1 \parallel (S_2 \parallel S_3')) \not\Downarrow_{\checkmark}$ which contradicts success sensitiveness. □

If we change the target language, we have to adapt the proof of Lemma 4.3.8. Therefore, we have first to revise the absolute result. To do so, we show that, because of the restrictive communication mechanism, without guards in choices all conflicts in $\mathrm{CSP}_{\mathrm{no}}$ are between τ-steps of a single choice only. Since choice is not distributable, all conflicts are local.

Lemma 4.3.12 (Absolute Result)**.** *All conflicts in* $\mathrm{CSP}_{\mathrm{no}}$ *are between* τ*-steps and are local.*

Proof. Two steps are in conflict if performing one step disables the other step by the consumption of a capability necessary to perform the other step. In $\mathrm{CSP}_{\mathrm{no}}$ a reduction step is either a τ-step or it reduces all unguarded capabilities of some subject. The later case is possible only if all parallel processes have an unguarded prefix with that subject and if there are not two unguarded output prefixes on this subject. Since all unguarded capabilities of some subject are reduced, an alternative step is either again a τ-step or a step on another subject. Without guards in choice it is not possible to remove an output or input prefix of subject y in a step on subject x. Thus, the only chance for conflicts is between τ-steps. The only way a τ-step may consume something necessary for an alternative step is within a choice. Since in $\mathrm{CSP}_{\mathrm{no}}$ only internal choice is allowed, i.e., all branches of a choice are guarded by τ, all conflicts in $\mathrm{CSP}_{\mathrm{no}}$ are between two τ-steps reducing the same internal choice. Since choice is not distributable, such steps are always local. □

As a consequence, all M in $\mathrm{CSP}_{\mathrm{no}}$ are also local, because of the conflict between b and a or c. Following the line of argumentation in Section 4.3.2, we show next that each good encoding of an M has to split up the conflicts of b with the steps a and c. It turns out that to adapt the proof of Lemma 4.3.8 it suffices to replace the argument on the absolute result in Lemma 4.3.1 by our new absolute result above.

Lemma 4.3.13. *Any good and distributability-preserving encoding from* π_{a} *(or* $\mathrm{CSP}_{\mathrm{in}}$*) into* $\mathrm{CSP}_{\mathrm{no}}$ *has to split up the conflict in* S *given by E1 (or by E2) of* b *with* a *and* c *such that there exists a maximal execution in* $[\![S]\!]$ *in which* a *is emulated but not* c*, and vice versa.*

Proof. The proof of Lemma 4.3.13 is similar to the proof of Lemma 4.3.8 above. It suffices to replace the sentences

> By Lemma 4.3.1, the conflicts between A, B, and C are such that B and A as well as B and C compete for some output but, by Lemma 4.3.7, A and C do not reduce the same outputs. Hence, the two conflicts cannot be ruled out in a single step.

by

> By Lemma 4.3.12, the conflicts between A, B, and C are such that B and A as well as B and C compete for some τ-capabilities within the same choice. Because the choice operator is not distributable but A and C are, the two conflicts cannot be ruled out in a single step.

Moreover, in case of E2, replace $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ with $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{in}}$. □

In this case we gain the argumentation in the proof of Theorem 4.3.9 and Theorem 4.3.11 for free.

Theorem 4.3.14 (Separation Result)**.** *There exists no good and distributability-preserving encoding from* π_{a} *(or* $\mathrm{CSP}_{\mathrm{in}}$*) into* $\mathrm{CSP}_{\mathrm{no}}$*.*

Proof. In case of π_{a}, the proof of Theorem 4.3.14 is similar to the proof of Theorem 4.3.9. It suffices to replace Lemma 4.3.8 by Lemma 4.3.13.

Else if the source language is $\mathrm{CSP}_{\mathrm{in}}$, the proof is similar to the proof of Theorem 4.3.11. Again it suffices to replace Lemma 4.3.8 by Lemma 4.3.13. □

4.4. Adapting an Absolute Result

In the last section we compare different process calculi by their ability to express the synchronisation pattern M. We learn that the different synchronisation mechanisms of the calculi lead to differences in the expressive power with respect to specific kinds of conflicts. By Section 4.2 we also know that the restriction in the choice operator leads to a separation result between π_{m} and π_{s}. Note that this separation result is based on the difference examined by the confluence property in Section 4.1.1. Confluence in turn basically describes the opposite of a conflict. Hence, in order to provide more intuitions on this separation result and on the difference in the expressive power of π_{m} and π_{s} with respect to conflicts, we prove once more a separation between these two calculi. More precisely, we show that the calculi can be distinguished by a new synchronisation pattern similar to the M but more complex. Not surprisingly, the new pattern combines again conflicting and distributable steps. Interestingly, it reflects a well-known standard problem in the area of distributed systems, namely the problem of the dining philosophers [Dij71, LR81, Lyn96, HP01].

We start with a simple observation on the asynchronous pi-calculus. Without choice each reduction step reduces exactly one output and one (replicated) input.

Observation 4.4.1. Each reduction step of π_{a} reduces exactly one output capability and exactly one (replicated) input capability, and all conflicts in π_{a} are on steps on the same link.

With separate choice a single step can reduce more than a single out- or input. But if we consider steps between two distributable subprocesses then each reduction step reduces only outputs in one subprocess and only inputs in the other. This reflects the confluence property stated in [Pal03]: without mixed choice consuming an input cannot immediately withdraw an output capability and vice versa.

Lemma 4.4.2 (Absolute Result)**.** *For all* $P \in \mathcal{P}_{\mathrm{s}}$ *and for all* $P_1, P_2 \in \mathcal{P}_{\mathrm{s}}$ *that are distributable within* P*, a reduction step between* P_1 *and* P_2 *either reduces only output guards in* P_1 *and only (replicated) input guards in* P_2*, or vice versa.*

Proof. By the reduction semantics of π_s in Figure 2.3, the derivation of each reduction step results from exactly one axiom, i.e., there are no branches in derivation trees of reduction steps in π_s. Moreover, a step between two distributable processes, i.e., a step that uses capabilities of two parallel composed processes, cannot result from PI-TAU$_{m,s}$. By the Axiom PI-COM$_{m,s}$ and PI-REP$_{m,s}$ an output guard within a sum and a (replicated) input guard of another sum are reduced, but no other output or (replicated) input guards are reduced outside the mentioned two sums. Remember that in π_s it is not allowed to place input and output guards within the same sum. Hence, if the step reduces an output guard in P_1 it has to reduce (replicated) input guards in P_2 but can neither reduce also input capabilities in P_1 nor output capabilities in P_2. The same holds if we swap the roles of P_1 and P_2. □

As a consequence, a chain of conflicting steps can build an M by alternating input and output capabilities as visualised in Example 4.3.2. But, by this method, no circle of odd length can be constructed as it is represented by the synchronisation pattern $\star$.

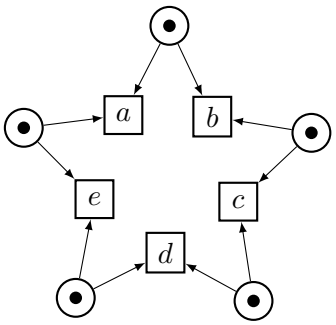


Figure 4.5.: The Synchronisation Pattern $\star$ in Petri nets.

Definition 4.4.3 (Synchronisation Pattern $\star$)**.** Let $\langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathcal{P}$ such that:

1. P can perform at least five alternative reduction steps $a : P \longmapsto P_a$, $b : P \longmapsto P_b$, $c : P \longmapsto P_c$, $d : P \longmapsto P_d$, and $e : P \longmapsto P_e$ such that P_a, P_b, P_c, P_d, and P_e are pairwise different.

2. Moreover, the steps a, b, c, d, and e form a circle such that a is in conflict with b, b is in conflict with c, c is in conflict with d, d is in conflict with e, and e is in conflict with a.

3. Finally, every pair of steps in $\{\, a, b, c, d, e \,\}$ that is not in conflict is parallel in P.

In this case, we denote the process P as $\star$. The synchronisation pattern $\star$ is visualised by the Petri net in Figure 4.5. If all pairs of parallel steps in $\{\, a, b, c, d, e \,\}$ are distributable in P, then we call the $\star$ *non-local*. Otherwise, the $\star$ is called *local*.

Note that for the pi-calculus the distinction between non-local and local $\star$—and also between non-local and local M above—is not important, because every $\star$ and every M is non-local in the pi-calculus.

To see the connection of this synchronisation pattern with the dining philosophers problem, consider the places in Figure 4.5 as the chopsticks of the philosophers, i.e., as resources, and the transitions as eating operations, i.e., as steps consuming resources. Each step needs mutually exclusive access to two resources and each resource is shared among two subprocesses. If both resources are allocated simultaneously, eventually exactly two steps are performed. As shown in the following, a fully distributable implementation of that pattern requires the expressive power of mixed choice.

By Example 4.3.2 we know that π_{s} can express distributable conflicts, but as shown in the following, π_{s} cannot express distributable conflicts that are arranged in a circle of odd degree greater than four as it is depicted by $\star$. To consider circles of degrees greater than four is necessary because smaller circles do not have parallel, i.e., distributable, steps. Hence, $\star$ represents the smallest example of the problematic structure, but separation can principally be proved for any such structure of odd degree and at least five steps. The main argument is that π_{s} can build chains of conflicts by alternating conflicts between output and input capabilities, but without mixed choice no cycle of odd length can be obtained in this way.

Lemma 4.4.4 (Absolute Result)**.** *There is no $\star$ in π_{s}.*

Proof. Assume the contrary, i.e., assume there is some $P \in \mathcal{P}_{\mathrm{s}}$ such that $a : P \longmapsto P_a$, $b : P \longmapsto P_b$, $c : P \longmapsto P_c$, $d : P \longmapsto P_d$, and $e : P \longmapsto P_e$ for some $P_a, P_b, P_c, P_d, P_e \in \mathcal{P}_{\mathrm{s}}$ that are pairwise different such that a is in conflict with b, b is in conflict with c, c is in conflict with d, d is in conflict with e, e is in conflict with a, and all pairs of steps in $\{ a, b, c, d, e \}$ that are not in conflict are parallel in P. Note that a communication step in π_{s} always reduces a sum of output guarded subterms and a replicated input or a sum of input guarded subterms. Accordingly, let i_x be the replicated input or the sum of input guards reduced by step $x \in \{ a, b, c, d, e \}$ in P and o_x be the sum of output guards reduced by step x, respectively.

Since a and c are distributable in P, by Lemma 3.4.7, P is distributable into the terms $P_1, P_2 \in \mathcal{P}_{\mathrm{s}}$ such that a is a step of P_1 and c is a step of P_2, i.e., there exists $P_1', P_2' \in \mathcal{P}_{\mathrm{s}}$ and a sequence of names $\tilde{x}$ such that $P \equiv (\nu\tilde{x})\,(P_1 \mid P_2)$, $P_a \equiv (\nu\tilde{x})\,(P_1' \mid P_2)$, and $P_c \equiv (\nu\tilde{x})\,(P_1 \mid P_2')$. Because b is in conflict with a and c, it reduces one capability in P_1 and one capability in P_2, i.e., b is a communication between P_1 and P_2. By Lemma 4.4.2, b reduces either only input capabilities or only output capabilities in P_1. Let us assume that b reduces only output capabilities in P_1. Since b is in conflict with a, it reduces an unguarded output capability in $o_a = o_b$, i.e., a and b compete for outputs in the same sum. Then, again by Lemma 4.4.2, the conflict between b and c comes from a competition between capabilities in $i_b = i_c$ in P_2. b and d are distributable, but c is in conflict with b and d. We know that the conflict between b and c comes from the competition between capabilities in $i_b = i_c$. By the same argumentation as before, then c and d compete for capabilities in $o_c = o_d$. Furthermore, d and e compete for capabilities in $i_d = i_e$. And thus, e and a compete for $o_e = o_a$. But then e and a as well as a and

b compete for capabilities in o_a. Because e and b are distributable and sums are not distributable, they cannot reduce the same output guarded sum. This is a conflict. (By the way, even if it would have been possible to distribute the output guarded sum, we could apply Lemma 4.4.2 once more to derive the conflict as in the second case.)

Now, in order to capture the other case, let us assume that a and b compete for a capability in $i_a = i_b$. Thus, b and c compete for a capability in $o_b = o_c$, c and d compete for a capability in $i_c = i_d$, d and e compete for a capability in $o_d = o_e$, and e and a compete for a capability in $i_e = i_a$. By Lemma 4.4.2, if a is in conflict with e and b, it is not possible that a reduces an input capability in e as well as in b, i.e., again this is a conflict. □

The following example shows that π_{m}, in contrast to π_{s}, can express the synchronisation pattern $\star$. We use this example as running counterexample in the following.

Example 4.4.5 ($\star$ in π_{m})**.** Consider a term $S \in \mathcal{P}_{\mathrm{m}}$ such that

$$S = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5 \tag{E3}$$

for some $S_1, \ldots, S_5 \in \{\, \mathsf{0}, \checkmark \,\}$. Then, S can perform the steps

Step a**:** $S \longmapsto S_a$ with $S_a = \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid S_5$,

Step b**:** $S \longmapsto S_b$ with $S_b = S_1 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5$,

Step c**:** $S \longmapsto S_c$ with $S_c = \overline{a} + b.S_1 \mid S_2 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5$,

Step d**:** $S \longmapsto S_d$ with $S_d = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid S_3 \mid \overline{e} + a.S_5$, and

Step e**:** $S \longmapsto S_e$ with $S_e = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid S_4$.

Obviously, a is in conflict with b and e but parallel to c and d, b is in conflict with c but parallel to d and e, c is in conflict with d but parallel to e, and d is in conflict with e. Hence, by Definition 4.4.3, S is a non-local $\star$.

Unfortunately the same cyclic dependencies between the conflicts in $\star$ that are used in the proof of Lemma 4.4.4 prevent us from initialising $S_1, \ldots, S_5$ such that $S_x \Downarrow_{\checkmark!}$ and $S_y \not\Downarrow_{\checkmark}$ for each pair of conflicting steps x and y. Note that in the proof of Lemma 4.3.8 we use the properties $S_a \Downarrow_{\checkmark!}$, $S_b \not\Downarrow_{\checkmark}$, and $S_c \Downarrow_{\checkmark!}$ to ensure that the conflict of b with a and c has to be translated into a conflict of $B : [\![S]\!] \Longmapsto T_b \asymp [\![S_b]\!]$ with $A : [\![S]\!] \Longmapsto T_a \asymp [\![S_a]\!]$ and $C : [\![S]\!] \Longmapsto T_c \asymp [\![S_c]\!]$. Here we use compositionality and the fact that initialising S_i for $1 \leq i \leq 5$ by either $\checkmark$ or $\mathsf{0}$ has no consequence on the surrounding contexts in the encoding, to show that the encoding has to preserve also the conflicts in E3.

Lemma 4.4.6. *Any good and distributability-preserving encoding* $[\![\cdot]\!]$ *from* π_{m} *into* π_{s} *has to translate the conflicts in* S *given by E3 into conflicts of the corresponding emulations.*

Proof. By operational completeness (Definition 3.3.4), all five steps of S have to be emulated in $[\![S]\!]$, i.e., there exists $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_\mathrm{s}$ such that $[\![S]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ for all $x \in \{ a, b, c, d, e \}$. Because $[\![\cdot]\!]$ preserves distributability, for each pair of steps x and y that are parallel in S, the emulations $X : [\![S]\!] \Longmapsto T_x$ and $Y : [\![S]\!] \Longmapsto T_y$ such that $T_x \asymp [\![S_x]\!]$ and $T_y \asymp [\![S_y]\!]$ are distributable. Note that X and Y refer to the upper case variants of x and y, respectively.

In Example 4.4.5 we do not initialise $S_1, \ldots, S_5$. Now, we consider all variants of S, where $S_1, \ldots, S_5 \in \{ \mathsf{0}, \checkmark \}$, i.e., each of these terms is either chosen to be empty or to present an unguarded occurrence of success. Since $\mathsf{n}(\checkmark) = \mathsf{n}(\mathsf{0}) = \emptyset$ and because of compositionality (Definition 3.3.1), the encodings of these variants of S differ only by the encodings of $S_1, \ldots, S_5$. The remaining operators and, hence, the remaining term has to be translated in exactly the same way. Accordingly, the encoding of a term $S_1, \ldots, S_5$ cannot influence the emulation of the steps of S.

Thus, for each triple of steps $x, y, z \in \{ a, b, c, d, e \}$ in S such that y is in conflict with x and z but x and z are parallel, we can choose $S_{\mathrm{f}(x)} = \checkmark = S_{\mathrm{f}(z)}$ and initialise all other terms in $\{ S_1, \ldots, S_5 \}$ by $\mathsf{0}$, where

$$\mathrm{f}\,(u) = \begin{cases} 1, & \text{if } x = b \\ 2, & \text{if } x = c \\ 3, & \text{if } x = d \\ 4, & \text{if } x = e \\ 5, & \text{if } x = a \end{cases}$$

for all $u \in \{ a, b, c, d, e \}$, such that $S_{\mathrm{f}(x)} \Downarrow_{\checkmark !}$ and $S_{\mathrm{f}(z)} \Downarrow_{\checkmark !}$ but $S_{\mathrm{f}(y)} \not\Downarrow_{\checkmark}$. Then, also $S_x \Downarrow_{\checkmark !}$ and $S_z \Downarrow_{\checkmark !}$ but $S_y \not\Downarrow_{\checkmark}$. Now we can proceed as in the proof of Lemma 4.3.8. Because S has no infinite execution and $[\![\cdot]\!]$ reflects divergence, $[\![S]\!]$ has no infinite execution. By success sensitiveness, Lemma 3.3.9 and 3.3.10, and because $\asymp$ is success respecting, we have $T_x \Downarrow_{\checkmark !}$, $T_y \not\Downarrow_{\checkmark}$, $T_z \Downarrow_{\checkmark !}$, and $T_x \not\asymp T_y \not\asymp T_z$. We conclude that, for all $T_x, T_y, T_z \in \mathcal{P}_\mathrm{J}$ such that $T_x \asymp [\![S_x]\!]$, $T_y \asymp [\![S_y]\!]$, and $T_z \asymp [\![S_z]\!]$ and for all sequences $X : [\![S]\!] \Longmapsto T_x$, $Y : [\![S]\!] \Longmapsto T_y$, and $Z : [\![S]\!] \Longmapsto T_z$, there is a conflict between a step of X and a step of Y, and there is a conflict between a step of Y and a step of Z. □

Similar to Section 4.3.2, we show that each good encoding of the counterexample requires that a conflict has to be distributed.

Lemma 4.4.7. *Any good and distributability-preserving encoding $[\![\cdot]\!]$ from π_m into π_s has to split up a least one of the conflicts in S given by E3 such that there exists a maximal execution in $[\![S]\!]$ in which only one source term step is emulated.*

Proof. By operational completeness (Definition 3.3.4), all five steps of S have to be emulated in $[\![S]\!]$, i.e., there exists $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_\mathrm{s}$ such that $X : [\![S]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ for all $x \in \{ a, b, c, d, e \}$, where X is the upper case variant of x. By Lemma 4.4.6, for all $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_\mathrm{s}$ and all $x \in \{ a, b, c, d, e \}$ such that $T_x \asymp [\![S_x]\!]$, there is a conflict

between a step of the following pairs of emulations: A and B, B and C, C and D, D and E, and E and A.

Since $[\![\cdot]\!]$ preserves distributability (Definition 3.4.3) and by Lemma 3.4.3, each pair of distributable steps in S has to be translated into emulations that are distributable within $[\![S]\!]$. Let $X, Y, Z \in \{ A, B, C, D, E \}$ be such that X and Z are distributable within $[\![S]\!]$ but Y is in conflict with X as well as Z. By Lemma 3.4.7, this implies that $[\![S]\!]$ is distributable into $T_1, T_1 \in \mathcal{P}_\mathrm{s}$ such that X is an execution of T_1 and Z is an execution of T_2. Since Y is in conflict with X and Z and because all three emulations are executions of $[\![S]\!]$, there is one step of Y that is in conflict with one step of X and there is one (possibly the same) step of Y that is in conflict with one step of Z. Moreover, since X and Z are distributable, if a single step of Y is in conflict with X as well as Z then this step is a communication between T_1 and T_2.

Assume that for all such combinations X, Y, and Z, the conflicts between Y and X or Z are ruled out by a single step of Y, i.e., both conflicts are ruled out by a communication step between some capabilities of X and some capabilities of Z. By Lemma 4.4.2, then this step reduces only input capabilities in one of the executions X and Z and only output capabilities in the respective other, i.e., X and Y compete either only for input or only for output capabilities and Y and Z compete for the respective other kind of capabilities. Without loss of generality let us assume that A and B compete for some output capabilities and, thus, B and C compete for some input capabilities, C and D compete for some output capabilities, D and E compete for some input capabilities, E and A compete for some output capabilities, and A and B compete for some input capabilities. This is a contradiction, because, by Lemma 4.4.2, A and B cannot compete for both input and output capabilities.

We conclude that there is at least one triple of emulations X, Y, and Z such that the conflict of Y with X and with Z results from two different steps in Y. Because X and Z are distributable, the reduction steps of X that leads to the conflicting step with Y and the reduction steps of Z that leads to the conflicting step with Y are distributable. We conclude, that there is at least one emulation of y, i.e., one execution $Y : [\![S]\!] \Longmapsto T_y \asymp [\![S_y]\!]$, starting with two distributable executions such that one is (in its last step) in conflict with the emulation of x in $X : [\![S]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ and the other one is in conflict with the emulation of z in $Z : [\![S]\!] \Longmapsto T_z \asymp [\![S_z]\!]$. In particular this means that also the two steps of Y that are in conflict with a step in X and a step in Z are distributable. Hence, there is no possibility to ensure that these two conflicts are decided consistently, i.e., there is a maximal execution of $[\![S]\!]$ that emulates X but neither Y nor Z.

In the set $\{ A, B, C, D, E \}$ there are—apart from X, Y, and Z—two remaining executions. One of them, say X', is in conflict with X and the other one, say Z', is in conflict with Z. Since X is emulated successfully, X' cannot be emulated. Moreover, note that Y and Z' are distributable. Thus, also Z' and the partial execution of Y that leads to the conflict with Z are distributable. Moreover, also the step of Y that already rules out Z cannot be in conflict with a step of Z'. Thus, although the successful completion of Z is already ruled out by the conflict with Y, there is some step of Z left, that is in conflict with one step in Z'. Hence, the conflict between Z and Z' cannot be ruled out

by the partial execution described so far that leads to the emulation of X but forbids to complete the emulations of X', Y, and Z. Thus, it cannot be avoided that Z wins this conflict, i.e., that also Z' cannot be completed. We conclude that there is a maximal execution of $[\![S]\!]$ such that only one of the five source term steps of S is emulated. □

Note that each maximal execution of E3 consists of exactly two distributable steps. So, Lemma 4.4.7 leads to a conflict with the requirements on a good and distributability-preserving encoding.

Theorem 4.4.8 (Separation Result)**.** *There exists no good and distributability-preserving encoding from* π_{m} *into* π_{s}*.*

Proof. Assume the contrary, i.e., there is a good and distributability-preserving encoding $[\![\cdot]\!]$ from π_{m} into π_{s}, and, thus, also of S given by E3. Since S has no infinite executions and because $[\![\cdot]\!]$ is divergence reflecting, $[\![S]\!]$ has no infinite executions. By Lemma 4.4.7 there exists a maximal execution in $[\![S]\!]$ in which only one source term step is emulated. Let us denote this step by $x \in \{ a, b, c, d, e \}$. Hence, $[\![S]\!] \Longmapsto T_x \Longmapsto T$ with $T \mapsto\!\!\!\!\!\!\rightarrow$ for some $T \in \mathcal{P}_{\mathrm{s}}$, because there is no infinite execution. Moreover, by operational soundness, $T_x \asymp T$, because after the emulation of x no other step is emulated. Note that since we do not fix $S_1, \ldots, S_5$ in Example 4.4.5 and by the argumentation in the Lemma 4.4.6 and 4.4.7, the above conditions hold for all variants of S such that $S_1, \ldots, S_5 \in \{ \mathsf{0}, \checkmark \}$. Let us consider the case that $S_{\mathrm{f}(x)} = \mathsf{0} = S_{\mathrm{f}(y)}$ but all other terms in $\{ S_1, \ldots, S_5 \}$ are equal to $\checkmark$, where f is the function defined in the proof of Lemma 4.4.6 and y is a step that is parallel to x within S. Then, $S_x \Downarrow_{\checkmark}$ but $S_x \not\Downarrow_{\checkmark!}$, i.e., the step x may lead to success (in case the next step is not y) or it does not lead to success (in case the next step is y). By success sensitiveness and because $\asymp$ is success sensitive, then also $T_x \Downarrow_{\checkmark}$ and $T \Downarrow_{\checkmark}$ but $T_x \not\Downarrow_{\checkmark!}$ and $T \not\Downarrow_{\checkmark!}$. But this contradicts the property that $T \mapsto\!\!\!\!\!\!\rightarrow$, because a term in π_{s} that cannot perform a step either already has an unguarded occurrence of success or can never reach some. We conclude that there cannot be such an encoding. □

Note that we could derive the same result if, as in Section 4.3.2, we allow for a not compositional encoding that consists of an inner compositional encoding surrounded by a fixed context parametrised on the free names of the source term. Moreover, since the synchronisation pattern $\star$ includes the pattern M—more precisely it consists of three cyclic overlapping M—separation results derived from these two patterns (with respect to the same quality criteria) automatically lead to a hierarchy. Here, by Theorem 4.3.9, Theorem 4.3.14, and Theorem 4.4.8, there exists no good and distributability-preserving encoding from π_{m} into J or $\mathrm{CSP}_{\mathrm{no}}$.

Corollary 4.4.9 (Separation Result)**.** *There exists no good and distributability-preserving encoding from* π_{m} *into* J *or* $\mathrm{CSP}_{\mathrm{no}}$*.*

4.5. Summary and Related Work

Within this chapter we showed the relevance of absolute results for the derivation of translational separation results. We showed how absolute results can be derived from

a syntactical difference between the considered calculi (Section 4.1.1), from standard problems (Section 4.1.2), by generalising another absolute result (Section 4.1.3), from problems used in other formalisms (Section 4.3.1), by transferring another absolute result on other process calculi (Section 4.3.3), and by adaptation of another absolute result (Section 4.4). In Section 4.2 we showed how different translational separation results with respect to different quality criteria can be obtained from a single absolute result. In particular the first three separation results in Section 4.2.1 reveal a general proof schema of translational separation results on top of absolute results, because in these cases the absolute result is well-suited for the considered sets of quality criteria. All these proofs argument by contradiction. Then:

1. A counterexample is derived from the positive absolute result, i.e., an example illuminating how the problem considered in the absolute result is solved in the source language is chosen as counterexample in the separation proof.

2. It is shown how the main properties of the counterexample, i.e., the properties that allow for a solution of the considered problem, are preserved by the quality criteria. The easier this is possible the better the respective absolute result is suited for the considered setting.

3. The negative absolute result is applied, i.e., the proof that the respective problem cannot be solved in the target language, to construct a contradiction with the preserved properties of the counterexample.

Also the separation results in Section 4.3 and Section 4.4 follow the same line of argumentation. Absolute results alone can neither be used to show that two languages have the same expressive power nor that a language is strictly more expressive than another, because they supply only an isolated single result with respect to a single problem instance and without any consequence to other not related problems. However, as we observe in the above separation results and other separation results in the literature (as e.g. [Bou88, HP01, CM03, Pal03, VPP07, Gor08a, Gor09, FL10, Gor10b, LV10]) we can benefit from absolute results for the derivation of translational separation results. Absolute results, in contrast to translational results, do not rely on any notion of quality criteria and are thus applicable in different settings. Moreover, they often reveal an additional intuition on the underlying difference between two languages and can guide a separation proof.

In Section 4.1 we proved that π_{m} is strictly more expressive than π_{s} by means of an absolute separation result about the ability to break initial symmetries. This result is independent of any notion of uniformity or reasonableness, i.e., does not rely on any set of quality criteria. Note that this absolute separation result implies that if there exists a good encoding from π_{m} into π_{s} then it has to be the encoding itself that breaks initial source term symmetries. By choosing the problem of breaking initial symmetries instead of leader election, we significantly weaken the underlying definition of symmetry in comparison to [Pal03]. Moreover, we could still apply our absolute separation result to derive that there is no uniform and reasonable encoding from π_{m}

into π_s (Section 4.2.1) considering three different definitions of reasonableness. It turns out that the concentration on the underlying problem of breaking initial symmetries allows us to use counterexamples different from leader election in translational separation results. Likewise, the separation result in the setting of [Gor10b] can be derived by our absolute result as well. Besides that, our absolute separation result allows us to weaken the definition of uniformity in comparison to the translational separation result of [Pal03], and also to weaken the definition of reasonableness in comparison to the translational separation result in the first setting of [Gor10b]. Moreover, considering our last translational separation result, we can even withdraw the assumption of divergence reflection. We conclude that the problem of breaking initial symmetries is better suited to obtain a translational separation result between π_m and π_s than the leader election problem originally used in [Pal03] to separate these two languages. Moreover, we prove that a difference between the expressive power of mixed and separated choice cannot only be derived with respect to the homomorphic translation of the parallel operator but also with respect to the weaker requirement on the preservation of distributability and that this difference forbids also for encodings that reflect the causal semantics of the source.

In order to clarify the difference between the respective calculi we presented a new separation result between the asynchronous pi-calculus and the join-calculus in Section 4.3 and between π_m and π_s in Section 4.4. The first results shows why the join-calculus can be considered as distributable, while the asynchronous pi-calculus is in general not distributable. The later result sheds more light on the difference in the expressive power of mixed choice compared to separate choice. To do so, we presented two generally formulated synchronisation patterns that expose the power of different synchronisation mechanisms in the pi-calculus family but can be used in a similar manner to reason about and to classify synchronisation mechanisms in other process calculi. We showed that absolute results based on synchronisation patterns reveal a proof method that is in general well suited to reason about the expressive power of synchronisation mechanisms by showing how the first result can be transferred with little effort to compare other source and target languages.

Note that all presented separation results between the different variants of the pi-calculus remain valid if we add the match prefix to the target language. In particular, also $\pi_s^=$ can not break symmetries, there is no $\star$ in $\pi_s^=$, and there exists no good encoding from π_m into $\pi_s^=$ that translates the parallel operator homomorphically or preserves distributability.

As already mentioned the relationship between the synchronous and the asynchronous pi-calculus was first analysed in [Pal03]. Later on, [VPP07] review this result and extend the consideration of leader election to a general proof technique for the comparison of process calculi. However, as already [Pal03], they rely on the homomorphic translation of the parallel operator to measure the quality of translational results with respect to the preservation of distribution. As discussed in Section 3.4, we consider this criterion as too strict for separation results. Also [CCP07] present a separation result between π_m and π_s. In contrast to [Pal03, VPP07] they do not require that the parallel operator is translated homomorphically, but they consider must-testing with an explicit observer

o. More precisely, they require P *must* o implies $[\![P]\!]$ *must* $[\![o]\!]$. From that they derive $[\![P \mid o]\!] = [\![P]\!] \mid [\![o]\!]$. We assume that the negative result is based on this introduction of homomorphy in the consideration of the process and the observer. Note that within the framework of Gorla [Gor10b] must-testing can also be applied with an explicit observer. But in general $[\![P \mid o]\!] = [\![P]\!] \mid [\![o]\!]$ does not hold. Instead compositionality only implies that $[\![P \mid o]\!] = \mathcal{C}_{\mid}^{N}([\![P]\!], [\![o]\!])$, where $\mathcal{C}_{\mid}^{N}$ is the context that is introduced in order to encode the parallel operator and $N = \mathsf{fn}(P) \cup \mathsf{fn}(o)$.

Encodings of choice, i.e., positive translational results between π_{m}, π_{s}, and π_{a}, are considered by Nestmann in [Nes00]. We discuss the positive result between π_{s} and π_{a} in the following chapter. Moreover, Nestmann presents some candidates for an encoding from π_{m} into π_{a} and shows that each either violates compositionality or divergence reflection. However, the encodings presented in [Nes00] are the basis for our encoding from π_{m} into π_{a} in the following chapter. Note that the expressive power of choice is also considered in other process calculi (see e.g. [BG95] for CCS or [Bou88] for CSP).

Also [FL10] compares different variants of the pi-calculus. In particular, they present another proof for the negative result between π_{m} and π_{s}. In contrast to [Gor10b] and the results presented here, they use a stricter notion of operational correspondence that does not allow for intermediate states or partial commitment.

5. The Design of Encodings

Mixed choice is a widely-used primitive in process calculi. It is interesting, as it allows to break symmetries in distributed process networks. The main purpose of this chapter is the presentation of a good encoding of mixed choice in the context of the pi-calculus.

It is well known that there is a good encoding from the choice-free synchronous pi-calculus into its asynchronous variant [Bou92, HT91, Hon92a]. The results in [Pal03], [Gor10b], and (the first half of) Chapter 4 also tell us that there is no good encoding from the synchronous pi-calculus including mixed choice (π_{m}) into its asynchronous fragment without choice (π_{a}) if the encoding translates the parallel operator homomorphically. Moreover, Section 4.1.3 and Section 4.2.1 tell us that the main reason for these separation results boils down to the fact that the full pi-calculus can break merely syntactic symmetries, whereas its asynchronous variant cannot and if there is a good encoding of mixed choice then this encoding has to break initial source term symmetries itself. Finding a reasonable, divergence reflecting encoding of mixed choice is an open problem. An "almost reasonable" divergent encoding was already presented in [Nes00]. It shows that, if divergence reflection is not required, the encoding can ensure that all undesired symmetric executions are divergent such that it is not necessary for the encoding function to break symmetry. [Nes00] also shows that symmetry can be broken globally by means of some centralised artefacts, which of course violate compositionality. In the following we present a symmetry breaking encoding by abandoning the requirement on the homomorphic translation of the parallel operator. The result is an encoding from π_{m} into $\pi_{\mathrm{a}}^{=}$ that meets all five of Gorla's criteria[1] and that exploits the parallel structure of source terms to break symmetries locally. So, merely considering the (abstract) behaviour of terms, the full pi-calculus and its asynchronous variant have the same expressive power.

We start with the discussion of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ from π_{s} into π_{a} that was introduced in [Nes00]. This encoding serves as starting point for the derivation of our new encoding. As already stated in [Nes96] there are several good reasons for dividing process calculus encodings into several phases. Accordingly, we extend $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ first to an intermediate encoding in Section 5.2.2 from π_{m} into π_{p} (an asynchronous variant of the pi-calculus augmented with polyadic synchronisation [CM03]) before we present the final encoding in Section 5.2.4. More precisely, we use the expressive power of polyadic synchronisation to abstract away from some technical details of the final encoding. Then, we further extend and refine the encoding to get rid of the introduced polyadic synchronisation, i.e., to obtain an encoding into the less expressive target language $\pi_{\mathrm{a}}^{=}$. Moreover, in Section 5.3 we present a second attempt for a good encoding from π_{m} into π_{a} that results

[1] Note that this encoding is neither prompt nor is the assumed equivalence $\asymp$ strict, so the similar separation results of [Gor10b] do not apply here.

from another idea to circumvent deadlocks. In Section 5.4 we discuss the composition of two good encodings.

5.1. Concept and Implementation

An encoding is a function from the process terms of a source language into the process terms of a target language, i.e., it is a formal mapping from the syntax of the source language into the syntax of the target language. Hence, given this formal mapping, it should be self-explanatory if the source and the target language are well-known or exhaustively explained. From a mathematical point of view this is surely correct and hence it should suffice to present an encoding by providing this function. However, encodings quickly become huge and complex. Thus, presenting just their functional representation is usually unsatisfactory.

$$
\begin{aligned}
[\![(\nu x)\, P]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\![P]\!]^{\mathrm{s}}_{\mathrm{a}} \\
[\![P \mid Q]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq [\![P]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![Q]\!]^{\mathrm{s}}_{\mathrm{a}} \\
\left[\!\!\left[\sum_{i\in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu l) \left(\overline{l}\langle \top \rangle \mid \prod_{i\in I} [\![\pi_i.P_i]\!]^{\mathrm{s}}_{\mathrm{a}} \right) \\
[\![\tau.P]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \left(\overline{l}\langle \bot \rangle \mid [\![P]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\ \mathsf{else}\ \overline{l}\langle \bot \rangle \\
[\![\overline{y}\langle z \rangle .P]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu s) \left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z) \rangle \mid s.\, [\![P]\!]^{\mathrm{s}}_{\mathrm{a}} \right) \\
[\![y(x) .P]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq (\nu r) \left(\overline{r} \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(y)\left(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\right) . \right. \\
&\qquad \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle \bot \rangle \mid \overline{l'}\langle \bot \rangle \mid \overline{s} \mid [\![P]\!]^{\mathrm{s}}_{\mathrm{a}} \\
&\qquad\qquad \mathsf{else}\ \overline{l}\langle \top \rangle \mid \overline{l'}\langle \bot \rangle \mid \overline{r} \\
&\qquad \left. \mathsf{else}\ \overline{l}\langle \bot \rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\left\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x) \right\rangle \right) \\
[\![y^{\star}(x) .P]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq \varphi^{\mathrm{s}}_{\mathrm{a}}(y)^{\star}(l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)) .\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle \bot \rangle \mid \overline{s} \mid [\![P]\!]^{\mathrm{s}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle \bot \rangle \\
[\![\checkmark]\!]^{\mathrm{s}}_{\mathrm{a}} &\triangleq \checkmark
\end{aligned}
$$

Figure 5.1.: An Encoding from π_{s} into π_{a}.

Consider, as an example, the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ from the pi-calculus with separate choice (π_{s}) into the asynchronous pi-calculus without choice (π_{a}) in Figure 5.1. It is (a slightly adapted version of an encoding) presented by Nestmann in [Nes00]. Note that this encoding is the simplest encoding considered within this chapter. Nonetheless, its presentation in Figure 5.1 is far from being self-explanatory. Therefore it is standard to explain the intuition of an encoding and its underlying ideas. Often also its development is illustrated by presenting problems of its design, showing wrong tracks, and explaining the idea behind solutions to overcome these problems. If there is enough space, often also examples of encoded terms and their behaviour are presented, to visualise the main idea of the encoding function. Sometimes abbreviations are introduced or intermediate

languages are used, to ease its presentation as well as its proof of correctness. An attentive reader may have noticed the construct $\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \ldots\ \mathsf{else}\ \ldots\ \mathsf{else}\ \ldots$, which is of course not part of the syntax of π_{a}, as well as the symbols $\top$ and $\bot$. Indeed, they are abbreviations introduced in [Nes96, Nes00] to visualise the main idea of the encoding function and ease its presentation. Moreover, in [Nes96] these abbreviations are rendered into an intermediate language to simplify the proof of correctness of encodings. Of course, the proof of its correctness always reveals many information about the underlying concept of an encoding function.

Within this thesis we deliberately distinguish between the intuition or idea of an encoding, denoted as its *concept*, and the *implementation* of this concept within the encoding function as terms of the target language. We do so because this distinction does not only help to explain how the encoding works, but also significantly eases to extend an encoding, to adapt it, or to transfer it to other source or target languages. Note that the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ above builds the basis of the main contribution of this chapter: an encoding of mixed choice that satisfies all criteria of the general framework in Section 3.3. Because of that, we review the presentation of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ in [Nes00]. For this, we present its underlying concept in Section 5.1.1, show how this concept is implemented to achieve $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and explain the abbreviations introduced for this encoding in Section 5.1.2, and visualise the main idea of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ on a small example in Section 5.1.3. For a more exhaustive explanation and analysis of this encoding including a discussion of its properties and on the problems of its development we refer to [Nes00]. There, also the correctness of this encoding with respect to divergence reflection and operational correspondence as well as a full abstraction result is proved. However, in order to compare this encoding to the encodings derived in this thesis and to obtain a hierarchy in Section 7.2, we prove correctness of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ with respect to the criteria of Section 3.3 in Chapter 6.

5.1.1. Concept of the Encoding

The encoding in Figure 5.1 is an encoding from π_{s}—a synchronous variant of the pi-calculus with separate choice—into π_{a}—the asynchronous pi-calculus without choice. Because of that, we denote it as $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. There are two syntactical differences between π_{s} and π_{a}. Since π_{s} is a synchronous variant of the pi-calculus it allows for output guards. By contrast, in π_{a} outputs can only guard the empty process. However, as already shown by [HT91] and [Bou92], this syntactical restriction does not influence the expressive power of the calculi. The second syntactical difference is the absence of choice in π_{a}. Although this difference is much more crucial, the good encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ shows that also this difference has little effect on the expressive power. To compensate for the lacking choice operator in π_{a}, $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ places (the encodings of) the branches of a source term sum in parallel. A locking mechanism ensures that nonetheless only a single of this now parallel arranged branches can be chosen to perform a step or, more precisely, to emulate a source term step.

Accordingly, the encoding introduces a so-called *sum lock* l carrying a boolean value for each sum. In the following we denote such a lock with a boolean value as boolean lock. Initially exactly one positive instantiation of this lock is provided to ensure that at

most one of its branches can be chosen. Consequently, the emulation of a source term step—which reduces one or two sums—turns the positive instantiation of the respective sum locks into negative instantiations to indicate that no further branch of these sums can be used in the emulation of source term steps. As part of such an emulation, the encoding has to check the values of the sum locks of the respective encoded sender and encoded receiver. If both are positive, i.e., if the encoded sender and the encoded receiver belong to two encoded sums that are not yet used within an emulation, the emulation is completed and both sum locks are set to false instantiations. Else, the emulation is aborted.

Analysing the rules Pi-Com$_{\mathrm{m,s}}$ and Pi-Rep$_{\mathrm{m,s}}$ (Figure 2.3) of the reduction semantics of π_{s}, we observe that a communication with a branch of a sum automatically removes the other branches of that sum. The negative instantiations of sum locks disable other branches of a sum, but they do not immediately withdraw these branches. They remain as junk. However, the negative instantiation of the sum lock signalises that such a remainder is junk and, hence, the encoding can ensure that these parts cannot contribute to the abstract behaviour of encoded terms, i.e., are not used in further emulations. To check whether a sum lock is still positive or negative, the encoding has to introduce additional steps, i.e., single source term steps are translated into sequences of steps.

Apart from boolean locks to translate sums, $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ introduces *sender locks* s. They are used to circumvent the first syntactical difference between π_{s} and π_{a}, i.e., to compensate the use of outputs as guards. Instead of an output, the encoded continuation of a sender is guarded by an input on s. If the encoding was able to successfully emulate a step on the respective source term sender, it provides an instantiation of the sender lock, i.e., an output on s. An additional target term step then unguards the encoding of the continuation.

5.1.2. Implementing the Concept

First note that the explanation above does neither talk about how to translate restriction nor parallel composition. In fact, both are simply translated homomorphically in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. The same holds for $\checkmark$, which is not considered in [Nes00] but added here in order to reason about the quality of this encoding within the general framework of Gorla.

The translation of the choice operator

$$\left[\!\!\left[\sum_{i \in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{s}}_{\mathrm{a}} \triangleq (\nu l) \left(\overline{l}\langle \top \rangle \mid \prod_{i \in I} [\![\pi_i.P_i]\!]^{\mathrm{s}}_{\mathrm{a}} \right)$$

introduces a sum lock with a positive instantiation, where $\prod_{i \in I} [\![\pi_i.P_i]\!]^{\mathrm{s}}_{\mathrm{a}}$ denotes the parallel composition of the encodings of all branches $\pi_i.P_i$ in the respective sum. As already mentioned, boolean values as the $\top$ above are not part of the asynchronous pi-calculus. The same holds for the test-constructs used by the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ to check whether an instantiation of a sum lock is positive or negative. Of course, the pi-calculus and also its asynchronous fragment is expressive enough to implement booleans and their tests. Accordingly, [Nes96, Nes00] introduce both as abbreviations of π_{a}-terms, which

ensure that $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ is indeed an encoding into π_{a}. Because these abbreviations significantly improve readability of the encoding, we use them also in the extensions $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ of this encoding in the next sections and do usually not unfold them. Their formal definition is given below.

In π_{s} each branch of a choice is guarded by an output prefix, an input prefix, or by τ. The encoding of output prefixes

$$[\![\, \overline{y}\langle z \rangle \, .P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \triangleq (\nu s) \left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)} \langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z) \rangle \mid s. [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \right)$$

introduces a sender lock s to guard the encoded continuation and augments the source term output $\overline{y}\langle z \rangle$ by additional information about the corresponding sum and sender lock. Note that $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ should be an encoding into π_{a}, which is a monadic calculus. Hence, an output transmitting three values as above is not allowed. However, within the presented encoding functions in the following, we treat the target language π_{a} as if it allows for polyadic communication. More precisely, we allow asynchronous links to carry any number of values from zero to seven, of course under the requirement that within each π_{a}-term no link name is used twice with different multiplicities. Note that these polyadic actions are again just abbreviations of monadic actions, i.e., they can be simply translated by a standard encoding as given in [SW01]. We discuss that issue in Section 5.4. For the moment, we silently use the polyadic versions of asynchronous pi-calculus here and in the following sections.

In contrast to [Nes00], we augment the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ by a renaming policy to reserve the names l, l' for sum locks, s for sender locks, r for receiver locks explained below, and t, f to implement boolean values. So, the renaming policy of this encoding, denoted by $\varphi^{\mathrm{s}}_{\mathrm{a}}$, is some arbitrary injective substitution such that $\forall n \in \mathcal{N} \, . \, \varphi^{\mathrm{s}}_{\mathrm{a}}(n) \cap (N \cup N') = \emptyset$, where $N = \{ \, l, l', s, r, t, f \, \}$ and N' contains some auxiliary names used to unfold the polyadic communications in Section 5.4. Note that the name l appears free in the encoding of guarded terms. It is reserved by the renaming policy, i.e., different from all translations of source term names, and bound by the encoding of a surrounding sum. Since there are no guarded terms beyond sums, in all encoded terms all occurrences of l are restricted. In [Nes00] indices on the encoding functions were used to capture the name l as parameter of the encoding function and, silently, it was required that the names in N are different from all source term names. The renaming policy turns the use of l as parameter superfluous and makes the convention on the distinction between source term names and the names in $N \cup N'$ explicit.

In $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ the receivers take control over the sum locks. If there are a source term sender and receiver on the same channel name, the translated sender sends the names of its sum lock and its sender lock s to the receiver. The receiver then checks for its own and the sum lock of the sender. If both locks are instantiated with true then he instantiates the sender lock and performs its subprocess. Moreover, both sum locks are instantiated with $\bot$ such that no other branch of the respective sums can be used for communication

and the sender lock is instantiated to enable the unguarding of the senders continuation.

$$\begin{aligned}[\![\, y(x).P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \triangleq\ & (\nu r)\,\big(\overline{r} \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(y)\big(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big)\big)\,. \\ & \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\ & \qquad \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\ & \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\big\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big\rangle\,\big)\end{aligned}$$

The *receiver lock* r allows to restart a test on the corresponding receiver if a former test failed due to a negative instantiation of the sender sum lock. To do so, in this case, the receiver lock is reinstantiated. To allow the first test it is initially instantiated. Moreover it blocks the search of a matching output partner for communication to avoid multiple concurrent tests on the same encoded input guarded term. The receiver lock is not reinstantiated in the case of a successful completion of a test nor of a test failing due to a negative instantiation of the sum lock of the receiver because, in these cases, a corresponding test will never succeed any more. Note that in each case the completion of a test either reinstantiates the instantiations of all consumed sum locks or turns them to negative instantiations. This ensures that later tests are possible, i.e., that there is always eventually exactly one instantiation of each sum lock. Of course it is possible that some sender lock is never instantiated. In that case it blocks the continuation of the respective output for ever.

We naturally extend this encoding of guards to translate the τ prefix, which was not considered in [Nes00]. Branches of choice guarded by τ can reduce without a communication partner. Thus, it suffices to check the corresponding sum lock.

The translation of replicated inputs

$$[\![\, y^{\star}(x).P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \triangleq \varphi^{\mathrm{s}}_{\mathrm{a}}(y)^{\star}(l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)).\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle$$

is similar to the translation of input guards. In contrast to them, replicated inputs can be used arbitrarily often. Hence, only the sum lock of the sender is checked.

Note that in the case of an empty sum (i.e., $I = \emptyset$) the encoding yields $(\nu l)\,\big(\overline{l}\langle\top\rangle \mid \mathsf{0}\big)$ which is semantically equivalent to $\mathsf{0}$. With that $\mathsf{0}$ is translated semantically equal to $\mathsf{0}$.

Locks, Booleans, and the test**-construct.** A *lock* is a special channel used by the encoding function to block some further behaviour. Therefore the term we want to block is guarded by an input on the lock channel such that the term is blocked until an output on this channel is available. In the encoding presented above there are two kinds of these simple locks: receiver locks, denoted by r, and sender locks, denoted by s. In both cases, the locks carry no values. Hence, they can be implemented simply by channels. The existence or absence of a respective output message on that channel decides whether the term tucked away behind the lock can be unguarded, i.e., can be used to emulate further behaviour.

In contrast, *boolean locks* are channels on which only the boolean values $\top$ (true) or $\bot$ (false) are transmitted. An output over a boolean lock with value $\top$ is called a positive instantiation of the respective lock while sending $\bot$ is denoted as negative instantiation.

At the receiving end of such a channel, the boolean value can be used to make a binary decision, which is done here within a test-*construct*. The test-construct and accordingly positive and negative instantiations of boolean locks are implemented in [Nes96, NP00] using restriction and the order of transmitted values.

Definition 5.1.1 (Booleans and Tests)**.** The test operator and *positive* or *negative instantiations* of boolean locks, denoted by $\overline{l}\langle\top\rangle$ and $\overline{l}\langle\bot\rangle$ for a boolean lock l, are abbreviations of the terms:

$$\begin{aligned}
\overline{l}\langle\top\rangle &\triangleq l(t,f)\,.\overline{t}\\
\overline{l}\langle\bot\rangle &\triangleq l(t,f)\,.\overline{f}\\
\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q &\triangleq (\nu t,f)\left(\overline{l}\langle t,f\rangle \mid t.P \mid f.Q\right) \qquad \text{for some } t,f \notin \mathsf{fn}(P \mid Q)
\end{aligned}$$

We observe that the boolean values $\top$ and $\bot$ are realised by a pair of links without parameters. Both are itself locks. The lock t representing $\top$ guards the then-case P, while the else-case Q is guarded by f representing the value $\bot$. Thus, the test-construct operates as guard for its subterms P and Q. Accordingly, a boolean lock can be considered as a lock with two parameters and the distinction between true and false is given by the order of these parameters. By using the renaming policy (compare to Definition 3.3.2), we can omit the condition on the freshness of t and f in the Definition of test-constructs. The renaming policy $\varphi^{\mathrm{s}}_{\mathrm{a}}$ reserves the names t and f to implement the boolean values $\top$ and $\bot$.

5.1.3. Encoding Example

Finally, let us consider the term

$$S = (\overline{y}\langle z_1\rangle\,.P_1 + \overline{y}\langle z_2\rangle\,.P_2) \mid (y(x)\,.P_3)$$

of the source language π_{s} as an example. It encoding is given by:

$$\begin{aligned}
[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} &= [\![\, \overline{y}\langle z_1\rangle\,.P_1 + \overline{y}\langle z_2\rangle\,.P_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\, y(x)\,.P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\\
&= (\nu l)\left(\overline{l}\langle\top\rangle \mid [\![\, \overline{y}\langle z_1\rangle\,.P_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\, \overline{y}\langle z_2\rangle\,.P_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right) \mid (\nu l)\left(\overline{l}\langle\top\rangle \mid [\![\, y(x)\,.P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\\
&= (\nu l)\Big(\overline{l}\langle\top\rangle \mid (\nu s)\left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l,s,\varphi^{\mathrm{s}}_{\mathrm{a}}(z_1)\rangle \mid s.[\![\, P_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\\
&\qquad\qquad \mid (\nu s)\left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l,s,\varphi^{\mathrm{s}}_{\mathrm{a}}(z_2)\rangle \mid s.[\![\, P_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\Big)\\
&\quad \mid (\nu l)\left(\overline{l}\langle\top\rangle \mid (\nu r)\left(\overline{r} \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(y)\big(l',s,\varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big)\,.\right.\right.\\
&\qquad\qquad \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\\
&\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r}\\
&\qquad\qquad \left.\left.\mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\big\langle l',s,\varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big\rangle\right)\right)
\end{aligned}$$

Without loss of generality let us assume that $\varphi^{\mathrm{s}}_{\mathrm{a}}(y) = y$, $\varphi^{\mathrm{s}}_{\mathrm{a}}(z_1) = z_1$, $\varphi^{\mathrm{s}}_{\mathrm{a}}(z_2) = z_2$, and $\varphi^{\mathrm{s}}_{\mathrm{a}}(x) = x$. After a preprocessing step on r there are two possible steps on y, one for

each such step in the source term. Note that both steps require a scope extrusion of the sum locks. Hence, the term

$$
\begin{aligned}
(\nu l_1, s_1, s_2, l_2, r)\Big(\, & \overline{l_1}\langle\top\rangle \mid s_1.[\![\, P_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \overline{y}\langle l_1, s_2, z_2\rangle \mid s_2.[\![\, P_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \mid \overline{l_2}\langle\top\rangle \mid r^{\star}.y\big(l', s, x\big)\,.\mathsf{test}\ l_2\ \mathsf{then\ test}\ l'\ \mathsf{then}\ \overline{l_2}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_2}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
& \qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\bot\rangle \mid \overline{y}\big\langle l', s, x\big\rangle \\
& \mid \mathsf{test}\ l_2\ \mathsf{then\ test}\ l_1\ \mathsf{then}\ \overline{l_2}\langle\bot\rangle \mid \overline{l_1}\langle\bot\rangle \mid \overline{s_1} \mid \{\, {}^{z_1}\!/_{x} \,\}\, [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\top\rangle \mid \overline{l_1}\langle\bot\rangle \mid \overline{r} \\
& \qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\bot\rangle \mid \overline{y}\langle l_1, s_1, x\rangle\Big)
\end{aligned}
$$

is one possible result of performing two steps in $[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}$. Now the unguarded nested test-construct can be reduced to its then-case

$$\overline{l_2}\langle\bot\rangle \mid \overline{l_1}\langle\bot\rangle \mid \overline{s_1} \mid \{\, {}^{z_1}\!/_{x} \,\}\, [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\,,$$

because both sum locks are instantiated with $\top$. As result both sum locks become false, an instantiation of the first sender lock becomes available, and the continuation of the receiver is unguarded. An additional step also unguards the continuation of the sender:

$$
\begin{aligned}
(\nu l_1, s_1, s_2, l_2, r)\Big(\, & [\![\, P_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \overline{y}\langle l_1, s_2, z_2\rangle \mid s_2.[\![\, P_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \mid r^{\star}.y\big(l', s, x\big)\,.\mathsf{test}\ l_2\ \mathsf{then\ test}\ l'\ \mathsf{then}\ \overline{l_2}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
& \qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\bot\rangle \mid \overline{y}\big\langle l', s, x\big\rangle \\
& \mid \overline{l_2}\langle\bot\rangle \mid \overline{l_1}\langle\bot\rangle \mid \{\, {}^{z_1}\!/_{x} \,\}\, [\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}\Big)
\end{aligned}
$$

We observe that the continuation of the second sender stays guarded forever, because the corresponding sum lock l_1 is instantiated by $\bot$. Moreover, note that in the term above everything except $[\![\, P_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\, P_3 \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is junk. This shows that junk or garbage is often a serious problem of encoding functions. In Section 6.3.5 we discuss different variants of junk and prove that $[\![\, \cdot \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is such that all junks it introduces do not contribute to the behaviour of target terms modulo a not-trivial equivalence. Thus, the junk introduced by $[\![\, \cdot \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ does no harm.

5.2. Extending Encodings

Of course it is much easier to design a new encoding function on top of an existing one. This not only allows to reuse parts of the encoding functions or abbreviations improving the presentation of an encoding function, but also to benefit from the solutions found by the former encoding for some problems in its development and, hopefully, also to reuse parts of the argumentation of its correctness. There are usually two reasons to extend

an encoding: either to change the set of its properties, i.e., satisfied quality criteria, or to extend it in order to capture a more expressive source language or a less expressive target language. Here we extend the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ discussed in the last section to an encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ that instead of separate choice encodes mixed choice, i.e., we extend the encoding in order to capture the more expressive source language π_{m}.

To do so, we extend first the concept of the respective encoding, to deal with the problems that arise from the new source or target language. We discuss in Section 5.2.1 in what sense mixed choice is more difficult than separate choice and how to overcome these problems. Unfortunately our solution leads to a very huge encoding function. Because of that, we introduce it in two steps. In Section 5.2.2 we use the expressive power of polyadic synchronisation to introduce an intermediate encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ of mixed choice. The presented encoding is a good encoding from π_{m} (without replication) into π_{p}. Moreover, it preserves distributability and is thus interesting on its own. On the other side, polyadic synchronisation is a very powerful synchronisation mechanism as already explained in [CM03]. Hence, π_{p} is strictly more expressive than π_{a}. However, in Section 5.2.4 we show that the full power of polyadic synchronisation is not necessary to encode mixed choice, by refining the intermediate encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ into the final encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is a good encoding of mixed choice with respect to the quality criteria of the general framework in Section 3.3. But it is an encoding from π_{m} into $\pi^{=}_{\mathrm{a}}$, i.e., it requires the match prefix in the target language. Unfortunately, we were not able to avoid the use of the match prefix and, indeed, there are good reasons to believe that it is not possible to encode mixed choice without the power of matching. We shortly discuss this problem at the beginning of Section 5.2.4. In the Sections 5.2.3 and 5.2.5 we present an example for each encoding function.

5.2.1. Extending the Concept

Sometimes extending an encoding in order to capture a more expressive variant of the source language is fairly easy. We can for instance extend our source and target language with the match prefix. To obtain an encoding from $\pi^{=}_{\mathrm{s}}$ into $\pi^{=}_{\mathrm{a}}$, it suffices to extend the encoding in Figure 5.1 with the homomorphic translation of the match prefix:

$$[\![\, [a = b]\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \triangleq [\varphi^{\mathrm{s}}_{\mathrm{a}}(a) = \varphi^{\mathrm{s}}_{\mathrm{a}}(b)]\, [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}$$

The match prefix and thus its translation does not interfere with the protocol implemented by the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. The step from separate choice to mixed choice is not that easy. As explained in [Nes00] the presence of mixed choice or, more precisely, cyclic dependencies within a single mixed choice or a set of mixed choices, leads to deadlocks in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. That is why this encoding is a good encoding from π_{s} into π_{a}, but not a good encoding from π_{m} into π_{a}. We explain the problem using two examples.

Example 5.2.1 (Incestuous sum)**.** Consider the sum $\overline{y}\langle z\rangle + y(x)$. It is called an *incestuous sum* because it contains two potential communication partners, i.e., there is an

output and a matching input within the same sum. Its encoding

$$
\begin{aligned}
(\nu l)\,\big(\;&\overline{l}\langle\top\rangle \mid (\nu s)\,(\overline{y}\langle l,s,z\rangle \mid s.\,[\![\,0\,]\!]^{\mathrm{s}}_{\mathrm{a}}) \\
&\mid (\nu r)\,\big(\overline{r} \mid r^{\star}.y(l',s,x)\,.\mathsf{test}\ l\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\,0\,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{y}\big\langle l',s,x\big\rangle\,)\big)
\end{aligned}
$$

deadlocks while performing the nested test-construct because it tries to check twice for the same lock, i.e., the first part of the nested test-construct consumes the sum lock instantiation and so the second part—which tests for the same lock—is deadlocked. Since the source term $\overline{y}\langle z\rangle + y(x)$ cannot perform a step as well this is not a problem. But consider the term $P = \overline{y}\langle z\rangle + y(x) \mid y(x)\,.Q$. It reduces to $Q\,\{\,{}^{z}\!/_{x}\,\}$. In this case the deadlock which may occur by first testing the communication within the incestuous sum leads to different behaviour of the target term, i.e., the target term may deadlock without reaching the encoding of $Q\,\{\,{}^{z}\!/_{x}\,\}$, while the source term reaches $Q\,\{\,{}^{z}\!/_{x}\,\}$ in every maximal execution (w.r.t. reduction semantics).

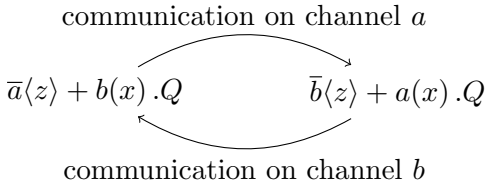


Figure 5.2.: Cyclic sums.

Example 5.2.2 (Cyclic sums)**.** With *cyclic sums* we denote a set of sums with cyclic dependencies of their potential communication partners as in $P = \overline{a}\langle z\rangle + b(x)\,.Q \mid \overline{b}\langle z\rangle + a(x)\,.Q$. The cyclic dependencies of P are depicted in Figure 5.2. Obviously P can reduce by a communication either on channel a or on channel b. The encoding of P

$$
\begin{aligned}
&(\nu l_1)\,\big(\overline{l_1}\langle\top\rangle \mid (\nu s_1)\,(\overline{a}\langle l_1,s_1,z\rangle \mid s_1.\,[\![\,0\,]\!]^{\mathrm{s}}_{\mathrm{a}}) \\
&\quad\mid (\nu r_1)\,\big(\overline{r_1} \mid r_1^{\star}.b\big(l',s,x\big)\,.\mathsf{test}\ l_1\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\,Q\,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r_1} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{b}\big\langle l',s,x\big\rangle\,)\big) \\
&\mid (\nu l_2)\,\big(\overline{l_2}\langle\top\rangle \mid (\nu s_2)\,\big(\overline{b}\langle l_2,s_2,z\rangle \mid s_2.\,[\![\,0\,]\!]^{\mathrm{s}}_{\mathrm{a}}\big) \\
&\quad\mid (\nu r_2)\,\big(\overline{r_2} \mid r_2^{\star}.a\big(l',s,x\big)\,.\mathsf{test}\ l_2\ \ \mathsf{then\ test}\ l'\ \ \mathsf{then}\ \overline{l_2}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\,Q\,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r_2} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_2}\langle\bot\rangle \mid \overline{a}\big\langle l',s,x\big\rangle\,)\big)
\end{aligned}
$$

will deadlock if the two nested test-constructs are performed simultaneously, i.e., if the first nested test-construct consumes the lock l_1 and before its second part is performed the

second nested test-construct tests the lock l_2. In this situation the process is deadlocked, because both instantiations of sum locks are consumed and so none of the remaining test-constructs can be resolved. Again the target term may deadlock without reaching the encoding of $Q\{\,{}^{z}/_{x}\,\}$, while the source term reaches $Q\{\,{}^{z}/_{x}\,\}$ in every maximal execution, i.e., there is a difference in the behaviour of the target and the source term.

Both cases result in a deadlock that is induced by the encoding function and not intended by the underlying source term. Note that such deadlocks are detected by operational soundness provided $\asymp$ is not trivial. In other words, not dealing with this problem adequately violates the operational correspondence criterion in Definition 3.3.4. In [Nes00] different attempts to overcome these deadlocks are discussed. The simplest way to resolve them is to implement the possibility to roll back a test. Unfortunately this directly leads to divergence introduced by the encoding and thus violates the divergence reflection criterion in Definition 3.3.5. Another attempt is to assume a total ordering among the threads or processes of the system, such that within the test-constructs the order of the locks tested can be determined by the order of the according threads. Since we have no such total ordering on the source terms, it has to be constructed by the encoding function. That can be done for instance by a two-level encoding or by an encoding with global knowledge about the source term. Both solutions violate the compositionality criterion in Definition 3.3.1. A third attempt is to choose the order of the locks tested at random. Again this violates—depending on the implementation—either the operational correspondence criterion or the divergence reflection criterion although deadlock or divergence may occur only with a very low probability. This approach was formally investigated in [HP05] in the context of the probabilistic pi-calculus. Nevertheless, as we will show in the following, there is a way to circumvent both problems, incestuous and cyclic sums, without referring to randomization within the framework of a good encoding of Gorla presented in Section 3.3.

To be precise, we present two different ideas to extend the encoding $[\![\,\cdot\,]\!]^{\mathrm{s}}_{\mathrm{a}}$ to cover mixed choice: one restricting the number of tests performed simultaneously (implemented in Section 5.3) and the other implementing an algorithm to compute an appropriate ordering of the sum locks during executions. The main trick of both ideas is to make use of the structure of source terms induced by the nesting of parallel operators. This is motivated by another problem of an encoding between π_{m} and (π_{s} or) π_{a}. As shown in Chapter 4, π_{m} and π_{s} differ not only for the existence of cyclic dependencies within sums. A more fundamental difference is that π_{m} can break initial symmetries, whereas this is impossible in π_{s} and its subcalculus π_{a}. Moreover, Section 4.2.1 shows that this difference leads to a separation result concerning encodings that translate the parallel operator homomorphically. We conclude from these considerations that an encoding between π_{m} and π_{a} cannot translate the parallel operator homomorphically and that it has to be the encoding function that breaks potential source term symmetries, because the target language is not able to do so. In Section 5.2.4 and Section 5.3, we propose two single-level encodings, in which the *symmetry is broken locally* at each parallel operator, while still allowing for an unconstrained composability of encoded terms. By doing so, we also avoid the problem with cyclic dependencies in sums. To explain

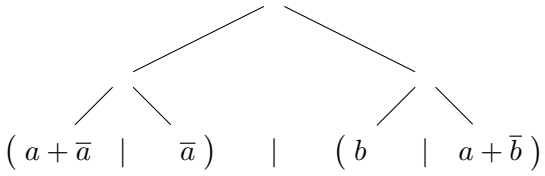


Figure 5.3.: Parallel structure.

the main idea of the encodings, let us consider the source term

$$S = (a + \overline{a} \mid \overline{a}) \mid (b \mid a + \overline{b})$$

and its *parallel structure* depicted in Figure 5.3.

As parallel structure, we denote the binary tree induced by the nesting of the parallel operators of a term, where the leafs are formed by its capabilities. Analysing the operational semantics of π_{m}, given by Figure 2.3, we observe that communication steps in π_{m} are always due to an input and an output guarded term on two different sides of a node within the parallel structure of a term. Moreover, we observe that each matching pair of communication partners is left and right of exactly one node of that binary tree, i.e., their closest common parent node. S, for instance, can perform a step on channel b by reducing the last two sums that meet at the right-most parallel operator, or it can perform a step on channel a e.g. by reducing the first and the last sum that are left and right to the outermost parallel operator, i.e., the root of the tree. Since there is no choice operator in π_{a}, its encoding forces us to represent its branches as parallel terms; otherwise there would be no way for them to be concurrently enabled. Obviously, and unfortunately, this changes the parallel structure of the original term. The sum lock is introduced to restore the lost information about the correspondence of branches to a sum. That suffices to encode separate choice, but as shown in the Examples 5.2.1 and 5.2.2 above it does not suffice to encode mixed choice. The main idea to overcome these problems is to exploit the parallel structure of the originating source term when enabling or guiding interactions in its translation at the target level. More precisely: (1) to avoid the problem of incestuous sums, the encodings will guarantee that emulations of source term steps will only be possible for senders and receivers emanating from two different sides of a parallel operator in the source term; (2) to avoid the problem of cyclic sums, the encoding either restricts the number of test-constructs being concurrently enabled at the level of the same parallel operator, i.e., at the same node in the parallel structure of the source term, or it orders the sum locks according to their origin in the parallel structure, by testing the sum lock of the leftmost leaf first. Both solutions break source term symmetries locally within each parallel operator encoding.

In essence, the detection of matching communication partners is ceded to the nodes of the parallel structure of the source term. More precisely, each parent node, i.e., each parallel operator encoding, is equipped with a protocol to process and guide communication *requests* from its child nodes. This is explicitly allowed by weak compositionality in contrast to homomorphic translations, and it is here that the source-term-level symmetry is

broken, because the protocol—in both encodings—handles requests from the left child and requests from the right child slightly differently. Now, as opposed to the previous globally-breaking proposals by Nestmann [Nes00], the overall exercise here is much more difficult: from the point of view of a single parallel operator encoding, communication may not only occur between its left- and right-hand subterms, but also between either of these two and some outer—unknown—communication partner in the environment. All guiding protocols residing at the various nodes in the binary parallelism-reflecting tree must play together, and the encoding function must treat them all alike (to avoid the term "symmetrically") to keep the encoding truly compositional. To this aim, the encoding of input and output capabilities announces their ability to send or receive in form of *requests* along special channels to their parent nodes. If, at the level of a node, a matching pair of communication partners is identified, the (nested) test-constructs are checked as described in the encoding of Figure 5.1. At the same time, to keep up with communication possibilities with "external" partners, the requests are passed on to the potential parent of this node.

As it turns out, guiding the flow of requests and implementing a not divergent protocol to find communication partners at the level of a parallel operator encoding awfully blows up the encoding function. Because of that, we explain first the flow of requests within the parallel structure of source terms on an intermediate encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and postpone the implementation of a divergence free algorithm to combine requests within a node to the final encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Hence, the intermediate encoding is presented in order to ease the explanation of the final encoding.

To shorten the presentation of the encodings a little bit, we define forwarders. A forwarder is a simple process that forwards each received message along some specified set of links.

Definition 5.2.3 (Forwarder)**.** Let I be a finite index set and for all $i \in I$ let y and y_i be channel names with the same multiplicity $n \in \mathbb{N}$, then a *forwarder* is given by:

$$y \twoheadrightarrow \{\, y_i \mid i \in I \,\} \triangleq y^{\star}(x_1, \ldots, x_n) \,.\, \left(\prod_{i \in I} \overline{y_i}\langle x_1, \ldots, x_n \rangle \right)$$

In case of a singleton set we omit the brackets, i.e., $y \twoheadrightarrow y' \triangleq y \twoheadrightarrow \{\, y' \,\}$.

5.2.2. An Intermediate Encoding

First in [Pal03], and later as well in [Gor10b], and by us in Section 4.2.1, it is proved that there is no encoding from π_{m} into π_{s} and, thus, no encoding from π_{m} into π_{a} that translates the parallel operator homomorphically. So we have to abandon this condition. As a natural consequence we implement the above described idea within the translation of the parallel operator. Unfortunately this significantly blows up the encoding, especially of the parallel operator and replicated input.

Because of that, we introduce the encoding in two steps, i.e., we present the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ as an intermediate step. Instead of π_{a} the target language of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is the strictly

more expressive asynchronous pi-calculus augmented with polyadic (here: 2-adic) synchronisation, denoted as π_{p}, as proposed by Carbone and Maffeis [CM03]. Note that in contrast to π_{a} the calculus π_{p} has the power to implement the match prefix as shown in [CM03]. This is important to obtain an encoding of mixed choice as we will see in Section 5.2.4. The use of an intermediate encoding allows us to focus on the essence of the encoding. As we use the polyadic synchronisation only in a limited manner, its usage can be expanded into the standard target calculus with match and is thus not critical for the intended result. Moreover, we omit the encoding of replicated inputs in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, because its implementation requires to combine the translation of an input prefix and the parallel operator. Thus, the translation of replicated inputs leads to the largest target terms, although it does not really require new ideas or concepts compared to the encoding of inputs or parallel composition. We postpone it to the final encoding in Section 5.2.4. So, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is an encoding of π_{m} without replicated input into π_{p}.

For sums, the translation via $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ follows exactly the scheme of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$.

$$\left[\!\!\left[\sum_{i\in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{m}}_{\mathrm{p}} \triangleq (\nu l)\left(\overline{l}\langle\top\rangle \mid \prod_{i\in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{p}} \right)$$

As described above, this translation splits up the encoded branches in parallel and introduces the sum locks, which are initialised by $\top$. To *order* these sum locks, we first have to transport them to a surrounding parallel operator encoding: for example, in $P \mid Q$, with P and Q being sequential processes, the sums occurring in either P or Q will have their locks ordered by means of the translation $[\![P \mid Q]\!]^{\mathrm{m}}_{\mathrm{p}}$. Therefore, in the translation, we let input- and output-guarded source terms not communicate directly, but instead require that they first register their send/receive abilities to a surrounding parallel operator encoding, by sending an *output request* $\overline{p_o}\langle y, l, s_1, s_2, z\rangle$ or an *input request* $\overline{p_i}\langle y, l, r_1, r_2\rangle$. A request carries all necessary information to resolve a (nested) test-construct, i.e., the translated link name, the corresponding sum lock, the sender or receiver locks, and, in case of an output request, the translation of the transmitted value. Note that a sender lock, i.e., the s_2 in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, is used to guard the encoded continuation of the sender, while over the receiver lock, i.e., the r_2 in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, the ordered sum locks are transmitted back to the receiver. By the renaming policy $\varphi^{\mathrm{m}}_{\mathrm{p}}$, the mapping $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is implicitly parametrised by the names p_o and p_i. Some of their occurrences are bound, while others—the outermost—remain free.

$$\begin{aligned}
[\![\, \tau.P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle \\
[\![\, \overline{y}\langle z\rangle.P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu s_1, s_2)\left(\overline{s_1} \mid s_1^{\star}.\overline{p_o}\langle \varphi^{\mathrm{m}}_{\mathrm{p}}(y), l, s_1, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(z)\rangle \mid s_2.[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \right) \\
[\![\, y(x).P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu r_1, r_2)\left(\overline{r_1} \mid r_1^{\star}.\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{p}}(y), l, r_1, r_2\rangle \mid r_2^{\star}\big(l_1, l_2, -, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(x), v, w\big).\right. \\
&\qquad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v} \\
&\qquad \left.\mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\right)
\end{aligned}$$

Apart from requests, the encoding of guarded terms is very similar to $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, the encoding of terms guarded by τ is even the same. The sender and the receiver lock are split into two parts. The respective first parts s_1 and r_1 ensure that the origin of a request in the parallel structure is always the same, in case it has to be retransmitted because of an aborted test-construct. The second sender lock s_2 is similar to the sender lock in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. The nested test-construct of the receiver tests the sum locks l_1 and l_2 in the order in which they are received on r_2. Moreover, the locks s_1 and r_1 are transmitted on r_2 as the values v and w ordered matching to their corresponding sum locks. In case the test-construct fails due to a negative instantiation of a sum lock the lock guarding the respective other out- or input request is instantiated.

The requests push the task of finding source term communication partners to the surrounding parallel operator encodings. There, a strict policy controls the redirection of requests. First, it restricts the request channels p_o and p_i for both of its parameters to be able to distinguish requests from the left from those from the right hand side.

$$\begin{aligned}[\![P \mid Q]\!]^{\mathrm{m}}_{\mathrm{p}} \triangleq\ & (\nu p_{o,up}, p_{i,up}, o, i)\,\big(\\ & (\nu p_o, p_i)\,\big([\![P]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big) \\ & \mid (\nu p_o, p_i)\,\big([\![Q]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big) \\ & \mid \mathsf{pushReq}\big)\end{aligned}$$

Within the terms procLeftOutReq, procLeftInReq, procRightOutReq, and procRightInReq communication partners, that meet at this node in the parallel structure, are identified and the test of the corresponding sum locks is induced. To enable the identification of communication partners in other parts of the tree, each parallel operator encoding pushes all received (left or right) requests further upwards to a surrounding parallel operator encoding by means of the forwarders in $\mathsf{pushReq} \triangleq p_{o,up} \twoheadrightarrow p_o \mid p_{i,up} \twoheadrightarrow p_i$.

Requests from the left are forwarded to the links $p_{o,up}$ or $p_{i,up}$, to be pushed further upwards with pushReq. Moreover, in order to combine requests from the left with requests from the right side, all left requests are transmuted into outputs on the channel $y \cdot o$ for output requests and $y \cdot i$ for input requests, where y is the translation of the original channel name corresponding to the respective source term sender or receiver.

$$\begin{aligned}\mathsf{procLeftOutReq} &\triangleq p_o{}^{\star}(y, l, s_1, s_2, z)\,.\,(\overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle) \\ \mathsf{procLeftOutReq} &\triangleq p_i{}^{\star}(y, l, r_1, r_2)\,.\,\big(\overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle\big)\end{aligned}$$

Here we use polyadic synchronisation to transfer the information whether the respective requests result from a source term sender or receiver into the channel name. This allows to transmute both, output and input requests, into output messages. The translated channel name, represented by y, is necessary to ensure that communication is emulated only on matching communication partners, i.e., on sender and receiver sharing the same channel. As a special feature of polyadic synchronisation, the restriction on o and i in the encoding of the parallel operator ensures that, for all y, the combined channels $y \cdot o$ and $y \cdot i$ are fresh for each node in the parallel structure. In fact, this feature is the main

reason for the use of polyadic synchronisation, because it allows us to restrict the search for communication partners within the encoding of parallel operators.

At the right hand side, output and input requests are transmuted into inputs: $y \cdot i$ for right output requests and $y \cdot o$ for right input requests. Hence, a communication on a channel $y \cdot o$ or $y \cdot i$ reveals a pair of communication partners. In this case the information necessary to resolve the respective test-construct are retransmitted over the receiver lock r_2 back to the receiver.

$$\begin{aligned}
\mathsf{procRightOutReq} &\triangleq p_o{}^\star(y, l_s, s_1, s_2, z)\,. \\
&\qquad (y \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle) \\
\mathsf{procRightInReq} &\triangleq p_i{}^\star(y, l_r, r_1, r_2)\,. \\
&\qquad (y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{i,up}}\langle y, l_r, r_1, r_2\rangle)
\end{aligned}$$

To avoid deadlock the sum lock of the left request is always checked first, i.e., the sum lock of the left request is transmitted as first and the sum lock of the right request as second parameter. Since the encoding relies on the parallel structure of the source term, which is a binary tree, to prefer always the left lock indeed results in a total ordering of the sum locks. The third parameter identifies always the sender lock, which is checked in the translation of replicated inputs. Then, there are the second sender lock and the translation of the sent value. The last two parameters are the first parts of the sender and receiver locks. They are ordered with respect to their corresponding sum locks in the first two parameters. Then all right requests are pushed upwards with pushReq.

Finally restriction and success are translated homomorphically:

$$\begin{aligned}
[\![\, (\nu x)\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \left(\nu \varphi^{\mathrm{m}}_{\mathrm{p}}(x)\right) [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \\
[\![\, \checkmark \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \checkmark
\end{aligned}$$

We observe that the encoding function introduces twenty-two different names, covered in the set $N = \{\, p_o, p_i, p_{o,up}, p_{i,up}, l, l_s, l_r, l_1, l_2, s_1, s_2, r_1, r_2, o, i, v, w, x, y, z, t, f \,\}$. Moreover, some more names will be necessary to unfold the polyadic communications into monadic communications in Section 5.4. The renaming policy $\varphi^{\mathrm{m}}_{\mathrm{p}}$ ensures that there are no clashes between these names and the names of the source terms. To achieve this, $\varphi^{\mathrm{m}}_{\mathrm{p}}$ can be every injective substitution such that, for all $n \in \mathcal{N}$, $\varphi^{\mathrm{m}}_{\mathrm{p}}(n)$ is neither in N nor one of the names reserved in Section 5.4. The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is given by the Figures 5.4 and 5.5.

5.2.3. Encoding Example

To illustrate the encoding and the emulation of source term steps, we consider

$$S = a(z_1)\,.\mathbf{0} + \overline{a}\langle z_2\rangle\,.\mathbf{0} \mid a(z_3)\,.\mathbf{0} + \overline{a}\langle z_4\rangle\,.\mathbf{0}$$

as an example. Note that this is a version of the term $a + \overline{a} \mid a + \overline{a}$ used in the Proof of Theorem 4.2.10, now without abbreviations, i.e., without omitting unnecessary

$$
\begin{aligned}
[\![\, (\nu x)\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \big(\nu \varphi^{\mathrm{m}}_{\mathrm{p}}(x)\big)\, [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \\
[\![\, P \mid Q \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu p_{o,up}, p_{i,up}, o, i)\, \big(\\
&\qquad (\nu p_o, p_i)\, \big([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \big) \\
&\qquad \mid (\nu p_o, p_i)\, \big([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq} \big) \\
&\qquad \mid \mathsf{pushReq} \big) \\
\Big[\!\Big[\sum_{i \in I} \pi_i.P_i \Big]\!\Big]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu l) \left(\overline{l}\langle \top \rangle \mid \prod_{i \in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{p}} \right) \\
[\![\, \tau.P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle \bot \rangle \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}}\ \mathsf{else}\ \overline{l}\langle \bot \rangle \\
[\![\, \overline{y}\langle z \rangle .P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu s_1, s_2) \left(\overline{s_1} \mid s_1^{\star}.\overline{p_o}\big\langle \varphi^{\mathrm{m}}_{\mathrm{p}}(y)\,, l, s_1, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(z) \big\rangle \mid s_2.\, [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \right) \\
[\![\, y(x) .P \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq (\nu r_1, r_2)\, \big(\overline{r_1} \mid r_1^{\star}.\overline{p_i}\big\langle \varphi^{\mathrm{m}}_{\mathrm{p}}(y)\,, l, r_1, r_2 \big\rangle \\
&\qquad \mid r_2^{\star}\big(l_1, l_2, -, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(x)\,, v, w\big)\,. \\
&\qquad\quad \mathsf{test}\ l_1\ \ \mathsf{then}\ \mathsf{test}\ l_2\ \ \mathsf{then}\ \overline{l_1}\langle \bot \rangle \mid \overline{l_2}\langle \bot \rangle \mid \overline{s_2} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \\
&\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \overline{l_1}\langle \top \rangle \mid \overline{l_2}\langle \bot \rangle \mid \overline{v} \\
&\qquad\qquad\ \ \mathsf{else}\ \overline{l_1}\langle \bot \rangle \mid \overline{w} \big) \\
[\![\, \checkmark \,]\!]^{\mathrm{m}}_{\mathrm{p}} &\triangleq \checkmark
\end{aligned}
$$

Figure 5.4.: An Encoding from π_{m} without replication into π_{p}.

$$
\begin{aligned}
\mathsf{procLeftOutReq} &\triangleq p_o{}^{\star}(y, l, s_1, s_2, z)\,. \big(\overline{y \cdot o}\langle l, s_1, s_2, z \rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z \rangle\big) \\
\mathsf{procLeftOutReq} &\triangleq p_i{}^{\star}(y, l, r_1, r_2)\,. \big(\overline{y \cdot i}\langle l, r_1, r_2 \rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2 \rangle\big) \\
\mathsf{procRightOutReq} &\triangleq p_o{}^{\star}(y, l_s, s_1, s_2, z)\,. \\
&\qquad (y \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1 \rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z \rangle) \\
\mathsf{procRightInReq} &\triangleq p_i{}^{\star}(y, l_r, r_1, r_2)\,. \\
&\qquad (y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1 \rangle \mid \overline{p_{i,up}}\langle y, l_r, r_1, r_2 \rangle) \\
\mathsf{pushReq} &\triangleq p_{o,up} \twoheadrightarrow p_o \mid p_{i,up} \twoheadrightarrow p_i
\end{aligned}
$$

Figure 5.5.: Auxiliary Functions of $[\![\, \cdot \,]\!]^{\mathrm{m}}_{\mathrm{p}}$.

parameters and trailing $\mathsf{0}$'s. Since a, z_1, z_2, z_3, and z_4 are no names reserved for the encoding function we can assume without loss of generality that $\varphi^{\mathrm{m}}_{\mathrm{p}}(a) = a$, $\varphi^{\mathrm{m}}_{\mathrm{p}}(z_1) = z_1$, $\varphi^{\mathrm{m}}_{\mathrm{p}}(z_2) = z_2$, $\varphi^{\mathrm{m}}_{\mathrm{p}}(z_3) = z_3$, and $\varphi^{\mathrm{m}}_{\mathrm{p}}(z_4) = z_4$. The corresponding target term $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{p}}$ is given by the term in Figure 5.6.

$$
\begin{array}{l}
(\nu p_{i,up}, p_{o,up}, o, i)\,\big(\\
\quad (\nu p_i, p_o)\,\big(\,(\nu l)\,\big(\overline{l}\langle\top\rangle \\
\qquad \mid (\nu r_1, r_2)\,\big(\,\overline{r_1} \mid r_1^{\star}.\overline{p_i}\langle a, l, r_1, r_2\rangle \mid r_2^{\star}(l_1, l_2, l_3, s_2, z_1, v, w)\,. \\
\qquad\qquad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid (\boldsymbol{\nu l})\,\big(\overline{\boldsymbol{l}}\langle\top\rangle \mid \mathbf{0}\big) \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v} \\
\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\big) \\
\qquad \mid (\nu s_1, s_2)\,\big(\overline{s_1} \mid s_1^{\star}.\overline{p_o}\langle a, l, s_1, s_2, z_2\rangle \mid s_2.\,(\boldsymbol{\nu l})\,\big(\overline{\boldsymbol{l}}\langle\top\rangle \mid \mathbf{0}\big)\big)\,\big) \\
\qquad \mid p_o{}^{\star}(y, l, s_1, s_2, z)\,.\,\big(\overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle\big) \\
\qquad \mid p_i{}^{\star}(y, l, r_1, r_2)\,.\,\big(\overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle\big)\,\big) \\
\quad \mid (\nu p_i, p_o)\,\big(\,(\nu l)\,\big(\overline{l}\langle\top\rangle \\
\qquad \mid (\nu r_1, r_2)\,\big(\,\overline{r_1} \mid r_1^{\star}.\overline{p_i}\langle a, l, r_1, r_2\rangle \mid r_2^{\star}(l_1, l_2, l_3, s_2, z_3, v, w)\,. \\
\qquad\qquad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid (\boldsymbol{\nu l})\,\big(\overline{\boldsymbol{l}}\langle\top\rangle \mid \mathbf{0}\big) \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v} \\
\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\big) \\
\qquad \mid (\nu s_1, s_2)\,\big(\overline{s_1} \mid s_1^{\star}.\overline{p_o}\langle a, l, s_1, s_2, z_4\rangle \mid s_2.\,(\boldsymbol{\nu l})\,\big(\overline{\boldsymbol{l}}\langle\top\rangle \mid \mathbf{0}\big)\big)\,\big) \\
\qquad \mid p_o{}^{\star}(y, l_s, s_1, s_2, z)\,.\,(y \cdot i(l_r, r_1, r_2)\,.\overline{r}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle) \\
\qquad \mid p_i{}^{\star}(y, l_r, r_1, r_2)\,.\,(y \cdot o(l_s, s_1, s_2, z)\,.\overline{r}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{i,up}}\langle y, l_r, r_1, r_2\rangle)\big) \\
\quad \mid p_{o,up}{}^{\star}(y, l, s_1, s_2, z)\,.\overline{p_o}\langle y, l, s_1, s_2, z\rangle \mid p_{i,up}{}^{\star}(y, l, r_1, r_2)\,.\overline{p_i}\langle y, l, r_1, r_2\rangle\,\big)
\end{array}
$$

Figure 5.6.: Encoding Example for $[\![\, \cdot \,]\!]^{\mathrm{m}}_{\mathrm{p}}$.

First, we observe that the encoding of $\mathsf{0}$—bold-faced in S and $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{p}}$—is simply $(\nu l)\,\big(\overline{l}\langle\top\rangle \mid \mathsf{0}\big)$ which is semantically equal to $\mathsf{0}$, because we have $(\nu l)\,\big(\overline{l}\langle\top\rangle \mid \mathsf{0}\big) \not\longrightarrow$. Moreover, we observe that although the source term is a symmetric network of degree 2 (with respect to identity) the resulting target term is not. Note that the source term symmetry is broken because of the different encodings of the left and the right hand side of the parallel operator and not by changing the degree of the source network. To restore the degree of the original network, it suffices to duplicate the last line of the encoding of the parallel operator and assign one instance of it to each side of the encoding of the parallel operator within different scopes of the names $p_{i,up}$ and $p_{o,up}$. We observe that there are initially four requests within the target term; one for each input or output capability of the source term. Moreover, since the corresponding capabilities are unguarded in S, the requests are unguarded in $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{p}}$ or can become so by a single target term step on the respective first sender lock.

Apart from the requests and sender locks, there are two more unguarded outputs. Both are the positive instantiations $\overline{l}\langle\top\rangle$ of sum locks to which no matching inputs are unguarded. Accordingly, initially there are two steps on sender locks and then four steps, one for each request.

First we take a look on the left hand side of the encoding of the parallel operator. The consumption of the left requests leads to:

$$
\begin{aligned}
&(\nu p_i, p_o, \boldsymbol{l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}})\,\big(\,\overline{l_{\leftarrow}}\langle\top\rangle \mid r^{\star}_{1,\leftarrow}.\overline{p_i}\langle a, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle\\
&\quad \mid r^{\star}_{2,\leftarrow}(l_1, l_2, l_3, s_2, z_1, v, w)\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\\
&\quad \mid s^{\star}_{1,\leftarrow}.\overline{p_o}\langle a, l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle \mid s_{2,\leftarrow}.[\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\quad \mid {p_o}^{\star}(y, l, s_1, s_2, z)\,.\,(\overline{y\cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle)\\
&\quad \mid \overline{\boldsymbol{a\cdot o}}\boldsymbol{\langle l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle} \mid \overline{\boldsymbol{p_{o,up}}}\boldsymbol{\langle a, l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle}\\
&\quad \mid {p_i}^{\star}(y, l, r_1, r_2)\,.\,\big(\overline{y\cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle\big)\\
&\quad \mid \overline{\boldsymbol{a\cdot i}}\boldsymbol{\langle l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle} \mid \overline{\boldsymbol{p_{i,up}}}\boldsymbol{\langle a, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle}\,\big)
\end{aligned}
$$

Analysing this term we observe that the requests were not completely consumed, but instead copied into a new version for each request with the same parameters but on different channel names, namely $\overline{p_{o,up}}\langle a, l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle$ and $\overline{p_{i,up}}\langle a, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle$. The purpose of these copies is to push the content of the requests over the restriction on p_i and p_o such that they can be pushed upwards in the parallel structure to enable communications with other parts of the binary tree. Note that the replicated inputs on the links p_i and p_o remain. So some of the requests might be processed at the beginning while other requests might be processed later. That allows us to handle the requests of the encoding of a continuation of some input or output guarded term as soon as the completion of a corresponding source term step removing the respective guard is emulated within the target term. Therefore note that the encodings of continuations of guarded terms, i.e., the $[\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}$ in our case, appear guarded within the encoding of the source term, where the guard is either a receiver lock in case of an input guarded source or a sender lock in case of an output guarded source. Moreover, we observe that these guards cannot be removed by reduction steps on requests. We also observe that the two reduction steps cause a scope extrusion of the restrictions on l, r_1, r_2, s_1 and s_2. Since in the current case there is only one instance of each of these locks no α-conversion is necessary. Multiple receiver/sender locks stem from multiple input/output guarded branches in the respective source term or from the case that at the corresponding side of the parallel operator a subtree of the parallel structure of the source term is encoded which can also lead to multiple sum locks. Later on we combine the requests of the left with the requests of the right in order to emulate a reduction step of the source term. Since on the right hand side there are different versions of these locks, we perform α-conversion to avoid ambiguity later, i.e., we index the locks on the left side by $\leftarrow$ and the locks on the right side by $\rightarrow$.

The processing of the requests on the right side of the encoding of a parallel operator

$$
\begin{aligned}
&(\nu p_i, p_o, \boldsymbol{l_{\rightarrow}}, \boldsymbol{r_{1,\rightarrow}}, \boldsymbol{r_{2,\rightarrow}}, \boldsymbol{s_{1,\rightarrow}}, \boldsymbol{s_{2,\rightarrow}}) \big(\overline{l_{\rightarrow}}\langle\top\rangle \mid r^{\star}_{1,\rightarrow}\overline{p_i}\langle a, l_{\rightarrow}, r_{1,\rightarrow}, r_{2,\rightarrow}\rangle \\
&\quad \mid r^{\star}_{2,\rightarrow}(l_1, l_2, l_3, s_2, z_3, v, w)\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w} \\
&\quad \mid s^{\star}_{1,\rightarrow}.\overline{p_o}\langle a, l_{\rightarrow}, s_{1,\rightarrow}, s_{2,\rightarrow}, z_4\rangle \mid s_{2,\rightarrow}.[\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}} \\
&\quad \mid p_o{}^{\star}(y, l_s, s_1, s_2, z)\,.\,(y \cdot i(l_r, r_1, r_2)\,.\overline{r}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle) \\
&\quad \mid \boldsymbol{a \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_{\rightarrow}, l_{\rightarrow}, s_{2,\rightarrow}, z_4, r_1, s_{1,\rightarrow}\rangle} \mid \boldsymbol{\overline{p_{o,up}}\langle a, l_{\rightarrow}, s_{1,\rightarrow}, s_{2,\rightarrow}, z_4\rangle} \\
&\quad \mid p_i{}^{\star}(y, l_r, r_1, r_2)\,.\,(y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{o,up}}\langle y, l_r, r_1, r_2\rangle) \\
&\quad \mid \boldsymbol{a \cdot o(l_s, s_1, s_2, z)\,.\overline{r_{2,\rightarrow}}\langle l_s, l_{\rightarrow}, l_s, s_2, z, s_1, r_{1,\rightarrow}\rangle} \mid \boldsymbol{\overline{p_{i,up}}\langle a, l_{\rightarrow}, r_{1,\rightarrow}, r_{2,\rightarrow}\rangle}\ \big)
\end{aligned}
$$

is similar. We observe that to enable a test the following information is necessary: the receiver and sum locks of the corresponding encoded input capability, and the sum lock, the sender locks, and the sent value of the corresponding encoded output capability. The requests cover all this information. If a right request is processed the already gathered information are filled in (see $a \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_{\rightarrow}, l_{\rightarrow}, s_{2,\rightarrow}, z_4, r_1, s_{1,\rightarrow}\rangle$ for the right output request and $a \cdot o(l_s, s_1, s_2, z)\,.\overline{r_{2,\rightarrow}}\langle l_s, l_{\rightarrow}, l_s, s_2, z, s_1, r_{1,\rightarrow}\rangle$ for the right input request). The missing details are gathered by the communication with a matching left request.

Now there are two concurrently enabled steps, at channel $a \cdot o$ and at $a \cdot i$. One for each possible step of the source term. Note that the two steps of the source term are in conflict, whereas the two steps here are not conflicting. The result of these two steps and the four steps on the channels $p_{i,up}$ and $p_{o,up}$ is given in Figure 5.7. We observe that the missing details are filled in. The results are two outputs on receiver locks and the first two parameters are in both cases the same, i.e., the order of the sum locks is the same. The consumption of these outputs enables a test of the sum locks of the respective found pair of matching communication partners. Note that since such a pair consists of a communication partner left and a partner right to the encoding of the respective parallel operator, these two sum locks are always different. Because of that the problem of incestuous sums described in Example 5.2.1 is avoided.

If we reduce both outputs on receiver locks, we obtain the unguarded test-constructs

$$
\begin{aligned}
&\mathsf{test}\ l_{\leftarrow}\ \mathsf{then\ test}\ l_{\rightarrow}\ \mathsf{then}\ \overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{l_{\rightarrow}}\langle\bot\rangle \mid \overline{s_{2,\rightarrow}} \mid \{\ {}^{z_4}/_{z_1}\ \}\left([\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\right) \\
&\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_{\leftarrow}}\langle\top\rangle \mid \overline{l_{\rightarrow}}\langle\bot\rangle \mid \overline{r_{1,\leftarrow}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{s_{1,\rightarrow}}
\end{aligned}
$$

and

$$
\begin{aligned}
&\mathsf{test}\ l_{\leftarrow}\ \mathsf{then\ test}\ l_{\rightarrow}\ \mathsf{then}\ \overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{l_{\rightarrow}}\langle\bot\rangle \mid \overline{s_{2,\leftarrow}} \mid \{\ {}^{z_2}/_{z_3}\ \}\left([\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\right) \\
&\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_{\leftarrow}}\langle\top\rangle \mid \overline{l_{\rightarrow}}\langle\bot\rangle \mid \overline{s_{1,\leftarrow}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{r_{1,\rightarrow}}.
\end{aligned}
$$

$$
\begin{aligned}
&(\nu p_{i,up}, p_{o,up}, \mathsf{i}, \mathsf{o}, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, l_{\rightarrow}, r_{1,\rightarrow}, r_{2,\rightarrow}, s_{1,\rightarrow}, s_{2,\rightarrow})\,(\\
&\quad(\nu p_i, p_o)\,(\overline{l_{\leftarrow}}\langle\top\rangle \mid r^{\star}_{1,\leftarrow}.\overline{p_i}\langle a, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle\\
&\qquad\mid r^{\star}_{2,\leftarrow}(l_1, l_2, l_3, s_2, z_1, v, w)\,.\mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v}\\
&\qquad\qquad\qquad\qquad\qquad\mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\\
&\qquad\mid s^{\star}_{1,\leftarrow}.\overline{p_o}\langle a, l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle \mid s_{2,\leftarrow}.[\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\qquad\mid p_o{}^{\star}(y, l, s_1, s_2, z)\,.\,(\overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle)\\
&\qquad\mid p_i{}^{\star}(y, l, r_1, r_2)\,.\,(\overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle)\,)\\
&\quad(\nu p_i, p_o)\,(\overline{l_{\rightarrow}}\langle\top\rangle \mid r^{\star}_{1,\rightarrow}.\overline{p_i}\langle a, l_{\rightarrow}, r_{1,\rightarrow}, r_{2,\rightarrow}\rangle\\
&\qquad\mid r^{\star}_{2,\rightarrow}(l_1, l_2, l_3, s_2, z_3, v, w)\,.\mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v}\\
&\qquad\qquad\qquad\qquad\qquad\mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\\
&\qquad\mid s^{\star}_{1,\rightarrow}\overline{p_o}\langle a, l_{\rightarrow}, s_{1,\rightarrow}, s_{2,\rightarrow}, z_4\rangle \mid s_{2,\rightarrow}.[\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\\
&\qquad\mid p_o{}^{\star}(y, l_s, s_1, s_2, z)\,.\,(y \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle)\\
&\qquad\mid \overline{\boldsymbol{r_{2,\leftarrow}}}\langle \boldsymbol{l_{\leftarrow}}, l_{\rightarrow}, l_{\rightarrow}, s_{2,\rightarrow}, z_4, \boldsymbol{r_{1,\rightarrow}}, s_{1,\rightarrow}\rangle\\
&\qquad\mid p_i{}^{\star}(y, l_r, r_1, r_2)\,.\,(y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{o,up}}\langle y, l_r, r_1, r_2\rangle)\\
&\qquad\mid \overline{r_{2,\rightarrow}}\langle \boldsymbol{l_{\leftarrow}}, l_{\rightarrow}, \boldsymbol{l_{\leftarrow}}, \boldsymbol{s_{2,\leftarrow}}, \boldsymbol{z_2}, \boldsymbol{s_{1,\leftarrow}}, r_{1,\rightarrow}\rangle\,)\\
&\quad\mid p_{o,up}{}^{\star}(y, l, s_1, s_2, z)\,.\overline{p_o}\langle y, l, s_1, s_2, z\rangle\\
&\quad\mid \overline{\boldsymbol{p_o}}\langle \boldsymbol{a}, \boldsymbol{l_{\leftarrow}}, \boldsymbol{s_{1,\leftarrow}}, \boldsymbol{s_{2,\leftarrow}}, \boldsymbol{z_2}\rangle \mid \overline{\boldsymbol{p_o}}\langle \boldsymbol{a}, \boldsymbol{l_{\rightarrow}}, \boldsymbol{s_{1,\rightarrow}}, \boldsymbol{s_{2,\rightarrow}}, \boldsymbol{z_4}\rangle\\
&\quad\mid p_{i,up}{}^{\star}(y, l, r_1, r_2)\,.\overline{p_i}\langle y, l, r_1, r_2\rangle \mid \overline{\boldsymbol{p_i}}\langle \boldsymbol{a}, \boldsymbol{l_{\leftarrow}}, \boldsymbol{r_{1,\leftarrow}}, \boldsymbol{r_{2,\leftarrow}}\rangle \mid \overline{\boldsymbol{p_i}}\langle \boldsymbol{a}, \boldsymbol{l_{\rightarrow}}, \boldsymbol{r_{1,\rightarrow}}, \boldsymbol{r_{2,\rightarrow}}\rangle\,)
\end{aligned}
$$

Figure 5.7.: Processing of Requests in the Example.

We observe, that for both test-constructs the sum lock $l_{\leftarrow}$ has to be checked first. By Definition 5.1.1, an instantiation of a sum lock has to be consumed to test the lock. Since there is only one instantiation of the lock $l_{\leftarrow}$ in the term in Figure 5.7, these two test-constructs are in conflict. Without loss of generality, let the first test-construct consume the given instantiation of the sum lock $l_{\leftarrow}$. The other case is similar. The instantiation of $l_{\leftarrow}$ is positive. Thus, the test-construct reduces to its then-case and tests the other lock $l_{\rightarrow}$. Again there is only a single instantiation of this lock which is positive. Also note, that the second test-construct is still blocked, because in the then-case of the first part of the nested test-construct no new instantiation of a sum lock is unguarded. After the test of the sum lock $l_{\rightarrow}$ the first test-construct reduces to:

$$\overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{l_{\rightarrow}}\langle\bot\rangle \mid \overline{s_{2,\rightarrow}} \mid \{\ {}^{z_4}/_{z_1}\ \}\left([\![\ 0\ ]\!]^{\mathrm{m}}_{\mathrm{p}}\right)$$

The reduction to the first case shows that our communication attempt on the identified pair of communication partners was successful, i.e., at this point we emulate the corresponding source term step. Both sum locks are changed to false instantiations. This

outlines that already a branch of each of these two sums was used for communication. Then there is an unguarded instantiation of the sender lock $s_{2,\rightarrow}$. So, we can remove the guard of the encoding of the continuation of the output guarded right source term within one more reduction step. The encoding of the continuation of the respective input guarded source term is the term $\{\, {}^{z_4}/_{z_1} \,\} \left([\![\, 0 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \right) = [\![\, 0 \,]\!]^{\mathrm{m}}_{\mathrm{p}}$. As we can observe it is already unguarded. Moreover, the value sent by the respective right source term output was received at the encoding of the left input guarded term as depicted by the substitution $\{\, {}^{z_4}/_{z_1} \,\}$.

S can perform only a single step, which we have emulated within its encoding. The encoded term can perform some post-processing steps. The negative instantiations of the sum locks enable the second nested test-construct. In this case only the first sum lock, the lock $l_{\leftarrow}$, is tested. Because of its negative instantiation, the second nested test-construct reduces to the else-case of its first part, i.e., to $\overline{l_{\leftarrow}}\langle\bot\rangle \mid \overline{r_{1,\rightarrow}}$. The consumed negative instantiation of the sum lock $l_{\leftarrow}$ is restored. Since the second sum lock $l_{\rightarrow}$ was not tested in this case, the test-construct restores also the corresponding input request by sending $\overline{r_{1,\rightarrow}}$. Hence, in case the sum lock is still positive, the encoding can search for another matching communication partner. Thus, reducing the second test-construct results in a duplicate of the right input request. However, since the corresponding sum lock is false, both versions of the right input request cannot be used to emulate a source term step. Even if the sum lock is still positive, duplicates of requests can lead to only a single emulation of a source term step, because the single instance of the sum lock forbids for simultaneous evaluated test-constructs and after the first successful emulation attempt the sum lock becomes false. Also note that, since for each given encoded source term there are initially only finitely many requests, it is not possible to copy a single request infinitely often. Similarly to the first right input request, the second right input request can be transmitted upwards in the parallel structure.

Note that the resulting target term contains five unguarded and not restricted requests: $\overline{p_o}\langle a, l_{\leftarrow}, s_{1,\leftarrow}, s_{2,\leftarrow}, z_2\rangle$, $\overline{p_o}\langle a, l_{\rightarrow}, s_{1,\rightarrow}, s_{2,\rightarrow}, z_4\rangle$, $\overline{p_i}\langle a, l_{\leftarrow}, r_{1,\leftarrow}, r_{2,\leftarrow}\rangle$, and twice $\overline{p_i}\langle a, l_{\rightarrow}, r_{1,\rightarrow}, r_{2,\rightarrow}\rangle$. They can be bound by a surrounding parallel operator encoding, i.e., by the next parent node. Since in our example there is no such surrounding parallel operator they remain free.

5.2.4. Refine the Encoding

In the encoding given in the last section we used polyadic synchronisation to combine the left and right requests within a node of the parallel structure, i.e., within the encoding of a parallel operator. Thereby, we relied on the comfortable binding mechanism of polyadic synchronisation that allows us to restrict a combined channel name by restricting only one of its names. We want to identify communication partners, i.e., the translation of a source term sender and receiver sharing the same channel name. Moreover, our encoding functions relies on the parallel structure and, hence, it is necessary to restrict the search of communication partners to single nodes in this structure, because we need to know whether a request arrives at this node from the left or the right side.

The problem is that within the pi-calculus we cannot restrict received names, but

only names known in advance. By Definition 3.3.1 of compositionality, we can base the encoding function on the free names of the source terms. But this does not help in case of restricted source term names. Requests inform the nodes of the parallel structure about potential communication partners. Remember that the necessity for such requests is already shown in the proof of Lemma 4.2.17. To identify matching partners they contain the translation of the original channel name of the corresponding source term sender or receiver as first parameter. With polyadic synchronisation we can restrict for each such name y a channel name $y \cdot o$ or $y \cdot i$ for each node by restricting o and i. Without polyadic synchronisation there is no possibility to do something like that. Thus we cannot use communication to identify matching communication partners. Instead we use the match prefix to do so. Note that the match prefix increases the expressive power of the pi-calculus, i.e., the considered target language $\pi_{\mathrm{a}}^{=}$ is more expressive then π_{a}. This is already discussed in [CM03]. Note that the criteria for a good encoding used in [CM03] to show that there exists no good encoding of the match prefix are stricter than the criteria used in this thesis. Instead of preservation and reflection of the ability to reach success, [CM03] requires preservation and reflection of all observables of the source language for their proof of separation. This is indeed a very strict requirement. However, we believe that this separation result, i.e., that it is impossible to encode the expressive power of the match prefix within the asynchronous pi-calculus, holds also with respect to the general framework in Section 3.3. Under this assumption, it seems impossible to encode mixed choice within π_{a}.

The match prefix behaves as a conditional guard: if the constraint, i.e., the equation, is satisfied the following term gets unguarded. We cannot ensure that we always magically pick the right two matching requests first, to check whether their first parameters are identical. Hence, we have to deal with the case that the first two parameters do not match. Since the mismatch prefix further significantly increases the expressive power of the calculus, we try to avoid it. Instead we implement an algorithm that combines each pair of left and right requests exactly once. In case a match is found, we proceed by inducing a test on the sum locks as before. Because every combination of left and right requests is checked, we can simply do nothing if the translated channel names do not match.

To keep track of the pairs already checked, we order the right requests within a so-called *chain* along which all left requests are forwarded. More precisely, we introduce two chains for each node of the parallel structure: one for right output and one for right input requests. For this, the encoding of a parallel operator has to reserve the names m_o and m_i as starting points of the chains, and the names c_o and c_i to prepare the adding of a new request to a chain. The tags o and i are not necessary any more.

$$
\begin{aligned}
[\![\, P \mid Q \,]\!]_{\mathrm{a}}^{\mathrm{m}} \triangleq\ & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(\\
& (\nu p_o, p_i)\,\big([\![\, P \,]\!]_{\mathrm{a}}^{\mathrm{m}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big) \\
& \mid (\nu p_o, p_i)\,\big([\![\, Q \,]\!]_{\mathrm{a}}^{\mathrm{m}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big) \\
& \mid \mathsf{pushReq}\big)
\end{aligned}
$$

We change the processing of left requests. Instead of transmuting the requests into

outputs on combined channel names, we forward them to the starting point of the chains on the channels m_o for left output requests and m_i for left input requests. Thus, left requests are processed by two simple forwarders, $\mathsf{procLeftOutReq} \triangleq p_o \twoheadrightarrow \{\, m_o, p_{o,up} \,\}$ and $\mathsf{procLeftInReq} \triangleq p_i \twoheadrightarrow \{\, m_i, p_{i,up} \,\}$.

The processing of requests from the right is more difficult. As described the encoding ensures that any request of the left hand side is combined exactly once with each opposite request of the right hand side. Then the respective first parameters of each pair of requests are compared, to reveal a pair that results from the translation of matching communication partners. If such a pair is found, the information necessary to resolve the respective test-construct is again retransmitted over the receiver lock back to the receiver, where the first parameter contains the respective sum lock at the left and the second parameter the sum lock at the right hand side.

$$
\begin{aligned}
\mathsf{procRightOutReq} \triangleq\ & \overline{c_o}\langle m_i\rangle \mid {c_o}^{\star}(m_i)\,.p_o(y, l_s, s, z)\,.\big(\\
& (\nu m_{i,up})\,\big({m_i}^{\star}\big(y', l_r, r\big)\,.\,\big([y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big) \\
& \qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\langle m_i\rangle)\,\big) \\
& \mid \overline{p_{o,up}}\langle y, l_s, s, z\rangle\,\big) \\
\mathsf{procRightInReq} \triangleq\ & \overline{c_i}\langle m_o\rangle \mid {c_i}^{\star}(m_o)\,.p_i(y, l_r, r)\,.\big(\\
& (\nu m_{o,up})\,\big({m_o}^{\star}\big(y', l_s, s, z\big)\,.\,\big([y' = y]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\
& \qquad \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big) \\
& \mid \overline{p_{i,up}}\langle y, l_r, r\rangle\,\big)
\end{aligned}
$$

In order to emulate arbitrary source term steps, all pairs of left and right requests have to be checked at least once. On the other side, a careless checking of the same pairs infinitely often introduces divergence. Thus, only a single copy of each left request is transmitted to the right side and, there, each pair of left and right requests is combined exactly once. To do so, the right requests are linked together within two chains; one for right output requests and one for right input requests. The first member of the chain receives all left requests via m_o or m_i, combines them with its own information, and sends a copy of each left request to the next member over $m_{o,up}$ or $m_{i,up}$, respectively. Subsequent members of a chain are linked by m_o or m_i, i.e., each member creates a new version of the corresponding name and sends this new version over c_o or c_i to enable the addition of a new member. Moreover, it transmits all received left requests along this new version. A new member is then added to the chain by the consumption of its request, also triggering to transmit a copy to pushReq via $p_{o,up}$ or $p_{i,up}$. The term pushReq remains unchanged.

Note that differently from to the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ there is no need to retransmit requests in case of failed test-constructs, because the protocol in the encoding of the parallel operator already ensures that each combination of requests is checked. Accordingly,

there is no need for the first part of sender and receiver locks.

$$
\begin{aligned}
[\![\, \overline{y}\langle z\rangle .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu s)\,(\overline{p_o}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s.[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}) \\
[\![\, y(x) .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu r)\,\big(\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid r^{\star}(l_1, l_2, -, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x)) . \\
&\qquad \mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \\
&\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle\,\big)
\end{aligned}
$$

The remaining operators (restriction, choice, τ-prefix, and success) are encoded like in $[\![\, \cdot \,]\!]^{\mathrm{m}}_{\mathrm{p}}$.

$$
\begin{aligned}
[\![\, (\nu x)\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu \varphi^{\mathrm{m}}_{\mathrm{a}}(x)\, P)\, [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
\left[\!\!\left[\sum_{i\in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu l)\left(\overline{l}\langle\top\rangle \mid \prod_{i\in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \right) \\
[\![\, \tau.P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle \\
[\![\, \checkmark \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \checkmark
\end{aligned}
$$

Replicated Input In the discussion so far, we omitted the encoding of replicated input, because it is slightly tricky. The crux is that each replicated input implicitly represents an unbounded number of copies of the respective input in parallel. Each such copy changes the parallel structure of the source term, on which our encoding function relies. Obviously, a compositional encoding cannot first compute the number of required copies: By the reduction semantics, the copies of a replicated input are generated as soon as they are needed. Likewise, the encoding of a replicated input adds a branch to the constructed parallel structure, for each emulated communication with a replicated input. To do so, it adapts the parallel operator translation for each unguarded continuation in encodedContinuations.

$$
\begin{aligned}
[\![\, y^{\star}(x) .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \triangleq\ & (\nu l, r, c_{r1}, c_{r2}, r_o, r_i)\,\big(\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \\
& \mid r^{\star}(-, -, l_s, s, z)\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle z\rangle\ \mathsf{else}\ \overline{l_s}\langle\bot\rangle \\
& \mid \overline{r_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid \overline{l}\langle\top\rangle \mid \mathsf{encodedContinuations}\big)
\end{aligned}
$$

To direct the flow of requests among the additional branches, the branches are also ordered into a chain.

$$
\begin{aligned}
\mathsf{encodedContinuations} \triangleq\ & \overline{c_{r2}}\langle r_o, r_i\rangle \mid {c_{r1}}^{\star}(\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.c_{r2}(r_o, r_i)\,. \\
& (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i)\,\big(\mathsf{pushReqIn} \\
& \quad \mid (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \quad \mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathsf{pushReqOut})\,\big)
\end{aligned}
$$

For each successful emulation of a replicated input, a new branch with the encoded continuation is unguarded by transmitting the received source term value over c_{r1}. As

in the chains of right requests, each branch in encodedContinuations restricts its own versions of r_o and r_i to receive all requests from its successor. These links are transmitted over c_{r2} to the respective next member. The translation of the replicated input serves itself as first member of the chain by providing its own request over r_i. Note that the third line of encodedContinuations is exactly the same as the right hand side of a parallel operator encoding. There, all received requests are combined with the requests of the respective continuation to enable the emulation of a communication with the replicated input or another of its unguarded continuations. Moreover, to enable an emulation of a communication with the rest of the term, its requests are pushed upwards. The remaining terms pushReqIn and pushReqOut direct the flow of requests.

$$\begin{aligned} \mathsf{pushReqIn} &\triangleq r_o \twoheadrightarrow \{\, m_o, r_{o,up} \,\} \mid r_i \twoheadrightarrow \{\, m_i, r_{i,up} \,\} \\ \mathsf{pushReqOut} &\triangleq p_{o,up} \twoheadrightarrow \{\, p_o, r_o \,\} \mid r_{o,up} \twoheadrightarrow r_o \mid p_{i,up} \twoheadrightarrow \{\, p_i, r_i \,\} \mid r_{i,up} \twoheadrightarrow r_i \end{aligned}$$

pushReqIn receives all requests from a predecessor in the chain and forwards one copy to the encoded continuation over m_o and m_i and one copy to pushReqOut. There all requests of the encoded continuation are pushed upwards to a surrounding parallel operator encoding (or the encoding of a formally guarding replicated input) over p_o or p_i, and for all such requests and all requests received from a previous member, a copy is forwarded to the successor over r_o or r_i.

The encoding function $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ introduces twenty-eight names covered in the set

$$\begin{aligned} N = \{ & p_o, p_i, p_{o,up}, p_{i,up}, m_o, m_i, c_o, c_i, m_{o,up}, m_{i,up}, l, l_s, l_r, l_1, l_2, s, \\ & r, c_{r1}, c_{r2}, r_o, r_i, r_{o,up}, r_{i,up}, y, y', z, t, f \}. \end{aligned}$$

Again additional names are necessary to unfold the polyadic communications. Hence, $\varphi^{\mathrm{m}}_{\mathrm{a}}$ is an arbitrary injective substitution such that, for all $n \in \mathcal{N}$, $\varphi^{\mathrm{m}}_{\mathrm{a}}(n)$ is neither a name of N nor one of the auxiliary links introduced to translate polyadic communications in Section 5.4. The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is given by the Figures 5.8 and 5.9.

For a more exhaustive description of the algorithm implemented by this encoding and how it emulates source term steps, we refer to the proof of its correctness in Chapter 6. There also some properties of this encoding, as for instance the kinds of junk it introduces, are analysed.

5.2.5. Encoding Example

In the encoding example presented in Section 5.2.3 we concentrate on the flow of requests within the parallel structure, on the identification of matching communication partners, and the processing of test-constructs. These concepts are implemented in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ in basically the same way. So, this time we focus the encoding example on the difference between the encodings $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, i.e., we concentrate on the chain of requests in the encodings of the parallel operator and on the encoding of replicated inputs. We consider the source term:

$$S = \overline{a}\langle c \rangle .0 \mid a^{\star}(d) . ((\nu e)\, \overline{a}\langle e \rangle .0 \mid \overline{e}\langle d \rangle .0)$$

$$
\begin{aligned}
[\![\, (\nu x)\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\, [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
[\![\, P \mid Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\, \big(\\
&\quad (\nu p_o, p_i)\, \big([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \big) \\
&\quad \mid (\nu p_o, p_i)\, \big([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq} \big) \\
&\quad \mid \mathsf{pushReq} \big) \\
\left[\!\!\left[\sum_{i \in I} \pi_i.P_i \right]\!\!\right]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu l) \left(\overline{l}\langle \top \rangle \mid \prod_{i \in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \right) \\
[\![\, \tau.P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle \bot \rangle \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle \bot \rangle \\
[\![\, \overline{y}\langle z \rangle .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu s)\, (\overline{p_o}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z) \rangle \mid s.\, [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}) \\
[\![\, y(x) .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu r)\, \big(\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r \rangle \mid r^{\star}(l_1, l_2, -, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,. \\
&\quad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle \bot \rangle \mid \overline{l_2}\langle \bot \rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\qquad\qquad \mathsf{else}\ \overline{l_1}\langle \top \rangle \mid \overline{l_2}\langle \bot \rangle \\
&\qquad \mathsf{else}\ \overline{l_1}\langle \bot \rangle\, \big) \\
[\![\, y^{\star}(x) .P \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq (\nu l, r, c_{r1}, c_{r2}, r_o, r_i)\, \big(\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r \rangle \\
&\quad \mid r^{\star}(-, -, l_s, s, z)\, .\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle \bot \rangle \mid \overline{s} \mid \overline{c_{r1}}\langle z \rangle\ \mathsf{else}\ \overline{l_s}\langle \bot \rangle \\
&\quad \mid \overline{r_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r \rangle \mid \overline{l}\langle \top \rangle \mid \mathsf{encodedContinuations} \big) \\
[\![\, \checkmark \,]\!]^{\mathrm{m}}_{\mathrm{a}} &\triangleq \checkmark
\end{aligned}
$$

Figure 5.8.: An Encoding from π_{m} into $\pi^{=}_{\mathrm{a}}$.

$$
\begin{aligned}
\mathsf{procLeftOutReq} &\triangleq p_o \twoheadrightarrow \{\, m_o, p_{o,up} \,\} \\
\mathsf{procLeftInReq} &\triangleq p_i \twoheadrightarrow \{\, m_i, p_{i,up} \,\} \\
\mathsf{procRightOutReq} &\triangleq \overline{c_o}\langle m_i \rangle \mid {c_o}^{\star}(m_i)\,.p_o(y, l_s, s, z)\,.\big(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \\
&\quad \mid (\nu m_{i,up})\,\big(\, {m_i}^{\star}\big(y', l_r, r\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big) \\
&\quad\quad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\langle m_i\rangle)\,\big)\big) \\
\mathsf{procRightInReq} &\triangleq \overline{c_i}\langle m_o \rangle \mid {c_i}^{\star}(m_o)\,.p_i(y, l_r, r)\,.\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \\
&\quad \mid (\nu m_{o,up})\,\big(\, {m_o}^{\star}\big(y', l_s, s, z\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\
&\quad\quad \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big)\big) \\
\mathsf{pushReq} &\triangleq p_{o,up} \twoheadrightarrow p_o \mid p_{i,up} \twoheadrightarrow p_i \\
\mathsf{encodedContinuations} &\triangleq \overline{c_{r2}}\langle r_o, r_i\rangle \mid {c_{r1}}^{\star}(\varphi_{\mathrm{a}}^{\mathrm{m}}(x))\,.c_{r2}(r_o, r_i)\,. \\
&\quad (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i)\,\big(\mathsf{pushReqIn} \\
&\quad\quad \mid (\nu p_o, p_i)\,(\llbracket\, P \,\rrbracket_{\mathrm{a}}^{\mathrm{m}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
&\quad\quad \mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathsf{pushReqOut})\,\big) \\
\mathsf{pushReqIn} &\triangleq r_o \twoheadrightarrow \{\, m_o, r_{o,up} \,\} \mid r_i \twoheadrightarrow \{\, m_i, r_{i,up} \,\} \\
\mathsf{pushReqOut} &\triangleq p_{o,up} \twoheadrightarrow \{\, p_o, r_o \,\} \mid r_{o,up} \twoheadrightarrow r_o \mid p_{i,up} \twoheadrightarrow \{\, p_i, r_i \,\} \mid r_{i,up} \twoheadrightarrow r_i
\end{aligned}
$$

Figure 5.9.: Auxiliary Functions of $\llbracket\, \cdot \,\rrbracket_{\mathrm{a}}^{\mathrm{m}}$.

Again, we can assume without loss of generality that $\varphi_{\mathrm{a}}^{\mathrm{m}}(a) = a$, $\varphi_{\mathrm{a}}^{\mathrm{m}}(c) = c$, $\varphi_{\mathrm{a}}^{\mathrm{m}}(d) = d$, and $\varphi_{\mathrm{a}}^{\mathrm{m}}(e) = e$. We concentrate first on the processing of requests on the encoding of the outermost parallel operator of S. Left to that operator is the translation of the output $\overline{a}\langle c\rangle\,.0$. It contains the output request $\overline{p_o}\langle a, l_1, s_1, c\rangle$, where l_1 is the sum lock introduced to encode the left sum containing only the single output and s_1 is the sender lock introduced by the encoding of the output. In the parallel operator encoding the link p_o is bound at the left side and the request is processed by

$$
\begin{aligned}
\mathsf{procLeftOutReq} &\overset{\text{Fig. 5.9}}{=} p_o \twoheadrightarrow \{\, m_o, p_{o,up} \,\} \\
&\overset{\text{Def. 5.2.3}}{=} {p_o}^{\star}(y, l, s, z)\,.\,(\overline{m_o}\langle y, l, s, z\rangle \mid \overline{p_{o,up}}\langle y, l, s, z\rangle)\,.
\end{aligned}
$$

As result a copy of this request is pushed upwards over $p_{o,up}$ and $\mathsf{pushReq}$ and another copy of the requests is transmitted over m_o to the right hand side of the parallel operator encoding. There the replicated input $a^{\star}(d)\,.\,((\nu e)\,\overline{a}\langle e\rangle\,.0 \mid \overline{e}\langle d\rangle\,.0)$ is encoded and leads to the input request $\overline{p_i}\langle a, l_2, r_1\rangle$ for some sum lock l_2 and some receiver lock r_1. This time the request channel p_i is bound by the restriction on the right hand side of the parallel operator encoding and, thus, the request is processed by

$$
\begin{aligned}
\mathsf{procRightInReq} &\overset{\text{Fig. 5.9}}{=} \overline{c_i}\langle m_o \rangle \mid {c_i}^{\star}(m_o)\,.p_i(y, l_r, r)\,.\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \\
&\quad \mid (\nu m_{o,up})\,\big(\, {m_o}^{\star}\big(y', l_s, s, z\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\
&\quad\quad \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big)\big).
\end{aligned}
$$

To do so, the chain that is implemented by procRightInReq has first to be prepared to accept a new member. Thus, a pre-processing step on the chain lock c_i is necessary which leads to

$$p_i(y, l_r, r) \,.\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \\ \mid (\nu m_{o,up}) \big(\boldsymbol{m_o^\star}\big(y', l_s, s, z\big) \,.\, \big([y' = y]\, \overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\ \mid \boldsymbol{(\nu m_o)}\,(m_{o,up} \twoheadrightarrow \boldsymbol{m_o} \mid \overline{c_i}\langle \boldsymbol{m_o}\rangle)\,)\big).$$

It instantiates the link m_o used by the member of the chain to receive left output requests. Since this is the first member, the link is set to the channel used to transfer left requests from the left hand side to the right hand side of the parallel operator encoding. For the next member a fresh instance of m_o is restricted in the last line. The forwarder $m_{o,up} \twoheadrightarrow m_o$ ensures that each left request that arrives is forwarded to the next member. The step on c_i unguards an input on a request channel that allows for our right request to register itself as first member of the chain. The consumption of the right request $\overline{p_i}\langle a, l_2, r_1\rangle$ leads to the term

$$(\nu m_{o,up}) \big(m_o{}^\star\big(y', l_s, s, z\big) \,.\, \big([y' = \boldsymbol{a}]\, \overline{\boldsymbol{r_1}}\langle l_s, \boldsymbol{l_2}, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\ \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big) \mid \overline{p_{i,up}}\langle a, l_2, r_1\rangle \,.$$

We observe that the information carried by $\overline{p_i}\langle a, l_2, r_1\rangle$ is filled in. The source term channel a instantiates the right part of the equation within the match prefix. The receiver lock r_1 ensures that, in case of a found match, the collected information is transmitted back to the encoding of the replicated receiver. And the sum lock of the receiver becomes the second parameter of the output on r_1, because it refers to a sum that is right of the parallel operator encoding and, hence, should be tested as second lock. In the current case, there is only one left output request $\overline{m_o}\langle a, l_1, s_1, c\rangle$. Hence,

$$(\nu m_{o,up}) \big(m_o{}^\star\big(y', l_s, s, z\big) \,.\, \big([y' = a]\, \overline{r_1}\langle l_s, l_2, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\ \mid [\boldsymbol{a} = a]\, \overline{r_1}\langle \boldsymbol{l_1}, l_2, \boldsymbol{l_1}, \boldsymbol{s_1}, \boldsymbol{c}\rangle \mid \overline{m_{o,up}}\langle \boldsymbol{a}, \boldsymbol{l_1}, \boldsymbol{s_1}, \boldsymbol{c}\rangle \\ \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big) \mid \overline{p_{i,up}}\langle a, l_2, r_1\rangle \,.$$

With $\overline{m_{o,up}}\langle a, l_1, s_1, c\rangle$ and the forwarder $m_{o,up} \twoheadrightarrow m_o$ a copy of the consumed left request is transmitted to the next member in the chain (if there is any). Apart from that, the missing information is filled in. Since the match $[a = a]$ is satisfied, it can be removed modulo structural congruence and a test-construct can be started within the encoded replicated input by consuming $\overline{r_1}\langle l_1, l_2, l_1, s_1, c\rangle$.

The replicated input $a^\star(d) \,.\, ((\nu e)\, \overline{a}\langle e\rangle \,.0 \mid \overline{e}\langle d\rangle \,.0)$ is translated into

$$(\nu l_2, r_1, c_{r1}, c_{r2}, r_o, r_i)\, \big(\overline{p_i}\langle a, l_2, r_1\rangle \\ \mid r_1^\star\big(l, l', l_s, s, z\big)\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle z\rangle\ \ \mathsf{else}\ \overline{l_s}\langle\bot\rangle \\ \mid \overline{r_i}\langle a, l_2, r_1\rangle \mid \overline{l_2}\langle\top\rangle \mid \overline{c_{r2}}\langle r_o, r_i\rangle \mid c_{r1}{}^\star(d) \,.\, c_{r2}(r_o, r_i) \,. \\ (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i)\, \big(\mathsf{pushReqIn} \\ \mid (\nu p_o, p_i)\, (\llbracket\, (\nu e)\, \overline{a}\langle e\rangle \,.0 \mid \overline{e}\langle d\rangle \,.0 \,\rrbracket_{\mathrm{a}}^{\mathrm{m}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\ \mid (\nu r_o, r_i)\, (\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathsf{pushReqOut})\,)\,)$$

within $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$. The Consumption of $\overline{r_1}\langle l_1, l_2, l_1, s_1, c\rangle$ unguards the test-construct

$$\mathsf{test}\ \boldsymbol{l_1}\ \mathsf{then}\ \overline{\boldsymbol{l_1}}\langle\bot\rangle \mid \overline{\boldsymbol{s_1}} \mid \overline{c_{r1}}\langle \boldsymbol{c}\rangle\ \mathsf{else}\ \overline{\boldsymbol{l_1}}\langle\bot\rangle$$

which reduces to $\overline{l_1}\langle\bot\rangle \mid \overline{s_1} \mid \overline{c_{r1}}\langle c\rangle$, because of the positive instantiation of l_1 on the left hand side. The instantiation of the sender lock s_1 allows to unguard the continuation of the sender. To unguard the encoded continuation of the replicated source term input, we have to add a branch to the parallel structure. To do so, we consume the instantiation of the chain lock c_{r1} which also finally transmits the received source term name c to the encoded continuation. A second step on the chain lock c_{r2} then initialises the channels r_o, r_i of the new branch, to enable the reception of requests from other branches resulting from earlier reductions of this replicated source term input as well as a copy of the input request of the replicated input itself. The new branch is given by the term

$$\begin{aligned}&(\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i)\,(\mathsf{pushReqIn}\\ &\quad \mid (\nu p_o, p_i)\,(\{\ \boldsymbol{c/d}\ \}\,[\![\ (\nu e)\,\overline{a}\langle e\rangle\,.0 \mid \overline{e}\langle d\rangle\,.0\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\ &\quad \mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathsf{pushReqOut})\,)\,)\end{aligned}$$

We observe that the second line of the new branch is similar to the encoding of the right hand side of a parallel operator. And, indeed, each branch unguarded by an emulation of a communication with a replicated source term input behaves as if its the right side of a parallel operator encoding. It receives "left" requests, i.e., the original input request of the replicated receiver and all requests of preceding branches, over $\mathsf{pushReqIn} \overset{\text{Fig. 5.9}}{=} r_o \twoheadrightarrow \{\ m_o, r_{o,up}\ \} \mid r_i \twoheadrightarrow \{\ m_i, r_{i,up}\ \}$. Since the new branch is the first branch, $\overline{r_i}\langle a, l_2, r_1\rangle$ is the only request that will ever reach it. Note that the new branch produces an output request $\overline{p_o}\langle a, l_3, s_2, e\rangle$ that leads to a matching pair of communication partners. As a consequence, the test-construct in the encoded replicated input is unguarded once more, reduces again to its then-case, and, thus, a second branch is added. This second branch is has its own versions of r_o, r_i. The first branch transmits all its requests over

$$\mathsf{pushReqOut} \overset{\text{Fig. 5.9}}{=} p_{o,up} \twoheadrightarrow \{\ p_o, r_o\ \} \mid r_{o,up} \twoheadrightarrow r_o \mid p_{i,up} \twoheadrightarrow \{\ p_i, r_i\ \} \mid r_{i,up} \twoheadrightarrow r_i$$

to the second branch. That leads again to a possible step with the replicated receiver adding a third branch and so forth.

5.3. Modifications

In Section 5.2.1 we present two very similar ideas to encode mixed choice. The first—proposing a total order on sum locks—was implemented within $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ in the last section. The other idea is to block the number of test-constructs to avoid deadlock by preventing that cyclic dependencies in source term sums lead to cyclic dependencies between test-constructs. It can be implemented by a simple modification of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$.

First, we introduce and restrict a fresh name c denoted as *coordinator lock* in the following. More precisely, each encoding of a parallel operator restricts its own version

of the coordinator lock and initially provides exactly one instantiation of this lock.

$$
\begin{aligned}
[\![\, P \mid Q \,]\!]^{\mathrm{m}}_{\mathrm{a},2} \triangleq\ & (\nu \boldsymbol{c}, m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i) \Big(\\
& (\nu p_o, p_i) \Big([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a},2} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \Big) \\
& \mid (\nu p_o, p_i) \Big([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a},2} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq} \Big) \\
& \mid \overline{\boldsymbol{c}} \mid \mathsf{pushReq} \Big)
\end{aligned}
$$

Then we remove the sum locks from input requests. They are not necessary in this variant of the encoding. The combination of left and right requests is the same as in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. But if a pair of matching requests is found, an instantiation of the coordinator lock has to be consumed before $\overline{r}\langle l_s, s, z, c\rangle$, i.e., before the sum lock of the sender, the sender lock, the sent value, and the name of the corresponding coordinator lock are transmitted to the encoded receiver and its (nested) test-construct. The input on the coordinator lock ensures that for each encoding of a parallel operator at most one test-construct can be unguarded concurrently. Note that this method prevents from deadlocks, because it prevents sets of nested test-constructs with cyclic dependencies between their respective first and second sum locks.

$$
\begin{aligned}
\mathsf{procRightOutReq} \triangleq\ & \overline{c_o}\langle m_i\rangle \mid {c_o}^{\star}(m_i)\,.p_o(y, l_s, s, z)\,.\big(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \\
& \mid (\nu m_{i,up}) \big({m_i}^{\star}\big(y', r\big)\,.\,\big(\big[y' = y\big]\,\boldsymbol{c}.\overline{r}\langle l_s, s, z, \boldsymbol{c}\rangle \mid \overline{m_{i,up}}\langle y', r\rangle\big) \\
& \qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\langle m_i\rangle)\,\big)\big) \\
\mathsf{procRightInReq} \triangleq\ & \overline{c_i}\langle m_o\rangle \mid {c_i}^{\star}(m_o)\,.p_i(y, r)\,.\big(\overline{p_{i,up}}\langle y, r\rangle \\
& \mid (\nu m_{o,up}) \big({m_o}^{\star}\big(y', l_s, s, z\big)\,.\,\big(\big[y' = y\big]\,\boldsymbol{c}.\overline{r}\langle l_s, s, z, \boldsymbol{c}\rangle \mid \overline{m_{o,up}}\langle y', l_s, s, z\rangle\big) \\
& \qquad \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big)\big)
\end{aligned}
$$

Restriction, choice, τ-prefix, output, and success are encoded exactly as in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. In the encoding of inputs we have again to remove the sum lock from the input request. Moreover, we have to adapt the input on the receiver lock according to the above mentioned output, i.e., the first parameter is the sum lock of the sender, the second parameter is the sender lock, the third parameter is the received value, and the last parameter is the name of the coordinator lock of which an instantiation was consumed to unguard the respective output on the receiver lock. In this version of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ it is not necessary to test the sum locks in a particular order, because the blocking implemented by the coordinator lock ensures the absence of deadlocks. Instead the sum lock of the receiver is checked first and then the received sum lock of the sender as it is already done in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. To ensure that each parallel operator encoding allows not only for the emulation of a single step, the completion of a nested test-construct restores the instantiation of

the coordinator lock.

$$\begin{aligned}[\![\, y(x)\,.P\,]\!]^{\mathrm{m}}_{\mathrm{a},2} \triangleq\ & (\nu r)\,\big(\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, r\rangle \mid r^{\star}\big(l', s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x)\,, \boldsymbol{c}\big)\,. \\ & \quad \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P\,]\!]^{\mathrm{m}}_{\mathrm{a},2} \mid \overline{\boldsymbol{c}} \\ & \qquad\qquad \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{\boldsymbol{c}} \\ & \quad \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\boldsymbol{c}}\big)\end{aligned}$$

Note that the coordinator lock is instantiated in each case of the nested test-construct, i.e., it is restored regardless of the values of the tested sum locks.

It remains to adapt the encoding of replicated inputs. Remember that the translation of replicated inputs is a combination of the protocol assumed for the encoding of the parallel operator and the protocol assumed for the encoding of inputs. So, we have to restrict a fresh version of the coordinator lock, provide exactly one initial instantiation of this lock, adapt the input on the receiver lock, and ensure that for each outcome of the test-construct an instantiation of the coordinator lock, whose instantiation was consumed to unguard the output on the receiver lock and whose name was transmitted over the receiver lock, is restored.

$$\begin{aligned}[\![\, y^{\star}(x)\,.P\,]\!]^{\mathrm{m}}_{\mathrm{a},2} \triangleq\ & (\nu \boldsymbol{c}, r, c_{r1}, c_{r2}, r_o, r_i)\,\big(\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, r\rangle \mid \overline{\boldsymbol{c}} \\ & \mid r^{\star}(l, s, z, \boldsymbol{c})\,.\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle z\rangle \mid \overline{\boldsymbol{c}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\boldsymbol{c}} \\ & \mid \overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, r\rangle \mid \mathsf{encodedContinuations}\big)\end{aligned}$$

Finally, change all remaining occurrences of $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ into $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a},2}$ and change the forwarders on requests matching to the new multiplicity of requests.

In general, adaptations of encodings are used to improve an encoding, i.e., to satisfy more or stricter criteria, or to obtain an alternative of an encoding which is e.g. better suited for another domain or can be implemented in an easier way in a particular setting. However, the presented modification above leads to an encoding that is very similar to the variant presented in Section 5.2. By Theorem 4.4.8 and because $[\![\,\cdot\,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is a good encoding (compare to [Nes00] or Chapter 6), both encodings from π_{m} into π_{a} do not preserve distributability. The encoding $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a},2}$ restricts the number of emulation attempts that can be performed concurrently, while $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ sequentialises only single steps of such emulation attempts. Accordingly, one may consider the encoding $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ as slightly "better" than $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a},2}$, but this is rather a point of view and does not underpin any of the quality criteria of the general framework. However, for the rest of this thesis we concentrate on the former encoding, i.e., do not prove that also the latter satisfies the criteria of the general framework.

5.4. Composing Encodings

Assume there are three process calculi $\mathcal{L}_A = \langle\, \mathcal{P}_A, \longmapsto_A \,\rangle$, $\mathcal{L}_B = \langle\, \mathcal{P}_B, \longmapsto_B \,\rangle$, and $\mathcal{L}_C = \langle\, \mathcal{P}_C, \longmapsto_C \,\rangle$ and two good encodings; the encoding $[\![\,\cdot\,]\!]^{A}_{B} : \mathcal{P}_A \to \mathcal{P}_B$ from $\mathcal{L}_A$ into $\mathcal{L}_B$ and the encoding $[\![\,\cdot\,]\!]^{B}_{C} : \mathcal{P}_B \to \mathcal{P}_C$ from $\mathcal{L}_B$ into $\mathcal{L}_C$. Moreover, let $[\![\,\cdot\,]\!]^{A}_{C}$ be the

compositions of these two encodings, i.e., let $[\![S]\!]^A_C = \Big[\!\Big[[\![S]\!]^A_B \Big]\!\Big]^B_C$ for all $S \in \mathcal{P}_A$. Then the natural question arises, whether the composition $[\![\cdot]\!]^A_C$ is again a good encoding from $\mathcal{L}_A$ into $\mathcal{L}_C$. Unfortunately, as discussed in [Gor10a], the composition of two good encodings is *not* necessarily a good encoding, because operational correspondence holds for the composition only under some assumptions. It is quite obvious that the composition $[\![\cdot]\!]^A_C$ is again success sensitive and reflects divergence.[2] [Gor10a] also proves that the first two criteria, compositionality and name invariance, hold for the composition (for an appropriate combination of the respective renaming policies).

The problem with operational correspondence is that it is defined modulo an equivalence $\asymp$ on the respective target language. This is necessary to get rid of junks possibly left over by emulations, but it prevents that the composition of two good encodings satisfies again operational correspondence. Note that the composition $[\![\cdot]\!]^A_C$ should satisfy operational correspondence with respect to the version of $\asymp$ of the second target language, i.e., of $\mathcal{L}_C$ in the current case. Let us denote the variant of $\asymp$ used in $[\![\cdot]\!]^B_C$ by $\asymp_C$ and the variant used in $[\![\cdot]\!]^A_B$ by $\asymp_B$, respectively. Then, [Gor10a] proves that $[\![\cdot]\!]^A_C$ satisfies operational correspondence if $[\![\cdot]\!]^B_C$ preserves the equivalence $\asymp_B$, i.e., if $S_1 \asymp_B S_2$ implies $[\![S_1]\!]^B_C \asymp_C [\![S_2]\!]^B_C$ for all $S_1, S_2 \in \mathcal{P}_B$, and $\asymp_C$ is reduction closed, i.e., $P \asymp_C Q$ and $P \Longmapsto_C P'$ implies that there exists $Q' \in \mathcal{P}_C$ such that $Q \Longmapsto_C Q'$ and $P' \asymp_C Q'$ for all $P, P', Q \in \mathcal{P}_C$.

Since the preservation of the equivalence $\asymp_B$ and the reduction closedness of $\asymp_C$ are very strict properties, Gorla provides also a second setting in which operational correspondence holds for the combination of good encodings. For this a stricter version of operational correspondence is proposed:

Completeness:	For all $S \Longmapsto_{\mathrm{S}} S'$, it holds $[\![S]\!] \Longmapsto_{\mathrm{T}} [\![S']\!] \mid T$ for some $T \asymp \mathsf{0}$.
Soundness:	For all $[\![S]\!] \Longmapsto_{\mathrm{T}} T$, there exists an S' such that $S \Longmapsto_{\mathrm{S}} S'$ and $T \Longmapsto_{\mathrm{T}} [\![S']\!] \mid T$ for some $T \asymp \mathsf{0}$.

Here, S and S' are terms of the source language and $\longmapsto_{\mathrm{S}}$ is its reduction semantics, whereas T is a term of the target language, $\longmapsto_{\mathrm{T}}$ is its reduction semantics, and $\asymp$ is an equivalence on the target language. Then, [Gor10a] proves that $[\![\cdot]\!]^A_C$ satisfies operational correspondence if $[\![\cdot]\!]^A_B$ and $[\![\cdot]\!]^B_C$ satisfy the stricter version of operational correspondence and if $[\![\cdot]\!]^B_C$ preserves the equivalence class of $\mathsf{0}$, i.e., if $S \asymp_B \mathsf{0}$ implies $[\![S]\!]^B_C \asymp_C \mathsf{0}$ for all $S \in \mathcal{P}_B$, and translates the parallel operator homomorphically.

The Figures 5.1, 5.4, and 5.8 specify (together with the abbreviations in Definition 5.2.3 and Definition 5.1.1) encodings from π_{s} into $\pi_{\mathrm{a}}^{\sim}$, from π_{m} into $\pi_{\mathrm{p}}^{\sim}$, and from π_{m} into $\pi_{\mathrm{a}}^{\sim}$. To turn these encodings into encodings from π_{s} into π_{a}, from π_{m} into π_{p}, and from π_{m} into π_{a}, respectively, we have to get rid of the polyadic communications. For this, [NP00] uses an encoding from $\pi_{\mathrm{a}}^{\sim}$ into π_{a} that goes back to [HT91]. Let us denote this encoding $[\![\cdot]\!]^{\sim}$. It acts homomorphic for all operators except for action prefixes for

[2] Note that in [Gor10a] a slightly stricter version of the general framework is presented that also requires preservation of divergence.

which it is defined as

$$\begin{aligned}
[\![\, \overline{y}\langle z_1,\ldots,z_n\rangle \,]\!]^{\sim} &\triangleq (\nu u)\,(\overline{y}\langle u\rangle \mid u(u_1).\,(\overline{u_1}\langle z_1\rangle \mid \ldots \mid u(u_n).\,(\overline{u_n}\langle z_n\rangle)\ldots))\\
[\![\, y(x_1,\ldots,x_n).P \,]\!]^{\sim} &\triangleq y(u).\Big((\nu u_1)\,\Big(\overline{u}\langle u_1\rangle \mid u_1(x_1).\Big(\ldots\\
&\qquad (\nu u_n)\,(\overline{u}\langle u_n\rangle \mid u_n(x_n).\,[\![\, P \,]\!]^{\sim})\ldots\Big)\Big)\Big),
\end{aligned}$$

where for simplicity we omit the renaming policy that has to ensure in this case that $u, u_1, \ldots, u_n \in \mathcal{N}$ are fresh names.

By the above argumentation, we can combine the polyadic variants of the encodings as given in Figures 5.1, 5.4, and 5.8 and the encoding $[\![\,\cdot\,]\!]^{\sim}$ to obtain the desired encodings if they satisfy one of the two discussed settings. Unfortunately, $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ as given in Figure 5.8 violates the second setting, because it does not translates the parallel operator homomorphically and does also not satisfy the stricter notion of operational correspondence. To use the first setting, we have to fix the relation $\asymp$ for this encoding which we deliberately avoid up to now. We discuss this equivalence in Section 6.3.4. Accordingly, we do not combine these encodings, but instead consider polyadic communications only as abbreviations. Note that we do not use the full power of polyadic communication. Thus, to obtain terms of the desired encodings into the target languages π_{a}, π_{p}, and $\pi^{=}_{\mathrm{a}}$, we have to unfold all polyadic communications in Figures 5.1, 5.4, and 5.8 as described by the following definition.

Definition 5.4.1 (Unfolding Polyadic Communications)**.** To unfold the polyadic communications we define (for each encoding) the function $\mathrm{u}(\cdot)$ that has to be applied on the definition of the encoding in the respective figure. We consider the encoding $[\![\,\cdot\,]\!]^{\mathrm{s}}_{\mathrm{a}}$ first. In this case, we have to unfold in Figure 5.1 all outputs and (replicated) inputs with multiplicity zero by

$$\begin{array}{lcllcl}
\mathrm{u}_1(\overline{t}) &\triangleq& (\nu v_t)\,\overline{t}\langle v_t\rangle & \mathrm{u}_1(t.P) &\triangleq& t(v_t).\,\mathrm{u}_1(P)\\
\mathrm{u}_1(\overline{f}) &\triangleq& (\nu v_f)\,\overline{f}\langle v_f\rangle & \mathrm{u}_1(f.P) &\triangleq& f(v_f).\,\mathrm{u}_1(P)\\
\mathrm{u}_1(\overline{s}) &\triangleq& (\nu v_s)\,\overline{s}\langle v_s\rangle & \mathrm{u}_1(s.P) &\triangleq& s(v_s).\,\mathrm{u}_1(P)\\
\mathrm{u}_1(\overline{r}) &\triangleq& (\nu v_{s,r})\,\overline{r}\langle v_{s,r}\rangle & \mathrm{u}_1(r^{\star}.P) &\triangleq& r^{\star}(v_{s,r}).\,\mathrm{u}_1(P)
\end{array}$$

all outputs and inputs with multiplicity two by

$$\begin{aligned}
\mathrm{u}_1\big(\overline{l}\langle t,f\rangle\big) &\triangleq (\nu u_{\sim,l})\,\big(\overline{l}\langle u_{\sim,l}\rangle \mid u_{\sim,l}(u_t).\,(\overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f).\,\overline{u_f}\langle f\rangle)\big)\\
\mathrm{u}_1(l(t,f).P) &\triangleq l(u_{\sim,l}).\,((\nu u_t)\,(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t).\,((\nu u_f)\,(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f).\,\mathrm{u}_1(P)))))
\end{aligned}$$

and all outputs and inputs with multiplicity three by

$$\begin{aligned}
\mathrm{u}_1\Big(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l,s,\varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle\Big) &\triangleq (\nu u_{\sim,i})\,\Big(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle u_{\sim,T_S}\rangle \mid u_{\sim,T_S}(u_l).\big(\overline{u_l}\langle l\rangle \mid u_{\sim,T_S}(u_s).\big(\overline{u_s}\langle s\rangle\\
&\qquad \mid u_{\sim,T_S}(u_{T_S}).\,\overline{u_{T_S}}\langle\varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle\,)\,)\Big)\\
\mathrm{u}_1(\varphi^{\mathrm{s}}_{\mathrm{a}}(y)(l,s,\varphi^{\mathrm{s}}_{\mathrm{a}}(x)).P) &\triangleq \varphi^{\mathrm{s}}_{\mathrm{a}}(y)(u_{\sim,T_S}).\big((\nu u_l)\,\big(\overline{u_{\sim,T_S}}\langle u_l\rangle \mid u_l(l).\big((\nu u_s)\,\big(\overline{u_{\sim,T_S}}\langle u_s\rangle\\
&\qquad \mid u_s(s).\,((\nu u_{T_S})\,(\overline{u_{\sim,T_S}}\langle u_{T_S}\rangle \mid u_{T_S}(\varphi^{\mathrm{s}}_{\mathrm{a}}(x)).\,\mathrm{u}_1(P)))\,\big)\big)\big)\big).
\end{aligned}$$

To obtain $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, we have to unfold in Figure 5.4 all outputs and inputs with multiplicity zero by

$$\begin{array}{rcl@{\qquad}rcl}
\mathrm{u}_2(\overline{t}) & \triangleq & (\nu v_t)\,\overline{t}\langle v_t\rangle & \mathrm{u}_2(t.P) & \triangleq & t(v_t)\,.\,\mathrm{u}_2(P)\\
\mathrm{u}_2(\overline{f}) & \triangleq & (\nu v_f)\,\overline{f}\langle v_f\rangle & \mathrm{u}_2(f.P) & \triangleq & f(v_f)\,.\,\mathrm{u}_2(P)\\
\mathrm{u}_2(\overline{s_1}) & \triangleq & (\nu v_{s,r})\,\overline{s_1}\langle v_{s,r}\rangle & \mathrm{u}_2(s_1.P) & \triangleq & s_1(v_{s,r})\,.\,\mathrm{u}_2(P)\\
\mathrm{u}_2(\overline{s_2}) & \triangleq & (\nu v_s)\,\overline{s_2}\langle v_s\rangle & \mathrm{u}_2(s_2.P) & \triangleq & s_2(v_s)\,.\,\mathrm{u}_2(P)\\
\mathrm{u}_2(\overline{r_1}) & \triangleq & (\nu v_{s,r})\,\overline{r_1}\langle v_{s,r}\rangle & \mathrm{u}_2(r_1.P) & \triangleq & r_1(v_{s,r})\,.\,\mathrm{u}_2(P)
\end{array}$$

all outputs and inputs with multiplicity two by

$$\begin{array}{rcl}
\mathrm{u}_2\big(\overline{l}\langle t,f\rangle\big) & \triangleq & (\nu u_{\sim,l})\,\big(\overline{l}\langle u_{\sim,l}\rangle \mid u_{\sim,l}(u_t)\,.\,(\overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f)\,.\,\overline{u_f}\langle f\rangle)\big)\\
\mathrm{u}_2(l(t,f)\,.P) & \triangleq & l(u_{\sim,l})\,.\,((\nu u_t)\,(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,((\nu u_f)\,(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,\mathrm{u}_2(P)))))
\end{array}$$

all outputs and inputs with multiplicity three by

$$\begin{array}{rcl}
\mathrm{u}_2\big(\overline{y\cdot i}\langle l,r_1,r_2\rangle\big) & \triangleq & (\nu u_{\sim,t_i})\,\big(\overline{y\cdot i}\langle u_{\sim,t_i}\rangle \mid u_{\sim,t_i}(u_l)\,.\big(\overline{u_l}\langle l\rangle \mid u_{\sim,t_i}(u_{s,r})\,.\big(\overline{u_{s,r}}\langle r_1\rangle\\
& & \quad \mid u_{\sim,t_i}(u_{r'})\,.\,\overline{u_{r'}}\langle r_2\rangle\big)\big)\big)\\
\mathrm{u}_2(y\cdot i(l,r_1,r_2)\,.P) & \triangleq & y\cdot i(u_{\sim,t_i})\,.\big((\nu u_l)\,\big(\overline{u_{\sim,t_i}}\langle u_l\rangle \mid u_l(l)\,.\big((\nu u_{s,r})\,\big(\overline{u_{\sim,t_i}}\langle u_{s,r}\rangle\\
& & \quad \mid u_{s,r}(r_1)\,.\,((\nu u_{r'})\,(\overline{u_{\sim,t_i}}\langle u_{r'}\rangle \mid u_{r'}(r_2)\,.\,\mathrm{u}_2(P)))\big)\big)\big)\big)
\end{array}$$

all outputs and (replicated) inputs with multiplicity four by

$$\begin{array}{rcl}
\mathrm{u}_2(\overline{p_i}\langle y,l,r_1,r_2\rangle) & \triangleq & \big(\nu u_{\sim,i'}\big)\,\big(\overline{p_i}\big\langle u_{\sim,i'}\big\rangle \mid u_{\sim,i'}(u_n)\,.\big(\overline{u_n}\langle y\rangle \mid u_{\sim,i'}(u_l)\,.\big(\overline{u_l}\langle l\rangle\\
& & \quad \mid u_{\sim,i'}(u_{s,r})\,.\big(\overline{u_{s,r}}\langle r_1\rangle \mid u_{\sim,i'}(u_{r'})\,.\,\overline{u_{r'}}\langle r_2\rangle\big)\big)\big)\big)\\
\mathrm{u}_2(\overline{y\cdot o}\langle l,s_1,s_2,z\rangle) & \triangleq & (\nu u_{\sim,t_o})\,\big(\overline{y\cdot o}\langle u_{\sim,t_o}\rangle \mid u_{\sim,t_o}(u_l)\,.\big(\overline{u_l}\langle l\rangle \mid u_{\sim,t_o}(u_{s,r})\,.\big(\overline{u_{s,r}}\langle s_1\rangle\\
& & \quad \mid u_{\sim,t_o}(u_s)\,.\big(\overline{u_s}\langle s_2\rangle \mid u_{\sim,t_o}(u_n)\,.\,\overline{u_n}\langle z\rangle\big)\big)\big)\big)\\
\mathrm{u}_2(p_i{}^{\star}(y,l,r_1,r_2)\,.P) & \triangleq & p_i{}^{\star}\big(u_{\sim,i'}\big)\,.\big((\nu u_n)\,\big(\overline{u_{\sim,i'}}\langle u_n\rangle \mid u_n(y)\,.\big((\nu u_l)\,\big(\overline{u_{\sim,i'}}\langle u_l\rangle\\
& & \quad \mid u_l(l)\,.\big((\nu u_{s,r})\,\big(\overline{u_{\sim,i'}}\langle u_{s,r}\rangle \mid u_{s,r}(r_1)\,.\big((\nu u_{r'})\,\big(\overline{u_{\sim,i'}}\langle u_{r'}\rangle\\
& & \quad \mid u_{r'}(r_2)\,.\,\mathrm{u}_2(P)\big)\big)\big)\big)\big)\big)\big)\big)\\
\mathrm{u}_2(y\cdot o(l,s_1,s_2,z)\,.P) & \triangleq & y\cdot o(u_{\sim,t_o})\,.\big((\nu u_l)\,\big(\overline{u_{\sim,t_o}}\langle u_l\rangle \mid u_l(l)\,.\big((\nu u_{s,r})\,\big(\overline{u_{\sim,t_o}}\langle u_{s,r}\rangle\\
& & \quad \mid u_{s,r}(s_1)\,.\big((\nu u_s)\,\big(\overline{u_{\sim,t_o}}\langle u_s\rangle \mid u_s(s_2)\,.\big((\nu u_n)\,\big(\overline{u_{\sim,t_o}}\langle u_n\rangle\\
& & \quad \mid u_n(z)\,.\,\mathrm{u}_2(P)\big)\big)\big)\big)\big)\big)\big)\big)
\end{array}$$

all outputs and replicated inputs with multiplicity five by

$$\begin{array}{rcl}
\mathrm{u}_2(\overline{p_o}\langle y,l,s_1,s_2,z\rangle) & \triangleq & \big(\nu u_{\sim,o'}\big)\,\big(\overline{p_o}\big\langle u_{\sim,o'}\big\rangle \mid u_{\sim,o'}(u_n)\,.\big(\overline{u_n}\langle y\rangle \mid u_{\sim,o'}(u_l)\,.\big(\overline{u_l}\langle l\rangle\\
& & \quad \mid u_{\sim,o'}(u_{s,r})\,.\big(\overline{u_{s,r}}\langle s_1\rangle \mid u_{\sim,o'}(u_s)\,.\big(\overline{u_s}\langle s_2\rangle\\
& & \quad \mid u_{\sim,o'}(u_n)\,.\,\overline{u_n}\langle z\rangle\big)\big)\big)\big)\big)\\
\mathrm{u}_2(p_o{}^{\star}(y,l,s_1,s_2,z)\,.P) & \triangleq & p_o{}^{\star}\big(u_{\sim,o'}\big)\,.\big((\nu u_n)\,\big(\overline{u_{\sim,o'}}\langle u_n\rangle \mid u_n(y)\,.\big((\nu u_l)\,\big(\overline{u_{\sim,o'}}\langle u_l\rangle\\
& & \quad \mid u_l(l)\,.\big((\nu u_{s,r})\,\big(\overline{u_{\sim,o'}}\langle u_{s,r}\rangle \mid u_{s,r}(s_1)\,.\big((\nu u_s)\,\big(\overline{u_{\sim,o'}}\langle u_s\rangle\\
& & \quad \mid u_s(s_2)\,.\,\big((\nu u_n)\,\big(\overline{u_{\sim,o'}}\langle u_n\rangle \mid u_n(z)\,.\,\mathrm{u}_2(P)\big)\big)\big)\big)\big)\big)\big)\big)\big)
\end{array}$$

and all outputs and replicated inputs with multiplicity seven by

$$\begin{aligned}
\mathrm{u}_2(\overline{r_2}\langle l_1, l_2, l_s, s_2, z, v, w\rangle) &\triangleq (\nu u_{\sim,r})\,\big(\overline{r_2}\langle u_{\sim,r}\rangle \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\langle l_1\rangle \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\langle l_2\rangle \\
&\quad \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\langle l_s\rangle \mid u_{\sim,r}(u_s)\,.\big(\overline{u_s}\langle s_2\rangle \\
&\quad \mid u_{\sim,r}(u_n)\,.\big(\overline{u_n}\langle z\rangle \mid u_{\sim,r}(u_{s,r})\,.\big(\overline{u_{s,r}}\langle v\rangle \\
&\quad \mid u_{\sim,r}(u_{s,r})\,.\,\overline{u_{s,r}}\langle w\rangle\,)\,)\,)\,)\,)\,)\,)\\
\mathrm{u}_2(r_2^\star(l_1, l_2, l_s, s_2, z, v, w)\,.P) &\triangleq r_2^\star(u_{\sim,r})\,.\big(\,(\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \mid u_l(l_1)\,.\big(\,(\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \\
&\quad \mid u_l(l_2)\,.\big(\,(\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \mid u_l(l_s)\,.\big(\,(\nu u_s)\,\big(\overline{u_{\sim,r}}\langle u_s\rangle \\
&\quad \mid u_s(s_2)\,.\big(\,(\nu u_n)\,\big(\overline{u_{\sim,r}}\langle u_n\rangle \mid u_n(z)\,.\big(\,(\nu u_{s,r})\,\big(\overline{u_{\sim,r}}\langle u_{s,r}\rangle \\
&\quad \mid u_{s,r}(v)\,.\big(\,(\nu u_{s,r})\,\big(\overline{u_{\sim,r}}\langle u_{s,r}\rangle \\
&\quad \mid u_{s,r}(w)\,.\,\mathrm{u}_2(P)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,)\,).
\end{aligned}$$

Finally, to obtain $[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}$, we have to unfold in Figure 5.8 all outputs and inputs with multiplicity zero by

$$\begin{array}{lcllcl}
\mathrm{u}_3\big(\overline{t}\big) & \triangleq & (\nu v_t)\,\overline{t}\langle v_t\rangle & \mathrm{u}_3(t.P) & \triangleq & t(v_t)\,.\,\mathrm{u}_3(P)\\
\mathrm{u}_3\big(\overline{f}\big) & \triangleq & (\nu v_f)\,\overline{f}\langle v_f\rangle & \mathrm{u}_3(f.P) & \triangleq & f(v_f)\,.\,\mathrm{u}_3(P)\\
\mathrm{u}_3(\overline{s}) & \triangleq & (\nu v_s)\,\overline{s}\langle v_s\rangle & \mathrm{u}_3(s.P) & \triangleq & s(v_s)\,.\,\mathrm{u}_3(P)
\end{array}$$

all outputs and inputs with multiplicity two by

$$\begin{aligned}
\mathrm{u}_3\big(\overline{l}\langle t, f\rangle\big) &\triangleq (\nu u_{\sim,l})\,\big(\overline{l}\langle u_{\sim,l}\rangle \mid u_{\sim,l}(u_t)\,.\,(\overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f)\,.\,\overline{u_f}\langle f\rangle)\big)\\
\mathrm{u}_3(\overline{c_{r2}}\langle r_o, r_i\rangle) &\triangleq (\nu u_{\sim,c})\,(\overline{c_{r2}}\langle u_{\sim,c}\rangle \mid u_{\sim,c}(u_o)\,.\,(\overline{u_o}\langle r_o\rangle \mid u_{\sim,c}(u_i)\,.\,\overline{u_i}\langle r_i\rangle))\\
\mathrm{u}_3(l(t, f)\,.P) &\triangleq l(u_{\sim,l})\,.\,((\nu u_t)\,(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,((\nu u_f)\,(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,\mathrm{u}_3(P)))))\\
\mathrm{u}_3(c_{r2}(r_o, r_i)\,.P) &\triangleq c_{r2}(u_{\sim,c})\,.\big(\,(\nu u_o)\,\big(\overline{u_{\sim,c}}\langle u_o\rangle \mid u_o(r_o)\,.\big(\,(\nu u_i)\,\big(\overline{u_{\sim,c}}\langle u_i\rangle \\
&\quad \mid u_i(r_i)\,.\,\mathrm{u}_3(P)\,)\,)\,)\,)
\end{aligned}$$

all outputs and (replicated) inputs with multiplicity three by

$$\begin{aligned}
\mathrm{u}_3(\overline{p_i}\langle y, l, r\rangle) &\triangleq (\nu u_{\sim,i})\,\big(\overline{p_i}\langle u_{\sim,i}\rangle \mid u_{\sim,i}(u_n)\,.\big(\overline{u_n}\langle y\rangle \mid u_{\sim,i}(u_l)\,.\big(\overline{u_l}\langle l\rangle \\
&\quad \mid u_{\sim,i}(u_r)\,.\,\overline{u_r}\langle r\rangle\,)\,)\,)\\
\mathrm{u}_3(p_i(y, l, r)\,.P) &\triangleq p_i(u_{\sim,i})\,.\big(\,(\nu u_n)\,\big(\overline{u_{\sim,i}}\langle u_n\rangle \mid u_n(y)\,.\big(\,(\nu u_l)\,\big(\overline{u_{\sim,i}}\langle u_l\rangle \\
&\quad \mid u_l(l)\,.\,((\nu u_r)\,(\overline{u_{\sim,i}}\langle u_r\rangle \mid u_r(r)\,.\,\mathrm{u}_3(P)))\,)\,)\,)\,)\\
\mathrm{u}_3(p_i{}^\star(y, l, r)\,.P) &\triangleq p_i{}^\star(u_{\sim,i})\,.\big(\,(\nu u_n)\,\big(\overline{u_{\sim,i}}\langle u_n\rangle \mid u_n(y)\,.\big(\,(\nu u_l)\,\big(\overline{u_{\sim,i}}\langle u_l\rangle \\
&\quad \mid u_l(l)\,.\,((\nu u_r)\,(\overline{u_{\sim,i}}\langle u_r\rangle \mid u_r(r)\,.\,\mathrm{u}_3(P)))\,)\,)\,)\,)
\end{aligned}$$

all outputs and (replicated) inputs with multiplicity four by

$$\begin{aligned}
\mathrm{u}_3(\overline{p_o}\langle y,l,s,z\rangle) &\triangleq (\nu u_{\sim,o})\,\big(\overline{p_o}\langle u_{\sim,o}\rangle \mid u_{\sim,o}(u_n).\big(\overline{u_n}\langle y\rangle \mid u_{\sim,o}(u_l).\big(\overline{u_l}\langle l\rangle \\
&\quad \mid u_{\sim,o}(u_s).\big(\overline{u_s}\langle s\rangle \mid u_{\sim,o}(u_n).\overline{u_n}\langle z\rangle\,\big)\big)\big)\big)\\
\mathrm{u}_3(p_o(y,l,s,z).P) &\triangleq p_o(u_{\sim,o}).\big((\nu u_n)\,\big(\overline{u_{\sim,o}}\langle u_n\rangle \mid u_n(y).\big((\nu u_l)\,\big(\overline{u_{\sim,o}}\langle u_l\rangle \\
&\quad \mid u_l(l).\big((\nu u_s)\,\big(\overline{u_{\sim,o}}\langle u_s\rangle \mid u_s(s).\big((\nu u_n)\,\big(\overline{u_{\sim,o}}\langle u_n\rangle \\
&\quad \mid u_n(z).\,\mathrm{u}_3(P)\,\big)\big)\big)\big)\big)\big)\big)\big)\\
\mathrm{u}_3(p_o{}^{\star}(y,l,s,z).P) &\triangleq p_o{}^{\star}(u_{\sim,o}).\big((\nu u_n)\,\big(\overline{u_{\sim,o}}\langle u_n\rangle \mid u_n(y).\big((\nu u_l)\,\big(\overline{u_{\sim,o}}\langle u_l\rangle \\
&\quad \mid u_l(l).\big((\nu u_s)\,\big(\overline{u_{\sim,o}}\langle u_s\rangle \mid u_s(s).\big((\nu u_n)\,\big(\overline{u_{\sim,o}}\langle u_n\rangle \\
&\quad \mid u_n(z).\,\mathrm{u}_3(P)\,\big)\big)\big)\big)\big)\big)\big)\big)
\end{aligned}$$

and all outputs and replicated inputs with multiplicity five by

$$\begin{aligned}
\mathrm{u}_3\big(\overline{r}\big\langle l_1,l_2,l',s,z\big\rangle\big) &\triangleq (\nu u_{\sim,r})\,\big(\overline{r}\langle u_{\sim,r}\rangle \mid u_{\sim,r}(u_l).\big(\overline{u_l}\langle l_1\rangle \mid u_{\sim,r}(u_l).\big(\overline{u_l}\langle l_2\rangle \\
&\quad \mid u_{\sim,r}(u_l).\big(\overline{u_l}\big\langle l'\big\rangle \mid u_{\sim,r}(u_s).\,\big(\overline{u_s}\langle s\rangle \mid u_{\sim,r}(u_n).\overline{u_n}\langle z\rangle\big)\big)\big)\big)\big)\\
\mathrm{u}_3\big(r^{\star}\big(l_1,l_2,l',s,z\big).P\big) &\triangleq r^{\star}(u_{\sim,r}).\big((\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \mid u_l(l_1).\big((\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \\
&\quad \mid u_l(l_2).\big((\nu u_l)\,\big(\overline{u_{\sim,r}}\langle u_l\rangle \mid u_l\big(l'\big).\big((\nu u_s)\,\big(\overline{u_{\sim,r}}\langle u_s\rangle \\
&\quad \mid u_s(s).\,((\nu u_n)\,(\overline{u_{\sim,r}}\langle u_n\rangle \mid u_n(z).\,\mathrm{u}_3(P)))\,\big)\big)\big)\big)\big)\big)\big)\big).
\end{aligned}$$

Note that the use of distinct auxiliary names for different purposes instead of the names $u, u_1, \ldots, u_n$ eases the typing of the respective encoding function in Section 6.2.3.

5.5. Summary

The main contribution of this chapter is the presentation of a good encoding of mixed choice. To the best of our knowledge, this is the first good encoding from π_{m} into $\pi_{\mathrm{a}}^{=}$—and in particular the first good encoding of mixed choice. We based this encoding on an encoding of separated choice, namely $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, from Nestmann in [Nes00]. Nestmann also introduces some candidates for an encoding from π_{m} into π_{a} that either violate compositionality or divergence reflection. Based on these discussions and the information gained from the translational separation results in (the first half of) Chapter 4 we derived a good encoding from π_{m} into $\pi_{\mathrm{a}}^{=}$. For this, we first reviewed the concept of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ from [Nes00] (Section 5.1). In order to simplify the presentation of the rather complex new encoding we introduced it in two steps. Note that also e.g. in [PS92, Nes96] encodings are introduced step-wise to reduce the complexity of the single encodings. In Section 5.2.2 we introduced an intermediate encoding, because this allows us to concentrate first on the identification of source term communication partners within an emulation and postpone at least some part of the complicated protocol guiding the flow of requests to the final encoding. By separating these two concepts and introducing them step by step, we hope to provide more intuition on our encoding function. Moreover, the intermediate encoding is interesting in its own, because it represents a good and distributability-preserving encoding from π_{m} into π_{p}. At least we can use it to

validate our formulation of the novel criterion to measure preservation of distributability in Section 3.4. The final encoding is presented in Section 5.2.4

Note that within this chapter we only presented the encodings, but of course we also have to prove their correctness with respect to the required quality criteria, i.e., to the general framework of Section 3.3. We do so and also discuss the main properties of these encodings in the next chapter.

In Section 5.3 we showed how an alternative of the concept behind the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ can be implemented into an adapted version of this encoding. We observed that the separation of the concept of an encoding and the implementation of this concept within the encoding function helps us to present an encoding and to adapt it. Already [Nes00] points out that the consideration of mixed choice in contrast to separate choice is a potential source of deadlock. The proof of Lemma 4.2.17 reveals two solutions to this problem. Either one has to order some capabilities necessary to emulate source term steps—in the presented encoding that are the sum locks—or one has to temporary block some alternative emulation attempts. The first solution was already discussed in [Nes00], but there such an ordering was assumed to be given in advance which violates compositionality. Our encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ generate this ordering on its own during emulation attempts. The second solution leads to the adapted version of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ presented in Section 5.3. In fact, this encoding was our first attempt to encode mixed choice with respect to the criteria of Section 3.3. However, the other attempt, i.e., that deadlocks can be avoided by the implementation of an algorithm to order sum locks, seems to be more intuitive for us. So, we concentrate on this solution.

In Section 5.4 we reviewed a discussion of [Gor10a] on the quality of the composition of two good encodings. Unfortunately, the composition of two good encodings does not necessarily result in a good encoding. The investigation of methods to overcome or to circumvent this drawback are an interesting topic of further research.

Moreover, note that the separation of the concept of an encoding and the implementation of this concept can also help to transfer the main idea of an encoding to another pair of a source and a target language. As example consider the well-known encoding of [FG96, Fou98] from the asynchronous pi-calculus into the join-calculus. As discussed in Section 4.3, this encoding cannot preserve distributability, but it can surely be accepted as a reasonable encoding that (together with the encoding for the opposite direction) shows that π_{a} and J have the same general expressive power. Unfortunately it does not fit into the general framework of Gorla presented in Section 3.3, because as a two-level encoding it violates compositionality. Instead it consists of an inner compositional encoding surrounded by a fixed context that is parametrised on the free names of the source term. This surrounding context introduces so-called firewalls. However, as we conjecture, this encoding can be turned into a good encoding by transferring the main concept of the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. The firewalls are necessary to allow for the emulation of source term communications on free names. This is not necessary if—as it is done in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ over the requests—source term communications are not emulated by a communication on the respective translations of the source term names, but by a communication on a fresh name introduced and maintained by the encoding function. Accordingly, we suggest to replace the concept of firewalls by the way source term communications

partners are identified and their communication is emulated in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, i.e., we suggest to transfer the concepts of the requests and the protocol assumed with the encoding of the parallel operator into an encoding from π_{a} into J. As we conjecture, this results in a good encoding that satisfies all criteria of the general framework. However, by transferring the main concept of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, of course, also its drawbacks are transferred. In particular, we suppose that to obtain a good encoding from π_{a} into J the target language has again to contain the match prefix.

6. Properties of Encodings

The main purpose of this chapter is the discussion of properties of encoding functions and their influence on the quality of an encoding as well as to prove correctness of the encodings of Chapter 5 with respect to the criteria of the general framework of Section 3.3. Apart from these criteria we discuss properties related to the steps of an emulation, the translation of observables, and junk.

In Section 6.1 we start with the analysis of the structural criteria. Structural criteria like compositionality or even the homomorphic translation of some operator may be hard to achieve within an encoding—in fact these requirements strongly limit the possibilities to achieve an encoding function—but they are in general easy to prove. Compositionality for instance can be shown by observation. Indeed we are also able to define a setting in which also name invariance can be checked by observation. Accordingly, the proofs of the structural criteria for the considered encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ are easy.

By contrast, proving the semantic criteria and in particular proving operational correspondence turns out to be very elaborate. To reduce the complexity of the proofs we introduce some type systems in Section 6.2. Type systems are a frequently used technique to reason about process terms [SW01]. In contrast to for example equivalence relations, type systems allow to formalise static properties of process terms without an exhaustive analysis of the state space reachable from the term. Since the three considered encodings are encodings of choice in the pi-calculus, we mainly focus on type systems for the pi-calculus. In particular we use the type systems to formalise the strict name schema used by the encoding functions to separate between links that are introduced for different purposes and prove a confluence property for some of these links.

In Section 6.3 we use the information gained from the type systems to abbreviate some of the proofs of the semantic criteria in the general framework. Furthermore, we discuss other semantic properties of encoding functions in this section. In particular we discuss the kind of steps of an emulation and how a distinction of these steps can help to reason about the reductions of target terms. Then we fix the relation between source term observables and their translation by the encoding functions within so-called *translated observables* and use them to define equivalences to reason about the target terms. Moreover, we discuss junk or garbage that is introduced during emulation attempts. Usually two kinds of junks are distinguished: *inactive junk*, i.e., a junk that remains which does not perform further reductions on its own nor interact with the surrounding target terms, and *active junk*. Of course, proving an encoding to be good requires to prove that its active junks do no harm, i.e., do not influence the abstract behaviour of the target term. However, the presented encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ induce the consideration of a second dimension of junk, namely *observable* and *unobservable junk*. In most cases developers of encodings make sure that all produced junk is unobservable,

i.e., using the standard notions of observables for the target language neither the steps on junk nor the junk itself is observable. Unfortunately, as for the presented encodings, it is not always possible to define the encoding function such that all produced junk is unobservable. Ideally $\asymp$—the equivalence on target terms whose purpose is to abstract from junks and modulo which we have to prove operational correspondence—is one of the well-known standard equivalences of the target calculus, which is usually some kind of bisimulation. In the case of π_{a}, asynchronous barbed bisimulation $\dot{\approx}_{\mathrm{a}}$ can be considered as (one of) its standard equivalences [MS92]. The completion of an emulation changes positive instantiations of sum locks into negative ones and so influences the translated observables, but the corresponding requests—in the case of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$—or in- and outputs of other branches—in the case of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$—remain as observable junk. While active junks often aggravates the proof of correctness of an encoding, due to intricate proofs to show that they do no harm, observable junks turn out to be even worse for an encoding, because they prevent from the use of standard equivalences to describe the abstract behaviour of a target term. In Section 6.3 we show how to deal with this problem in order to nonetheless prove the correctness of the considered encodings.

In Section 6.4 we consider correctness with respect to the additional domain-specific criterion introduced in Section 3.4 to measure preservation of distributability. In particular we prove that $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ preserve distributability and explain why $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ does not.

Note that, due to the complexity of the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, some of the following proofs become rather lengthy. This holds in particular for some of the inductions. We postpone some of these straightforward inductions to the appendix.

6.1. Structural Criteria

The five criteria of Gorla are divided into two structural and three semantic criteria. Structural criteria describe the structure of the encoding, whereas the semantic criteria describe how it should behave or more precisely how the encoded terms should behave. However, also structural criteria influence (at least indirectly) the behaviour of encoded terms. Thus, there is no strict borderline between syntactic and semantic criteria. However, a main difference between these two kinds of criteria is that structural criteria can usually be proven by analysing only the syntax of encoded terms, i.e., by analysing the encoding function, while to prove semantic criteria it is necessary to analyse the behaviour of target terms, i.e., their executions, with respect to the behaviour of the original source terms.

In general, structural criteria are easy to prove but often very hard to achieve. A very good example is the homomorphic translation of an operator. It is the strictest structural criterion presented in Chapter 3 and maybe the strictest criterion at all that still allows for non trivial and meaningful encodings. It completely determines how the encoding of the respective operator has to look like and, thus, can always be checked by observation. Moreover, the homomorphic translation of an operator usually ensures the correctness of this part of the encoding function also with respect to all other criteria,

because it intuitively states that the encoding function is not allowed to change the use of the respective operator. So, homomorphically translated operators significantly ease the proof of correctness of an encoding. On the other side, the homomorphic translation of an operator is an extremely strict criterion. It requires that the respective operator is part of both the source and the target language. Hence, it can usually only be applied within the same family of process calculi. An exception is the homomorphic translation of the parallel operator, because we assume that this operator is introduced by any process calculus in Section 2.1. Note that similar requirements can be found for example in [Gor10b] or [VPP07]. As an example, consider the encoding function $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ in Figure 5.1 at Page 112. It is easy to observe that restriction, the parallel operator, and success are translated homomorphically. Of course, proving a separation result based on this criterion is more elaborate, because we have to show that under the assumption of e.g. the homomorphic translation of the parallel operator there is no encoding satisfying the other required properties (see Section 4.2.1). But even in this case—e.g. by comparing the separation results in Section 4.2.1 and Section 4.2.2—we observe that a stricter structural criterion leads to easier proofs. In the current case, i.e., in the general framework, we have to consider compositionality and name invariance.

6.1.1. Compositionality

Compositionality is strictly weaker than the homomorphic translation of an operator. An encoding is compositional if it defines a fixed context for each operator including holes for the translation of its parameters. By Definition 3.3.1 of compositionality,

> The encoding $[\![\cdot]\!]$ is *compositional* if, for every operator $\mathbf{op} : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{S}}$ of $\mathcal{L}_{\mathrm{S}}$ and for every subset of names N, there exists a context $\mathcal{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{T}}$ such that, for all $x_1, \ldots, x_n \in \mathcal{N}$ and all $S_1, \ldots, S_m \in \mathcal{P}_{\mathrm{S}}$ with $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$, it holds that
>
> $$[\![\, \mathbf{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m) \,]\!] = \mathcal{C}^N_{\mathbf{op}}(x_1, \ldots, x_n, [\![\, S_1 \,]\!], \ldots, [\![\, S_m \,]\!]) \,.$$

the context is allowed to depend on the free names of the subterm-parameters. Again, that an encoding satisfies this criterion can be checked easily by analysing the definition of the encoding function, i.e., by observation. Consider the definition of the encoding of a single operator. It has to be possible to separate the right hand side of such a definition into a context possibly parametrised on the set N with exactly one hole for each parameter of the operator and within these holes there has to be the names and the encodings of the subterms used as parameters of the operator at the left hand side. We check this condition by analysing whether (1) the right hand side of a definition is a term of the target language, i.e., only target language operators are used and the encoded subterms (if there are any) appear as parameter of a target language operator at a position on which a subterm is required[1], (2) the only allowed parameter on the right hand side of a definition is the set N, and (3) for any parameter of the source

[1] Note that, if the source language operator on the left hand side requires only a single parameter, the right hand side can be defined by (the encoding of) this parameter without surrounding context.

language operator at the left, there is exactly one occurrence of the respective name or the translation of the respective subterm on the right hand side, i.e., we have to count these occurrences.

Analysing the encoding functions in Figure 5.1, Figure 5.4, and Figure 5.8, we observe that none of the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ uses the possibility to parametrise the contexts by the set N of the free names of the subterms or any other parameter. Hence, the second condition is satisfied trivially for all three encoding functions. Also, all encodings satisfy obviously the first condition. However, with respect to the very strict interpretation of the compositionality criterion given above, none of the encodings satisfies the last condition, because they use names that are parameters on the left hand side several times on the right hand side, as for instance the names y and x in the encoding of input guarded terms of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ in Figure 5.1.

$$
\begin{aligned}
[\![\, \boldsymbol{y}(\boldsymbol{x}) \,.P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \;\triangleq\; (\nu r)\, \big(\overline{r} \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{y})\big(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{x})\big)\,. \\
\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
\mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
\mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{y})}\big\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{x})\big\rangle\,\big)
\end{aligned}
$$

Fortunately, it is simple to repair this shortcoming. All we need to do is to introduce a mechanism to copy names, as in

$$
\begin{aligned}
[\![\, \boldsymbol{y}(\boldsymbol{x}) \,.P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \;\triangleq\; (\nu r, \boldsymbol{u})\, \big(\overline{\boldsymbol{u}}\langle\varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{y}), \varphi^{\mathrm{s}}_{\mathrm{a}}(\boldsymbol{x})\rangle \mid \boldsymbol{u}\big(\boldsymbol{y'}, \boldsymbol{x'}\big).\big(\overline{r} \mid r^{\star}.\boldsymbol{y'}\big(l', s, \boldsymbol{x'}\big)\,. \\
\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
\mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
\mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\boldsymbol{y'}}\big\langle l', s, \boldsymbol{x'}\big\rangle\,\big)\big)
\end{aligned}
$$

Note that this mechanism introduces—due to the unfolding of polyadic communication—exactly five additional internal and encapsulated steps for each source term input guard but, except for these steps, it does not change the behaviour of the term. Also note that none of the encodings presented in Chapter 5 is prompt (Definition 3.2.1). Thus, we can ignore these additional steps. Moreover, we observe that this kind of duplication of a name is possible in all name passing calculi. Thus, in order to simplify the presentation of encoding functions, we allow the duplication of names in name passing calculi and regard corresponding encoding functions nonetheless as compositional.

Observation 6.1.1. The encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ are compositional.

Consequently, in higher order languages, where terms can be transmitted, we also allow to copy the subterm-parameters. Moreover, in calculi that implement a context-sensitive guard that is either unobservable or can be restricted, we also allow to omit names or subterms that are parameters on the left hand side. Again, we can convert an encoding omitting a parameter into a strict compositional encoding by adding a subterm with that parameter, for example in parallel to the rest of the right hand side, that is

guarded such that this guard can never reduce and is not observable with respect to the chosen equivalence $\asymp$.

Let us have a closer look at the contexts introduced by $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. In the encodings of restriction and success the context is used only to translate source term names according to the renaming policy. Apart from that the encoding of these operators are homomorphic. The encoding of the choice operator introduces a positive instantiation of a fresh sum lock and splits up the encodings of the branches in parallel, because there is no choice operator in the target language. Therefore, of course we have to consider the choice operator as binary with an index set and a set or list of its branches as parameters. We left the question whether there is an encoding from π_{m} with the binary choice operator into $\pi^{=}_{\mathrm{a}}$ as an open question to further research. The encodings of input and output guarded terms and the encoding of terms guarded by τ introduce rather small contexts. However, in the case of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ the contexts introduced to translate the binary parallel operator and replicated input are rather complicated and huge. Remember that we claim in Section 2.1 that the parallel operator is binary. Comparing its encoding with the encoding of the choice operator, we observe that this claim is crucial, because it forbids the introduction of a global coordinator for all parallel terms as the sum lock is for all the branches of a choice.

6.1.2. Name Invariance

Name invariance ensures that the encoding function does not rely on specific names of source terms. Hence, it forbids that encodings are designed against or for some special examples or counterexamples. To obtain name invariance the renaming policy can be of great assistance. It prevents conflicts between source term names and names introduced by the encoding function by translating source term names into fresh names if necessary. Indeed, a strict use of the renaming policy together with the preservation of bound occurrences of source term names suffices to ensure name invariance of compositional encodings, where strict means that all source term names and all names introduced by the encoding function are clearly separated by the renaming policy.

Definition 6.1.2 (Strict Renaming)**.** Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, and $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ be an encoding from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ with the renaming policy $\varphi_{[\![ ]\!]} : \mathcal{N} \to \mathcal{N}^{l}$. Moreover, let $[\![\cdot]\!]$ be compositional. The encoding make *strict use of the renaming policy*, if

1. the renaming policy is applied on all names at the right hand side that appear also on the left hand side of a definition, i.e., on all source term names, but on no other name, and,

2. for all names $n \in \mathcal{N}$ that appear at the right hand side of a definition but not at the left hand side, the renaming policy ensures that $\forall m \in \mathcal{N} \,.\, \varphi_{[\![ ]\!]}(m) \neq n$.

Note that the combination of these two conditions allows an encoding to forget source term names, i.e., there are names at the left hand side of a definition such that neither the name itself nor its translation occurs at the right hand side.

Definition 6.1.3 (Preservation of Binding)**.** Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$, $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, and $[\![\, \cdot \,]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ be an encoding from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ with the renaming policy $\varphi_{[\![\,]\!]} : \mathcal{N} \to \mathcal{N}^l$. Moreover, let $[\![\, \cdot \,]\!]$ be compositional. The encoding *preserves the binding of names*, if, for all names $n \in \mathcal{N}$ that are bounded at the left hand side of a definition, all occurrences of n or its translation $\varphi_{[\![\,]\!]}(n)$ are bound on the right hand side.

A strict use of the renaming policy ensures that there are no clashes between source term names and names introduced by the encoding function. Because of that, it allows to translate substitutions on source term names into substitutions on target terms without the risk that names introduced by the encoding function are accidentally captured. The preservation of the binding of source term names is necessary to ensure that the substitution on source terms and its translation on target terms behave similarly.

Lemma 6.1.4. *Every compositional encoding that makes strict use of the renaming policy and preserves the binding of names is name invariant.*

Proof. Let $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, and $[\![\, \cdot \,]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ be an encoding from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ with the renaming policy $\varphi_{[\![\,]\!]} : \mathcal{N} \to \mathcal{N}^l$.

By Definition 3.3.3 it suffice to show, that:

$$\forall \sigma : \mathcal{N} \to \mathcal{N} \,.\, \exists \sigma' : \mathcal{N} \to \mathcal{N} \,.\, \forall S \in \mathcal{P}_{\mathrm{S}} \,.\, [\![\, \sigma\,(S) \,]\!] \equiv_\alpha \sigma'\,([\![\, S \,]\!]) \wedge \forall z \in \mathcal{N} \,.\, \varphi_{[\![\,]\!]}(\sigma\,(z)) = \sigma'\,\big(\varphi_{[\![\,]\!]}(z)\big)$$

We proceed with an induction over the structure of S, i.e., over the syntax of the source language. Without loss of generality let, for the base case as well as for the induction step, $\sigma = \{\, {}^{y_1}/_{x_1}, \ldots {}^{y_n}/_{x_n} \,\}$ for some $n \in \mathbb{N}$ and some names $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathcal{N}$. We can choose $\sigma' \triangleq \{\, {}^{\varphi_{[\![\,]\!]}(y_1)}/_{\varphi_{[\![\,]\!]}(x_1)}, \ldots, {}^{\varphi_{[\![\,]\!]}(y_n)}/_{\varphi_{[\![\,]\!]}(x_n)} \,\}$, where if $l > 1$ then each ${}^{\varphi_{[\![\,]\!]}(y_1)}/_{\varphi_{[\![\,]\!]}(x_1)}$ represents ${}^{\varphi_{[\![\,]\!]}(y_1).1}/_{\varphi_{[\![\,]\!]}(x_1).1}, \ldots, {}^{\varphi_{[\![\,]\!]}(y_1).l}/_{\varphi_{[\![\,]\!]}(x_1).l}$. So, $\forall z \in \mathcal{N} \,.\, \varphi_{[\![\,]\!]}(\sigma\,(z)) = \sigma'\,\big(\varphi_{[\![\,]\!]}(z)\big)$.

Base Case: The constants of a process calculus are its operators without subprocesses as parameters. Although in most of the process calculi the constants have no parameters at all, as for instance the constants $\mathsf{0}$ and $\checkmark$ in π_{m}, we do not forbid that constants are parametrised on names. Let us consider an arbitrary such operator $S = \mathrm{op}\,(a_1, \ldots, a_n)$ for some names $a_1, \ldots, a_n \in \mathcal{N}$. By compositionality, the encoding of this operator is given by $[\![\, \mathrm{op}\,(a_1, \ldots, a_n) \,]\!] = \mathcal{C}^{\emptyset}_{\mathrm{op}}(a_1, \ldots, a_n)$ for some context $\mathcal{C}^{\emptyset}_{\mathrm{op}}([\cdot]_1, \ldots, [\cdot]_n) : \mathcal{N}^n \to \mathcal{P}_{\mathrm{T}}$. Since there are no subterms and thus no free names of subterms, this context is not parametrised on names. Without loss of generality let us assume that for all $1 \leq i \leq n$ either all occurrences of a_i are free or all occurrences of a_i are bound in $\mathrm{op}\,(a_1, \ldots, a_n)$, because else the operator can be simply replaced by an operator with additional names separating the bound and unbound occurrences. Thus, $\sigma\,(\mathrm{op}\,(a_1, \ldots, a_n)) \equiv_\alpha \mathrm{op}\,(b_1, \ldots, b_n)$, where, for all $1 \leq i \leq n$, if $a_i \in \mathsf{fn}(\mathrm{op}\,(a_1, \ldots, a_n))$ then $b_i = \sigma\,(a_i)$ and else b_i is either a_i or a fresh name to avoid a name clash if $\sigma\,(a_j) = a_i$ for some $1 \leq j \leq n$ with $j \neq i$.

$$[\![\, \sigma\,(S) \,]\!] = [\![\, \sigma\,(\mathrm{op}\,(a_1, \ldots, a_n)) \,]\!] \equiv_\alpha [\![\, \mathrm{op}\,(b_1, \ldots, b_n) \,]\!] = \mathcal{C}^{\emptyset}_{\mathrm{op}}(b_1, \ldots, b_n)$$

Because of the first condition of Definition 6.1.2, all occurrences of the names $b_1, \ldots, b_n$ are translated by the renaming policy, i.e., appear in the target term as $\varphi_{[\![\,]\!]}(b_1), \ldots, \varphi_{[\![\,]\!]}(b_n)$. Moreover, $\varphi_{[\![\,]\!]}(\sigma(a_i)) = \sigma'\left(\varphi_{[\![\,]\!]}(a_i)\right)$ for all $1 \leq i \leq n$, because $\forall z \in \mathcal{N} \,.\, \varphi_{[\![\,]\!]}(\sigma(z)) = \sigma'\left(\varphi_{[\![\,]\!]}(z)\right)$. Note that, by the preservation of the binding of names (Definition 6.1.3), for all $1 \leq i \leq n$ such that the translated name $\varphi_{[\![\,]\!]}(b_i)$ is free in $\mathcal{C}^{\emptyset}_{\mathrm{op}}(b_1, \ldots, b_n)$ the name a_i is free in $\mathrm{op}(a_1, \ldots, a_n)$ and $\varphi_{[\![\,]\!]}(b_i) = \varphi_{[\![\,]\!]}(\sigma(a_i))$. Hence, by the second condition of Definition 6.1.2, all free occurrences of translated names in $\mathcal{C}^{\emptyset}_{\mathrm{op}}(b_1, \ldots, b_n)$ are of the form $\varphi_{[\![\,]\!]}(\sigma(a_i))$ for some $1 \leq i \leq n$ and we substitute them by $\sigma'\left(\varphi_{[\![\,]\!]}(a_i)\right)$. Again this may cause some α-conversion to avoid name capture if $\sigma'\left(\varphi_{[\![\,]\!]}(a_i)\right) = \varphi_{[\![\,]\!]}(b_j)$ for some name $\varphi_{[\![\,]\!]}(b_j)$ bound in $\mathcal{C}^{\emptyset}_{\mathrm{op}}(b_1, \ldots, b_n)$. Since all names in σ' are in the co-domain of $\varphi_{[\![\,]\!]}$ but, by the second condition of Definition 6.1.2, no name of $\mathcal{C}^{\emptyset}_{\mathrm{op}}([\cdot]_1, \ldots, [\cdot]_n)$ is in this set, σ' has no effect on the context but only on the translated source names. Hence, we can pull the substitution σ' outwards, i.e.,:

$$[\![\, \sigma(S) \,]\!] \equiv_\alpha \mathcal{C}^{\emptyset}_{\mathrm{op}}(b_1, \ldots, b_n) \equiv_\alpha \sigma'\left(\mathcal{C}^{\emptyset}_{\mathrm{op}}(a_1, \ldots, a_n)\right) = \sigma'([\![\, S \,]\!])$$

Induction Hypothesis:

$$\forall S \in \mathcal{P}_{\mathrm{S}} \,.\, [\![\, \sigma(S) \,]\!] \equiv_\alpha \sigma'([\![\, S \,]\!]) \tag{IH}$$

Induction Step: Let $S = \mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m)$ for some names $a_1, \ldots, a_n \in \mathcal{N}$ and some source terms $S_1, \ldots, S_m \in \mathcal{P}_{\mathrm{S}}$ when op is an arbitrary operator of the source language that is not a constant, i.e., $m > 0$. By compositionality, the encoding of this operator is given by $[\![\, \mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m) \,]\!] = \mathcal{C}^{N}_{\mathrm{op}}(a_1, \ldots, a_n, [\![\, S_1 \,]\!], \ldots, [\![\, S_m \,]\!])$ for some $\mathcal{C}^{N}_{\mathrm{op}}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{T}}$ and $N = \mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m)$. Again, without loss of generality let us assume that there are no name clashes in $\mathrm{op}(a_1, \ldots, a_n, [\cdot]_{n+1}, \ldots, [\cdot]_{n+m})$, i.e., the sets of free and bound names of $\mathrm{op}(a_1, \ldots, a_n, [\cdot]_{n+1}, \ldots, [\cdot]_{n+m})$ are disjoint in this partially instantiated context. We want to pull the substitution σ inside. Therefore, we have to take care of the free names of the subterms $S_1, \ldots, S_m$ that are bound by the operator. Let us denote these names by $v_1, \ldots, v_k \in \mathcal{N}$, i.e., $(\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m)) \cap \mathsf{bn}(\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m)) = \{\, v_1, \ldots, v_k \,\}$. To pull σ inside, we substitute these names with fresh names. Accordingly, let $w_1, \ldots, w_k \in \mathcal{N}$ be fresh names, i.e., $\{\, w_1, \ldots, w_k \,\} \cap (\mathsf{n}(\sigma) \cup \mathsf{n}(\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m))) = \emptyset$ and the $w_1, \ldots, w_k$ are pairwise different. Thus, $\varphi_{[\![\,]\!]}(w_i) \notin \mathsf{n}(\sigma')$ for all $1 \leq i \leq k$. Without loss of generality let us assume that $\{\, v_1, \ldots, v_k \,\} \subseteq \{\, a_1, \ldots, a_n \,\}$, i.e., that the operator does not bind fixed names, otherwise we can replace the operator by an operator with additional parameters covering these fixed names. Then

$$\sigma(\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m)) \equiv_\alpha \mathrm{op}(b_1, \ldots, b_n, \sigma(\theta(S_1)), \ldots, \sigma(\theta(S_m))),$$

where $\theta = \{ {}^{w_1}/_{v_1}, \ldots, {}^{w_k}/_{v_k} \}$ and, for all $1 \leq i \leq n$,

$$b_i = \begin{cases} \sigma(a_i) & \text{, if } a_i \in \mathsf{fn}(\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m)) \\ \theta(a_i) & \text{, else if } a_i \in \{ v_1, \ldots, v_k \} \\ f_i & \text{, else if } \exists j \in \{ 1, \ldots, n \} \,.\, a_i = \sigma(a_j) \\ a_i & \text{, else} \end{cases}$$

where all f_i are fresh names.

$$\begin{aligned} [\![\sigma(S)]\!] &= [\![\sigma(\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m))]\!] \\ &\equiv_\alpha [\![\mathrm{op}(b_1, \ldots, b_n, \sigma(\theta(S_1)), \ldots, \sigma(\theta(S_m)))]\!] \\ &= \mathcal{C}_{\mathrm{op}}^{\sigma(\theta(N))}(b_1, \ldots, b_n, [\![\sigma(\theta(S_1))]\!], \ldots, [\![\sigma(\theta(S_m))]\!]) \end{aligned}$$

Because of the first condition of Definition 6.1.2, all occurrences of the names $b_1, \ldots, b_n$ are translated by the renaming policy, i.e., appear in the target term as $\varphi_{[\![]\!]}(b_1), \ldots, \varphi_{[\![]\!]}(b_n)$. Moreover, $\varphi_{[\![]\!]}(\sigma(a_i)) = \sigma'(\varphi_{[\![]\!]}(a_i))$ for all $1 \leq i \leq n$, because $\forall z \in \mathcal{N} \,.\, \varphi_{[\![]\!]}(\sigma(z)) = \sigma'(\varphi_{[\![]\!]}(z))$. Note that, by the preservation of the binding of names (Definition 6.1.3), for all $1 \leq i \leq n$ such that the translated name $\varphi_{[\![]\!]}(b_i)$ is free in $\mathcal{C}_{\mathrm{op}}^{\sigma(\theta(N))}(b_1, \ldots, b_n, [\![\sigma(\theta(S_1))]\!], \ldots, [\![\sigma(\theta(S_m))]\!])$, the name a_i is free in $\mathrm{op}(a_1, \ldots, a_n)$ and $\varphi_{[\![]\!]}(b_i) = \varphi_{[\![]\!]}(\sigma(a_i))$. Hence, by the second condition of Definition 6.1.2, all free occurrences of translated names in the target term are of the form $\varphi_{[\![]\!]}(\sigma(a_i))$ for some $1 \leq i \leq n$ and we substitute them by $\sigma'(\varphi_{[\![]\!]}(a_i))$. Again this may cause some α-conversion to avoid name capture. Since all names in σ' are in the co-domain of $\varphi_{[\![]\!]}$ but, by the second condition of Definition 6.1.2, no name of $\mathcal{C}_{\mathrm{op}}^{\sigma(\theta(N))}([\cdot]_1, \ldots, [\cdot]_{n+m})$ or $\mathcal{C}_{\mathrm{op}}^{\theta(N)}([\cdot]_1, \ldots, [\cdot]_{n+m})$ is in this set, σ' has no effect on the context but only on the translated source names. Because of that, we can pull the substitution σ' outwards such that

$$\begin{aligned} [\![\sigma(S)]\!] &\equiv_\alpha \mathcal{C}_{\mathrm{op}}^{\sigma(\theta(N))}(b_1, \ldots, b_n, [\![\sigma(\theta(S_1))]\!], \ldots, [\![\sigma(\theta(S_m))]\!]) \\ &\equiv_\alpha \mathcal{C}_{\mathrm{op}}^{\sigma(\theta(N))}(b_1, \ldots, b_n, \sigma'([\![\theta(S_1)]\!]), \ldots, \sigma'([\![\theta(S_m)]\!])) \qquad \text{by (IH)} \\ &\equiv_\alpha \sigma'\left(\mathcal{C}_{\mathrm{op}}^{\theta(N)}(b'_1, \ldots, b'_n, [\![\theta(S_1)]\!], \ldots, [\![\theta(S_m)]\!])\right), \end{aligned}$$

where b'_i is $\theta(a_i)$ if $a_i \in \{ v_1, \ldots, v_k \}$, and else $b'_i = a_i$, for all $1 \leq i \leq n$. By undoing θ, we conclude:

$$\begin{aligned} [\![\sigma(S)]\!] &\equiv_\alpha \sigma'\left(\mathcal{C}_{\mathrm{op}}^{\theta(N)}(b'_1, \ldots, b'_n, [\![\theta(S_1)]\!], \ldots, [\![\theta(S_m)]\!])\right) \\ &\equiv_\alpha \sigma'(\mathcal{C}_{\mathrm{op}}^{N}(a_1, \ldots, a_n, [\![S_1]\!], \ldots, [\![S_m]\!])) \\ &= \sigma'([\![\mathrm{op}(a_1, \ldots, a_n, S_1, \ldots, S_m)]\!]) = \sigma'([\![S]\!]) \end{aligned}$$

□

Analysing the encoding functions in Figure 5.1, Figure 5.4, and Figure 5.8, we observe that all of three encodings $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$, and $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ make strict use of the renaming policy and preserve the binding of names. Thus, they are name invariant.

Corollary 6.1.5. *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *are name invariant.*

Of course, not every name invariant encoding makes strict use of the renaming policy, preserves the binding of names, or even is compositional. For instance, there is no need to guard names introduced by the encoding function with the help of the renaming policy against source term names, if these names are introduced restricted and encapsulated such that already the encoding function ensures that there are no clashes with source term names. In cases like that, the proof of Lemma 6.1.4 can serve as guideline to show name invariance: (1) find an appropriate definition of σ', (2) perform an induction on the structure of source terms, (3) rename the names free in the subterm-parameters but bound by the operator into fresh names to avoid capture, and (4) then, in each case, (a) pull σ inside the operator, (b) apply the encoding function, and (c) pull σ' outside.

If the encoding does not preserve the binding of source term names, σ' can be chosen by first restricting σ to the free names of the source term S. But then σ' is chosen with respect to S, i.e., we can only show that for all combinations of S and σ we find an appropriate σ' ($\forall S \,.\, \forall \sigma \,.\, \exists \sigma'$ instead of $\forall \sigma \,.\, \exists \sigma' \,.\, \forall S \,.\,$). In contrast, σ' in Lemma 6.1.4 depends only on σ and the respective renaming policy. Moreover, we observe that for all considered encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ we can prove the first case of name invariance in Definition 3.3.3 for all kinds of substitutions σ, i.e., it suffice to consider equivalence modulo α-conversion.

Corollary 6.1.6 (Encoding substitutions)**.** *For all substitutions* $\sigma = \{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}$ *it holds that*

$$\forall S \in \mathcal{P}_{\mathrm{s}} \,.\, [\![\sigma\,(S)]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv_\alpha \{ {}^{\varphi^{\mathrm{s}}_{\mathrm{a}}(y_1)}/_{\varphi^{\mathrm{s}}_{\mathrm{a}}(x_1)}, \ldots, {}^{\varphi^{\mathrm{s}}_{\mathrm{a}}(y_n)}/_{\varphi^{\mathrm{s}}_{\mathrm{a}}(x_n)} \} ([\![S]\!]^{\mathrm{s}}_{\mathrm{a}}),$$
$$\forall S \in \mathcal{P}_{\mathrm{m}} \,.\, [\![\sigma\,(S)]\!]^{\mathrm{m}}_{\mathrm{p}} \equiv_\alpha \{ {}^{\varphi^{\mathrm{m}}_{\mathrm{p}}(y_1)}/_{\varphi^{\mathrm{m}}_{\mathrm{p}}(x_1)}, \ldots, {}^{\varphi^{\mathrm{m}}_{\mathrm{p}}(y_n)}/_{\varphi^{\mathrm{m}}_{\mathrm{p}}(x_n)} \} \left([\![S]\!]^{\mathrm{m}}_{\mathrm{p}}\right), \text{ and}$$
$$\forall S \in \mathcal{P}_{\mathrm{m}} \,.\, [\![\sigma\,(S)]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv_\alpha \{ {}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(y_1)}/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x_1)}, \ldots, {}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(y_n)}/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x_n)} \} ([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}).$$

6.2. Type Systems

Another static way to analyse encodings are type systems. Type systems for the pi-calculus were already introduced by Milner in [Mil93b]. They provide a mechanism to classify elements of a language. In the case of the pi-calculus types are usually assigned to the names as e.g. in [San92, Mil93b, Tur96, PS96, SW01]. Like a process calculus, a type systems consists of a syntax to build types and a set of typing rules. In contrast to process calculi, these rules (usually) do not describe the behaviour of types—in fact most types are static and have no behaviour—instead they describe a property of typed process terms called *well-typed*. A typed process term within a type environment, i.e., a process term augmented with a type for each of its names, is well-typed if its typing agrees with the behaviour of the term. An introduction and overview for type systems in the context of the pi-calculus can e.g. be found in [SW01].

In principle we use type systems for two reasons. (1) they allow us to pick up elements of the target language by their type instead of their name. Since nearly all

names introduced by the encoding functions are restricted, this significantly eases the identification of particular objects in the encoding function. Moreover, types allow us to sum up different terms or capabilities that are introduced for the same purpose and hence do not need to be considered separately. (2) we use type systems to prove some static properties on the general use of some links. We show for example that some links are never used for input but only replicated input and that the use on other links is so limited that we can mostly ignore their existence. Both significantly eases and shortens the considerations and proofs in Section 6.3.

In Section 6.2.1 we review some general concepts of type systems as for example type environments. To introduce type systems we present then a very simple type system in Section 6.2.2. Basically, it is a mixture of the basic type system introduced in [SW01] and an adaptation of the *sortings* introduced in [Mil93b] to type the polyadic version of the pi-calculus. Note that Section 6.2.2 serves as both an introduction to type systems as well as a tutorial on how to derive a type system for an encoding. Nonetheless, already this exemplary type system is of great assistance (at least as intuition) in reasoning about the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, because it exposes the strict name schema used there to introduce names for different purposes and, moreover, proves that the encoding indeed makes strict use of that schema. This allows us to formulate and prove invariants on the respective parts of the protocol underlying the encoding function in Section 6.3.2. In Section 6.2.3 we cover the problem of typing a monadic pi-calculus and, in particular, of typing the translation of polyadic into monadic communication. Therefore, we revisit the type system introduced for this translation in [QW00]. This type system closes the gap between the first introduced type system and the target languages of our encoding functions. In particular we prove that the unfolding of polyadic communication as defined in Section 5.4 preserves the types of the basic type system. Because of that we can mainly ignore the unfolding of polyadic communication in Section 6.3. Section 6.2.4 discusses types with polarities and multiplicities [PS96, KPT99, San99]. Polarities tell us whether a link can be used for output, input, or both, whereas multiplicities describe how often a link can be used. Such information can be used to prove termination properties as deadlock- or divergence-freedom (see e.g. [Kob98]). We use polarities and multiplicities to show a partial confluence property for some of the links introduced by the encoding functions. Partial confluence (compare to Section 4.4) basically means absence of conflicts. This again allows us to abstract from communications on links in some of the proofs of Section 6.3.

6.2.1. Terminology

We mainly follow the Definitions of [SW01] for typing. Accordingly, we distinguish between names used as links—*link types*—and names used as values communicated over links—*value types*. The atom type, denoted as $\mathfrak{v}$, serves as basic and only type of pure values, i.e., each name that is never used as link but only transmitted on links is typed by the atom type. In [SW01] two approaches to build type systems—the by-name and the by-structure approach—are distinguished. In the by-name approach two types are different if they have a different name, where in the by-structure approach

distinguishable types have to differ in their structure, e.g. the type of a link carrying a value of an atom type is distinguished from a link type characterising a link that carries a link that carries a value. Hence, the by-name approach is more flexible, because it allows to distinguish types with the same structure. On the other side, the by-structure approach is mathematically more elegant, because it has a natural interpretation as a logic. The type systems presented in the following are a mixture of both. Basically, we follow the by-structure approach but, in order to gain more information, we distinguish between different variants of the atom type to distinguish structurally equivalent types of names which are introduced for different purposes.

The syntax of a type system consist of three sets: the set of value types $\mathbb{V}$, the set of link types $\mathbb{L}$, and the set of types $\mathbb{T}$ covering at least all value and all link types. Remember that we mainly follow the by-structure approach when building type systems. Accordingly, in [SW01] link names are defined by the grammar $L ::= \sharp(V)$, where the prefix $\sharp$ marks a so-called *connection type*, that is a constructor for link types, followed by the type of the transmitted value which again can be a link type or a value type. We extend this concept to cover polyadic communication, i.e., allow lists of value or link types as arguments of a link type. Usually, we use $V, V', V_1, \ldots$ to range over value types, $L, L', L_1, \ldots$ to range over link types, and $T, T', T_1, \ldots$ to range over types.

The allocation of types to names is done in form of type assignments. Note that, similarly to [SW01], we omit curly brackets when presenting sets of type assignments.

Definition 6.2.1 (Type Assignment)**.** Let $\mathbb{T}$ be the set of types of a type system, $n \in \mathcal{N}$ be a name, and $T \in \mathbb{T}$ be a type. Then the *type assignment* $n:T$ allocates the type T to the name n. Moreover, let $n_1, \ldots, n_m : T$ abbreviate the set of assignments $n_1:T, \ldots, n_m:T$ for some names $n_1, \ldots, n_m \in \mathcal{N}$ (that are pairwise disjoint) and a type $T \in \mathbb{T}$.

Note the side condition that within sets of type assignments the names should be pairwise disjoint. This condition ensures that there is no set with a pair of type assignments that assign different types to a single name. Hence, we require that sets of type assignments are unambiguous. Because of that also type environments, that are simply sets of type assignments, have to be unambiguous.

Definition 6.2.2 (Type Environment)**.** Let $\mathbb{T}$ be the set of types of a type system. Then *type environments* are defined by

$$\Gamma ::= \emptyset \quad | \quad \Gamma, n:T$$

where $n \in \mathcal{N}$ is a name, $T \in \mathbb{T}$ is a type, and $n:T$ is a type assignment. We assume that for all type environments Γ the names of different assignments are disjunct. Moreover, let $\Gamma(n) = T$ if $n : T \in \Gamma$, i.e., if $n : T$ is an assignment in Γ, let $\mathsf{n}(\Gamma)$ return the set of names in Γ, and let Γ_1, Γ_2 denote the union of the environments Γ_1 and Γ_2 under the assumptions that $\Gamma_1(n) = \Gamma_2(n)$ for all names $n \in \mathsf{n}(\Gamma_1) \cap \mathsf{n}(\Gamma_2)$. Consequently, we abbreviate $\emptyset, \Gamma$ by Γ.

We use $\Gamma, \Gamma', \Gamma_1, \ldots$ to range over type environments.

A *typing* of a term (or an encoding function) is a set of assignments that allocate a type to each name of the term (or to each name of the right hand side of each definition). We assume that the typing of a term is given by a typed term and a type environment, where the typed variant of a term is the term augmented with type assignments for all names under restriction and the corresponding type environment provides the types for all free names of the term. All other types, i.e., the types of the arguments of links, can be derived in the by-structure approach from the type of the respective link. The syntax of typed terms can simply be obtained from the syntax of the corresponding language—in the case of encodings this is the respective target language—by replacing the restriction operator, i.e., $(\nu x)\,P$ in the pi-calculus, by its typed version $(\nu x{:}T)\,P$, where $x{:}T$ is a type assignment. Because of that, we do not explicitly distinguish between the untyped syntax of an untyped term and the typed syntax of a typed term, but instead use only the untyped version silently assuming that it is replaced by its typed variant if necessary. Again, we abbreviate $(\nu x_1{:}T_1)\,(\ldots(\nu x_n{:}T_n)\,(P)\ldots)$ by $(\nu x_1{:}T_1,\ldots,x_n{:}T_n)\,P$. For a process calculus $\langle\, \mathcal{P}, \longmapsto \,\rangle$ we denote the set of typed processes by $\mathcal{P}{:}\mathbb{T}$, where $\mathbb{T}$ is the set of types of the respective type system. We write $\mathcal{T}(P)$ to denote the typed variant of a term P with respect to a set of type assignments $\mathcal{T}$ that defines (at least) the types of all restricted names of P. Note that two restrictions of syntactical the same name lead to the same type in $\mathcal{T}(P)$. The typed variant of an encoding $[\![\,\cdot\,]\!]$, denoted by $\mathcal{T}[\![\,\cdot\,]\!]$, is obtained similarly by replacing all restrictions on the right hand side of all definitions by the corresponding typed restriction, where again for all names under restriction $\mathcal{T}$ has to provide a type assignment. Consequently, $(\mathcal{P}{:}\mathbb{T})\!\restriction_{\mathcal{T}[\![\,\cdot\,]\!]}$ denotes the set of typed target terms with respect to the typed encoding $\mathcal{T}[\![\,\cdot\,]\!]$ if $\mathcal{P}$ is the set of processes of the target language of $[\![\,\cdot\,]\!]$. We do not introduce the typed variants of the concepts of free names, bound names, names, structural congruence, or the reduction relation but simply use the corresponding untyped versions instead, because the differences between the typed and untyped versions are trivial.

The property of a typed process or a type assignment to be well-typed with respect to a given type environment is expressed by type judgements.

Definition 6.2.3 (Type Judgement)**.** Let $\mathbb{T}$ be the types of a type system and Γ a type environment in this type system. Then $\Gamma \vdash x{:}T$ and $\Gamma \vdash P$ are *type judgements* if $x{:}T$ is a type assignment for some $x \in \mathcal{N}$ and $T \in \mathbb{T}$ and P is a typed process.

If a type judgement $\Gamma \vdash E$ can be derived by the typing rules of the type system then E is *well-typed with respect to* Γ. In this cases, we also say that the typed process P or the type assignment $x : T$ *respects* Γ, and if $\Gamma \vdash x : T$ that the type T can be derived for x from Γ. Consequently, we denote a typed encoding $\mathcal{T}[\![\,\cdot\,]\!]$ as well-typed with respect to a type environment Γ if each encoded term respects Γ, i.e., if for all S of the respective source language we can derive $\Gamma \vdash \mathcal{T}[\![\,S\,]\!]$. Note that in Section 6.2.3 we extend type judgements with two additional sets. Hence, type judgements in the monadic type system in Section 6.2.3 as well as in the linear type system in Section 6.2.4 have the form $\Gamma;\Delta;\Psi \vdash P$ for a typed term P, a type environment Γ and two auxiliary sets Δ and Ψ.

Typing rules describe the inference of type judgements much the same as reduction rules describe the inference of reduction steps. Axioms allow to derive type judgements for names and constant terms, whereas the remaining rules describe how type judgements of composed terms or constants containing names can be derived from type judgements of their respective subterms and names. Remember that with type systems we allocate types to names. Hence, constants as $\mathsf{0}$ or $\checkmark$ that do not contain any names are always trivially well-typed. Similarly, because the prefix τ does not contain any names, typed processes guarded by τ are usually well-typed if their continuation is well-typed. Because of that, all of the following type systems contain rules similar to:

$$\text{T-Nil}\quad \frac{}{\Gamma \vdash \mathsf{0}} \qquad \text{T-Succ}\quad \frac{}{\Gamma \vdash \checkmark} \qquad \text{T-Tau}\quad \frac{\Gamma \vdash P}{\Gamma \vdash \tau.P}$$

Furthermore, they contain the axiom

$$\text{T-Name}\quad \frac{}{\Gamma, x{:}T \vdash x{:}T}$$

to illustrate that a type assignment $x : T$ can be derived from a type environment Γ', if Γ' already contains the type assignment $x : T$. Type assignments that are not provided by the type environment have to be derived from the typing rules using either the information given by the link types about the parameters of a link or by the typed restrictions. Link types are handled quite different by the following type systems but typed restrictions are processed by a rule or a set of rules comparable to

$$\text{T-Res}\quad \frac{\Gamma, x{:}T \vdash P}{\Gamma \vdash (\nu x{:}T)\, P},$$

i.e., a typed term under a typed restriction is well-typed with respect to a type environment Γ if the unrestricted term is well-typed with respect to the union of Γ and the type assignment provided by the typed restriction.

As mentioned above, type systems are a static way to reason about process terms, i.e., they describe properties of terms without analysing their reductions. Nevertheless, type systems usually define properties that are robust against reduction steps. They do so, because they usually satisfy a so-called *subject reduction* property. Subject reduction states consistency between the typing rules of the type system and the semantics of a calculus. More precisely, we require that well-typedness is preserved by reduction steps, i.e., that the derivative of a typed term is well-typed if the original term is well-typed with respect to the same type environment. Of course, we have to prove subject reduction with respect to the target language of the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Unfortunately, the match prefix in the target language of the last encoding needs special consideration. The problem is that by the reduction rule $\text{Pi-Cong}_{\mathrm{m,s,a,p}}$ and the structural congruence rule $[a = a]\, P \equiv P$ it is possible to introduce fresh free names within reductions for which no typing information is given. Obviously, the so introduced matching prefixes $[a = a]$ do not contribute to the behaviour of target terms and, hence, are ignored. Accordingly, we prove subject reduction only for closed type environments, i.e., type environments that provide a type for all free names of a typed term.

Definition 6.2.4 (Closed Type Environment)**.** Let $\mathbb{T}$ be the types of a type system and Γ a type environment in this type system. Then Γ is *closed* for some typed process P, if $\Gamma(a)$ is defined for all names that are free in P.

To simplify the proof of subject reduction we also consider some other properties of type systems. The *strengthening* property states that a type judgement remains valid if we remove superfluous type assignments from the type environment, i.e., type assignments on names that are not free in the respective term. Similarly, *weakening* describes the property that type judgements remains valid if we add type assignments as long as these new type assignments are not in conflict to the type environment, i.e., do not allocate a type to a name for which the type environment already specifies a different type. Moreover, we consider robustness of type judgements with respect to substitutions, to capture communication steps in the subject reduction lemma, and with respect to structural congruence, to capture the rule $\text{Pi-Cong}_{\text{m,s,a,p}}$.

6.2.2. A Basic Type System

We use a very simple type system to extract some static information on encoded terms. Intuitively, we use types to distinguish the names introduced by the encoding function for different purposes, i.e., to manifest the strict naming scheme of $[\![\cdot]\!]^{\text{s}}_{\text{a}}$, $[\![\cdot]\!]^{\text{m}}_{\text{p}}$, and $[\![\cdot]\!]^{\text{m}}_{\text{a}}$. Moreover, as a side effect, we prove that the representation of polyadic channels is done consistently, by proving that no name of a target term is used with different multiplicities. However, to show that all target terms are well-typed we have to unfold all abbreviations, i.e., to boil down target terms to elements of $\mathcal{P}^{=}_{\text{a}}$ without forwarders, test-constructs, or polyadic communication. Unfortunately, as already pointed out by Milner in [Mil93b], the translation of polyadic communication causes some troubles and requires a rather complicated type system as introduced in [Yos96] or [QW00]. Because of that, we postpone the treatment of polyadic communication (at least of channels with multiplicity greater than one) to Section 6.2.3.

We want to type the encoding functions $[\![\cdot]\!]^{\text{s}}_{\text{a}}$, $[\![\cdot]\!]^{\text{m}}_{\text{p}}$, and $[\![\cdot]\!]^{\text{m}}_{\text{a}}$. Therefore, we determine first what kind of types are necessary and collect the corresponding type assignments in the sets $\mathcal{T}^1_{\text{B}}$, $\mathcal{T}^2_{\text{B}}$, and $\mathcal{T}^3_{\text{B}}$. The typed encodings are then given by $\mathcal{T}^1_{\text{B}}[\![\cdot]\!]^{\text{s}}_{\text{a}}$, $\mathcal{T}^2_{\text{B}}[\![\cdot]\!]^{\text{m}}_{\text{p}}$, and $\mathcal{T}^3_{\text{B}}[\![\cdot]\!]^{\text{m}}_{\text{a}}$, respectively.

If we analyse the encoding function $[\![\cdot]\!]^{\text{m}}_{\text{a}}$, we observe that as part of the underlying algorithm all source term names are used within the encoding only as values but never as links. As consequence, we can type all source term names by the atom type, which throws away all potential requirements on the typing of source terms. However, to obtain more information, we distinguish between different kinds of atom types. The first, $\mathfrak{v}_{\mathfrak{n}}$, is reserved for translations of source term names. Accordingly, $\mathcal{T}^3_{\text{B}}$ contains the type assignments $\varphi^{\text{m}}_{\text{a}}(x), \varphi^{\text{m}}_{\text{a}}(y), \varphi^{\text{m}}_{\text{a}}(z), y, y', z : \mathfrak{v}_{\mathfrak{n}}$ and we obtain

$$\mathcal{T}^3_{\text{B}}[\![\,(\nu x)\,P\,]\!]^{\text{m}}_{\text{a}} \triangleq (\nu\varphi^{\text{m}}_{\text{a}}(x) : \mathfrak{v}_{\mathfrak{n}})\,(\mathcal{T}^3_{\text{B}}[\![\,P\,]\!]^{\text{m}}_{\text{a}})$$

as the typed version of the encoding of restriction. Apart from translated source term names, names that are never used as links are necessary to implement channels of

multiplicity zero, i.e., channels without parameters. Such channels are used to implement boolean values with the links t and f, and to implement a sender lock s. If an instantiation of a sum lock is checked the value true is represented by the message $\overline{t} \overset{\text{Def. 5.4.1}}{=} (\nu v_t)\,\overline{t}\langle v_t\rangle$ and false is represented by $\overline{f} \overset{\text{Def. 5.4.1}}{=} (\nu v_f)\,\overline{f}\langle v_f\rangle$. An instantiation of a sender lock, necessary to unguard the encoded continuation of a source term sender, is implemented by $\overline{s} \overset{\text{Def. 5.4.1}}{=} (\nu v_s)\,\overline{s}\langle v_s\rangle$. Since the auxiliary values v_t, v_f, and v_s are never used as links, we can type them by the atom type. But then the type of true, false, and an instantiation of a sender lock are equally represented by $\sharp(\mathfrak{v})$. So, in order to distinguish boolean values and instantiations of sender locks, we use the three different value types $\mathfrak{v}_\top$, $\mathfrak{v}_\bot$, and $\mathfrak{v}_\mathfrak{s}$ to type the auxiliary names v_t, v_f, and v_s, respectively. Accordingly, we obtain $\sharp(\mathfrak{v}_\top)$ as type of t, $\sharp(\mathfrak{v}_\bot)$ as type of f, and $\sharp(\mathfrak{v}_\mathfrak{s})$ becomes the type of sender locks. To simplify the presentation of types, we introduce abbreviations for some link types, e.g. we abbreviate the type of sender locks by $\mathfrak{s}$.

A positive instantiation of a sum lock $\overline{l}\langle\top\rangle$ is, by Definition 5.1.1, an abbreviation of $l(t,f)\,.\,\overline{t}$, i.e., a sum lock l is a link carrying two values. Following the by-structure approach, sum locks are typed by a link type with the types of t and f as arguments. Thus, $\mathcal{T}^3_{\mathrm{B}}$ contains the type assignments $l, l_1, l_2, l_s, l_r : \sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))$ and the typed version of the translation of a sum is

$$\mathcal{T}^3_{\mathrm{B}}\Big[\!\Big[\sum_{i\in I} \pi_i.P_i \Big]\!\Big]^{\mathrm{m}}_{\mathrm{a}} \triangleq (\nu l:\mathfrak{l}) \left(l(t,f)\,.\,\big((\nu v_t:\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\big) \mid \prod_{i\in I} \mathcal{T}^3_{\mathrm{B}}[\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \right),$$

where $\mathfrak{l}$ is used to abbreviate the type $\sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))$. The counterpart of instantiations of sum locks are test-constructs. Their unfolding into $\mathcal{P}^{=,\sim}_{\mathrm{a}}$ do not introduce new names with respect to the unfolding of instantiations of sum locks. Thus, the typed version of a test-construct is obtained straightforwardly as

$$\mathcal{T}^3_{\mathrm{B}}(\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q) = (\nu t:\sharp(\mathfrak{v}_\top)\,,f:\sharp(\mathfrak{v}_\bot))\,\big(\overline{l}\langle t,f\rangle \mid t.\mathcal{T}^3_{\mathrm{B}}(P) \mid f.\mathcal{T}^3_{\mathrm{B}}(Q)\big)\,.$$

It is also straightforward to determine the type of all other names introduced by the encoding function $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ if we use instead of $\pi^{=}_{\mathrm{a}}$ its variant with polyadic communication as target language and, consequently, do not unfold the abbreviations of inputs or outputs transmitting more than a single value. Figure 6.1 presents an overview of all names and their types for $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Thus, $\mathcal{T}^3_{\mathrm{B}}$ is the set of all type assignments defined in Figure 6.1. We observe, that to type this encoding we need only four different value types and eleven different (polyadic) link types. Interestingly, there are no cyclic dependencies between types, as they result e.g. from a link carrying an argument of its own or a more complex type. There are three reasons for this nice discipline on the use of links in the encoding $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$: (1) the renaming policy $\varphi^{\mathrm{m}}_{\mathrm{a}}$ separates clearly between translated source term names and names introduced by the encoding function, (2) to type the encoding function we can completely ignore the types of source terms, because translated source term names are used as values only, and (3) the algorithm underlying the encoding function is such that all steps necessary to emulate a source term step are directed, i.e., there are no cycles in the flow of information. An advantage of the absence

Description	Names	Type
source term names	$\varphi_{\mathrm{a}}^{\mathrm{m}}(x)\,, \varphi_{\mathrm{a}}^{\mathrm{m}}(y)\,, \varphi_{\mathrm{a}}^{\mathrm{m}}(z)\,,$ y, y', z	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\sharp(\mathfrak{v}_{\top})$
	f	$\sharp(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_{\top})\,, \sharp(\mathfrak{v}_{\bot}))$
sender locks	s	$\mathfrak{s} = \sharp(\mathfrak{v}_{\mathfrak{s}})$
receiver locks	r	$\mathfrak{r} = \sharp(\mathfrak{l}, \mathfrak{l}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}})$
output requests	$p_o, m_o, p_{o,up},$ $m_{o,up}, r_o, r_{o,up}$	$\mathfrak{o} = \sharp(\mathfrak{v}_{\mathfrak{n}}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}})$
input requests	$p_i, m_i, p_{i,up},$ $m_{i,up}, r_i, r_{i,up}$	$\mathfrak{i} = \sharp(\mathfrak{v}_{\mathfrak{n}}, \mathfrak{l}, \mathfrak{r})$
chain locks	c_o	$\sharp(\mathfrak{i})$
	c_i	$\sharp(\mathfrak{o})$
	c_{r1}	$\sharp(\mathfrak{v}_{\mathfrak{n}})$
	c_{r2}	$\sharp(\mathfrak{o}, \mathfrak{i})$

Figure 6.1.: Basic Types in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$.

of such cyclic dependencies in types is that we are not forced to introduce recursive types as explained in Section 6.7 of [SW01]. We conclude that to type $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ we can fix the sets of value and link types to the types given in Table 6.1, i.e., there is no need to consider e.g. all kinds of link types $\sharp(V_1, V_2)$ of multiplicity two for some pair of value or link types V_1 and V_2 but only $\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_{\top})\,, \sharp(\mathfrak{v}_{\bot}))$ and $\sharp(\mathfrak{o}, \mathfrak{i})$.

Within the other two encodings $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ and $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ the translations of source term names are used as links. However, in $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ translated source term names are used only as part of links in polyadic synchronisation. This allows us to cheat a little bit, by capturing the typing information for polyadic synchronisation links within the type of the second part of the link, namely the output tag o or the input tag i. Because of this trick, we can again type all translated source term names by the pure value type $\mathfrak{v}_{\mathfrak{n}}$. Note that this is indeed a trick to ease the typing of the encoding $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$; it is by no means a standard way to type polyadic synchronisation. Apart from the links for polyadic synchronisation the encoding $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ can be typed very similarly to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. They differ mainly in the multiplicity of (second) receiver locks and out- and input requests. Moreover, $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ introduces an additional sender and an additional receiver lock in comparison to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. We observe that these two looks are transmitted as the last two parameters in outputs on second receiver locks r_2 as in procRightOutReq

$$p_o{}^{\star}(y, l_s, s_1, s_2, z)\,.\,(y \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle)$$

Description	Names	Type
source term names	$\varphi_{\mathrm{p}}^{\mathrm{m}}(x)\,, \varphi_{\mathrm{p}}^{\mathrm{m}}(y)\,, \varphi_{\mathrm{p}}^{\mathrm{m}}(z)\,,$ y, z	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\sharp(\mathfrak{v}_{\top})$
	f	$\sharp(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_{\top})\,, \sharp(\mathfrak{v}_{\bot}))$
sender and receiver locks	s_1, r_1, v, w	$\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$
	s_2	$\mathfrak{s} = \sharp(\mathfrak{v}_{\mathfrak{s}})$
	r_2	$\mathfrak{r}' = \sharp(\mathfrak{l}, \mathfrak{l}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))$
output requests	$p_o, p_{o,up}$	$\mathfrak{o}' = \sharp(\mathfrak{v}_{\mathfrak{n}}, \mathfrak{l}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}})$
input requests	$p_i, p_{i,up}$	$\mathfrak{i}' = \sharp(\mathfrak{v}_{\mathfrak{n}}, \mathfrak{l}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \mathfrak{r}')$
tags	o	$\mathfrak{t}_{\mathfrak{o}} = \sharp(\mathfrak{l}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}})$
	i	$\mathfrak{t}_{\mathfrak{i}} = \sharp(\mathfrak{l}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \mathfrak{r}')$

Figure 6.2.: Basic Types in $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$.

and in procRightInReq

$$p_i^{\star}(y, l_r, r_1, r_2)\,.\,(y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{i,up}}\langle y, l_r, r_1, r_2\rangle)\,.$$

The order of these parameters depends on the origin of the respective request in the parallel structure. This parallel structure is captured by the encoding in a static way, i.e., the restriction of the out- and input requests ensures that a left part of a parallel operator encoding can always be identified as left part. It is not possible to destruct the encoded parallel structure by structural congruence or reduction rules. Thus, it is indeed possible to reason about the translation of the parallel structure of source terms within a type systems. But this requires some effort and is beyond the scope of this thesis. We will use invariants instead to reason about the structure of target terms and the information flow of requests along the encoded parallel structure. Since first sender locks as well as first receiver locks are both links carrying no value, we can type them by the same type. This allows us to type also second receiver locks properly without reasoning about the encoded parallel structure within target terms. By the way, this allows us in Definition 5.4.1 to use the same auxiliary value $v_{s,r}$ to unfold polyadic communication of these two links. In order to avoid confusion with translated source term names, booleans, and second sender locks, we type $v_{s,r}$ by a fresh atom type $\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. Accordingly, the type of first sender and first receiver locks becomes $\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$ and second receiver locks are typed by $\mathfrak{r}' = \sharp(\mathfrak{l}, \mathfrak{l}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}}, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))$. The type of the remaining links is then obtained by simply following the by-structure approach. Note that the use and, hence, also the typing of booleans, sum locks, and (second) sender locks is the same

for all three encoding functions. Figure 6.2 presents an overview of all names and their types for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $\mathcal{T}^2_{\mathrm{B}}$ is the set of type assignments specified in this figure.

For the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ we cannot perform such a trick. In the by-structure approach the type of an output $\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle$ on the translation of the source term name y depends on the type of the source term name z. Because of this, we can type $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ only with respect to well-typed source terms. More precisely, we assume that there is a type system for the source language π_{s} of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ with the types $\mathbb{T}_{\mathrm{s}}$ that also follow the by-structure approach such that the types of parameters of a link can be obtained from the respective link type and consider only source terms S that are well-typed within this type systems given a typing that provides a type for all names of S. We denote source terms that satisfy these properties as well-structured.

Definition 6.2.5 (Well-Structured Source)**.** Let $S \in \pi_{\mathrm{s}}$. Without loss of generality let us assume that S is free of name clashes, i.e., $\mathsf{fn}(S) \cap \mathsf{bn}(S) = \emptyset$ and no name is bound twice in S. Then S is *well-structured* if there exists a type system of π_{s} with the types $\mathbb{T}_{\mathrm{s}}$ and a set of type assignments $\mathcal{T}_S$ such that

1. for all names $x \in \mathsf{n}(S)$ there is a type assignment $x{:}T_S$ in $\mathcal{T}_S$ with $T_S \in \mathbb{T}_{\mathrm{s}}$,
2. for all $y, z \in \mathsf{n}(S)$ and all $S' \in \pi_{\mathrm{s}}$ such that $\overline{y}z.S'$ or $y(z)\,.S'$ is a subterm of S, $y{:}T_S \in \mathcal{T}_S$ and $z{:}T'_S \in \mathcal{T}_S$ imply $T_S = \sharp(T'_S)$, and
3. there exists a type environment $\Gamma_S \subseteq \mathcal{T}_S$ such that $\Gamma_S \vdash \mathcal{T}_S(S)$.

In this case, we also say that S is well-structured with respect to $\mathbb{T}_{\mathrm{s}}$ and $\mathcal{T}_S$.

We can type all target terms in $\mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ with respect to all well-structured source terms of π_{s}. Remember that the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ in Figure 5.1 translates source term outputs $\overline{y}\langle z\rangle$ into polyadic outputs $\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\rangle$, i.e., it adds a sum lock and a sender lock to outputs (and also inputs) on translated source term names. To capture this, we inductively replace source term link types by link types carrying additionally a sum lock type and a sender lock type.

Definition 6.2.6 (Translation of Source Types)**.** Let $\mathbb{T}_{\mathrm{s}}$ be the set of types of a type system in π_{s} and $T_S \in \mathbb{T}_{\mathrm{s}}$. Then the translation of the source type T_S into a type of the basic type system, denoted by $\widetilde{T_S}$, is defined as:

$$\widetilde{T_S} = \begin{cases} \sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T'_S}\right), & \text{if } \exists T'_S \in \mathbb{T}_{\mathrm{s}} \,.\, T_S = \sharp(T'_S) \\ T_S, & \text{else} \end{cases}$$

Moreover, we extend the translation of source types to the translation of type assignments, i.e., $\widetilde{x{:}T_S} \triangleq \varphi^{\mathrm{s}}_{\mathrm{a}}(x){:}\widetilde{T_S}$, and to the translation of sets of type assignments, i.e., $\widetilde{\mathcal{T}_S} \triangleq \left\{ \widetilde{x{:}T_S} \mid x{:}T_S \in \mathcal{T}_S \right\}$.

The remaining types, i.e., the types of the names introduced by the encoding function, are obtained for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ as for the other encodings. An overview of all names and their

Description	Names	Type	Condition
source term names	$\varphi^{\mathrm{s}}_{\mathrm{a}}(x)$	V_S	if $V_S \in \mathbb{T}_{\mathrm{s}}$, $V_S \neq \sharp(T_S)$, and $\Gamma \vdash x : V_S$
		$\sharp(\mathfrak{l}, \mathfrak{s}, T_S)$	if $T_S \in \mathbb{T}_{\mathrm{s}}$ and $\Gamma \vdash x : \sharp(T_S)$
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$	$\sharp(\mathfrak{l}, \mathfrak{s}, T_S)$	where $T_S \in \mathbb{T}_{\mathrm{s}}$ and $\Gamma \vdash x : \sharp(T_S)$
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(z)$	V_S	if $V_S \in \mathbb{T}_{\mathrm{s}}$, $V_S \neq \sharp(T_S)$, and $\Gamma \vdash z : V_S$
		$\sharp(\mathfrak{l}, \mathfrak{s}, T_S)$	if $T_S \in \mathbb{T}_{\mathrm{s}}$ and $\Gamma \vdash z : \sharp(T_S)$
auxiliary values	v_t	$\mathfrak{v}_\top$	
	v_f	$\mathfrak{v}_\bot$	
	v_s	$\mathfrak{v}_\mathfrak{s}$	
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$	
booleans	t	$\sharp(\mathfrak{v}_\top)$	
	f	$\sharp(\mathfrak{v}_\bot)$	
sum locks	l, l'	$\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_\top), \sharp(\mathfrak{v}_\bot))$	
sender locks	s	$\mathfrak{s} = \sharp(\mathfrak{v}_\mathfrak{s})$	
receiver locks	r	$\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$	

Figure 6.3.: Basic Types in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$.

types for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ is given in Figure 6.3. Again, $\mathcal{T}^1_{\mathrm{B}}$ is the set of type assignments specified in this figure, where in this case $\mathcal{T}^1_{\mathrm{B}}$ is defined with respect to a set $\mathbb{T}_{\mathrm{s}}$ of source term types.

The set of types of the basic type system is the union of the types given in Figures 6.1, 6.2, and 6.3.

Definition 6.2.7 (Basic Types)**.** Let $\mathbb{T}_{\mathrm{s}}$ be the types of the type system of the source language π_{s}. The types of the basic type system, denoted as *basic types*, are given by the sets

$$
\begin{aligned}
\mathbb{V}_{\mathrm{B}} &\triangleq \{\, \mathfrak{v}_{\mathfrak{n}}, \mathfrak{v}_\top, \mathfrak{v}_\bot, \mathfrak{v}_\mathfrak{s}, \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \,\} \cup \{\, V_S \mid V_S \in \mathbb{T}_{\mathrm{s}} \wedge \forall T_S \in \mathbb{T}_{\mathrm{s}} \,.\, V_S \neq \sharp(T_S) \,\} \\
\mathbb{L}_{\mathrm{B}} &\triangleq \{\, \sharp(\mathfrak{v}_\top), \sharp(\mathfrak{v}_\bot), \mathfrak{l}, \mathfrak{s}, \mathfrak{r}, \mathfrak{o}, \mathfrak{i}, \sharp(\mathfrak{i}), \sharp(\mathfrak{o}), \sharp(\mathfrak{v}_{\mathfrak{n}}), \sharp(\mathfrak{o}, \mathfrak{i}), \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), \mathfrak{r}', \mathfrak{o}', \mathfrak{i}', \mathfrak{t}_{\mathfrak{o}}, \mathfrak{t}_{\mathfrak{i}} \,\} \\
&\qquad \cup \left\{\, \widetilde{T_S} \mid \sharp(T_S) \in \mathbb{T}_{\mathrm{s}} \,\right\} \\
\mathbb{T}_{\mathrm{B}} &\triangleq \mathbb{V}_{\mathrm{B}} \cup \mathbb{L}_{\mathrm{B}}
\end{aligned}
$$

of basic value types $\mathbb{V}_{\mathrm{B}}$, basic link types $\mathbb{L}_{\mathrm{B}}$, and basic types $\mathbb{T}_{\mathrm{B}}$, respectively.

The typing rules of the basic type system are given by the rules in Figure 6.4. Note that the Rules T-Res$_{\mathrm{B}}$, T-In$_{\mathrm{B}}$, and T-Rep$_{\mathrm{B}}$ assume the implicit side condition that the required enlargement of the type environment is possible, i.e., $\Gamma(x)$ is not defined or equal to T for the Rule T-Res$_{\mathrm{B}}$ and, for all $1 \leq i \leq n$, $\Gamma(x_i)$ is not defined or equal to T_i for T-In$_{\mathrm{B}}$ and T-Rep$_{\mathrm{B}}$.

The typing rules of the basic type system are very similar to the rules presented at Page 241 in [SW01]. As explained above, the axioms T-Name$_{\mathrm{B}}$, T-Nil$_{\mathrm{B}}$, and T-Succ$_{\mathrm{B}}$

$$\text{T-Name}_\text{B}\ \frac{}{\Gamma, x{:}T \vdash x{:}T} \qquad \text{T-Nil}_\text{B}\ \frac{}{\Gamma \vdash \mathsf{0}} \qquad \text{T-Succ}_\text{B}\ \frac{}{\Gamma \vdash \checkmark}$$

$$\text{T-Res}_\text{B}\ \frac{\Gamma, x{:}T \vdash P}{\Gamma \vdash (\nu x{:}T)\, P} \qquad \text{T-Par}_\text{B}\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

$$\text{T-Mat}_\text{B}\ \frac{\Gamma \vdash a{:}T \quad \Gamma \vdash b{:}T \quad \Gamma \vdash P}{\Gamma \vdash [a = b]\, P} \qquad \text{T-Tau}_\text{B}\ \frac{\Gamma \vdash P}{\Gamma \vdash \tau.P}$$

$$\text{T-Out}_\text{B}\ \frac{\Gamma \vdash y{:}\sharp(T_1, \ldots, T_n) \quad \Gamma \vdash z_1{:}T_1 \quad \ldots \quad \Gamma \vdash z_n{:}T_n}{\Gamma \vdash \overline{y}\langle z_1, \ldots, z_n\rangle}$$

$$\text{T-OutPS}_\text{B}\ \frac{\Gamma \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma \vdash o{:}\sharp(T_1, \ldots, T_n) \quad \Gamma \vdash z_1{:}T_1 \quad \ldots \quad \Gamma \vdash z_n{:}T_n}{\Gamma \vdash \overline{y \cdot o}\langle z_1, \ldots, z_n\rangle}$$

$$\text{T-In}_\text{B}\ \frac{\Gamma \vdash y{:}\sharp(T_1, \ldots, T_n) \quad \Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash P}{\Gamma \vdash y(x_1, \ldots, x_n).P}$$

$$\text{T-InPS}_\text{B}\ \frac{\Gamma \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma \vdash o{:}\sharp(T_1, \ldots, T_n) \quad \Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash P}{\Gamma \vdash y \cdot o(x_1, \ldots, x_n).P}$$

$$\text{T-Rep}_\text{B}\ \frac{\Gamma \vdash y{:}\sharp(T_1, \ldots, T_n) \quad \Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash P}{\Gamma \vdash y^\star(x_1, \ldots, x_n).P}$$

Figure 6.4.: Typing Rules of the Basic Type System.

state that a type assignment of a type environment respects this type environment and that $\mathsf{0}$ and $\checkmark$ respect every type environment—a simple consequence of the fact that neither $\mathsf{0}$ nor $\checkmark$ contain names. T-Res$_\text{B}$ states that a typed term P under a typed restriction $(\nu x{:}T)$ respects a type environment Γ if P respects $\Gamma, x{:}T$. A parallel composition respects a type environment if each of its subterms respects this environment. Similarly, by the Rule T-Tau$_\text{B}$, $\tau.P$ respect Γ if P respects Γ. The Rule T-Mat$_\text{B}$ states that a typed process P guarded by a matching prefix $[a = b]$ respects a type environment Γ if a and b have the same type that can be derived from Γ and P respects Γ. The rules for output and input are variants of the corresponding rules in [SW01] extended to polyadic communication. An output guarded process $\overline{y}\langle z_1, \ldots, z_n\rangle$ respects a type environment Γ if for the channel y a link type with the n arguments $T_1, \ldots, T_n$ and for each of the transmitted values z_i the corresponding type T_i can be derived from Γ. Note that, because we type an asynchronous calculus, outputs have no continuations. Accordingly, an input guarded process $y(x_1, \ldots, x_n).P$ respects Γ if again for y a link type with the n arguments $T_1, \ldots, T_n$ can be derived from Γ and P respects $\Gamma, x_1{:}T_1, \ldots, x_n{:}T_n$. The Rules T-OutPS$_\text{B}$ and T-InPS$_\text{B}$ are necessary to type links for polyadic synchronisation in $[\![\cdot]\!]^\text{m}_\text{p}$. They require that polyadic links $y \cdot o$ are constructed from a source term name

y, i.e., the type of y has to be $\mathfrak{v}_\mathfrak{n}$, and a tag o that represents the actual type of the link represented by $y \cdot o$. Apart from that T-OUTPS$_\text{B}$ and T-INPS$_\text{B}$ are similar to T-OUT$_\text{B}$ and T-IN$_\text{B}$. The Rule T-REP$_\text{B}$ for replicated inputs is similar to T-IN$_\text{B}$.

Definition 6.2.8 (Basic Type System)**.** The *basic type system* is given by the basic types in Definition 6.2.7 and the typing rules in Figure 6.4.

In the basic type system a typed process is well-typed with respect to a type environment, if we can derive a type judgement by the typing rules in Figure 6.4. Some of these rules, as for instance T-RES$_\text{B}$, require that a subterm is well-typed under an extended type environment. This is necessary to check type constrains on bound names, i.e., the added type assignments provide the types of the bound names. By the implicit assumption on type environments in Definition 6.2.2, extensions are allowed only if the new type assignment is not in conflict with the type environment, i.e., for some new assignment $x{:}T$ either the type environment does not contain a type assignment for x or it already assigns the type T to x. Hence, to type processes, it is sometimes necessary to apply α-conversion, in order to avoid name clashes between a typed process and its type environment. In the following, whenever we prove a type judgement, i.e., not only for the basic type system, we will silently apply α-conversion if necessary. Thus, we equate typed processes that are equivalent modulo α-conversion.

To ensure that the set of typing rules of a type system indeed guarantees agreement with the behaviour of the term, we augment each type system with the already mentioned subject reduction lemma. In order to simplify its proof, we show some properties of the basic type system. Note that we implicitly assume the basic type system for the rest of this subsection. The first lemma states two properties. A type assignment can be derived if and only if it is already part of the type environment. And if it is possible to derive a type assignment then this assignment is unambiguous, i.e., given a type environment each name can be well-typed for at most one type.

Lemma 6.2.9. $\Gamma \vdash x{:}T$ *iff* $\Gamma(x) = T$*. And for every type environment* Γ *and name* x *there is at most one type* T *such that* $\Gamma \vdash x{:}T$*.*

Proof. By the typing rules in Figure 6.4, $\Gamma \vdash x : T$ can only be derived from the Rule T-NAME$_\text{B}$ under the condition that $\Gamma(x) = T$, i.e., $\Gamma = \Gamma', x{:}T$, which proves the first property. Moreover, $\Gamma \vdash x{:}T'$ for some $T' \neq T$ can only be derived if $\Gamma'(x) = T'$. But the implicit assumption on type environments in Definition 6.2.2 states that Γ cannot contain two assignments with different types for the same name. We conclude that for all Γ, x, T, and T' such that $\Gamma \vdash x{:}T$ as well as $\Gamma \vdash x{:}T'$ we have $T = T'$. □

Strengthening allows to remove superfluous type assignments from type judgements.

Lemma 6.2.10 (Strengthening)**.** *If* $\Gamma, x'{:}T' \vdash P$ *and* $x' \notin \mathsf{fn}(P)$ *then* $\Gamma \vdash P$*, and if* $\Gamma, x'{:}T' \vdash x{:}T$ *and* $x' \neq x$ *then* $\Gamma \vdash x{:}T$*.*

Proof. By Lemma 6.2.9, $\Gamma, x'{:}T' \vdash x{:}T$ and $x' \neq x$ imply $\Gamma(x) = T$, i.e., $\Gamma = \Gamma', x{:}T$. Then $\Gamma', x{:}T \vdash x{:}T$ follows directly by T-NAME$_\text{B}$.

For the other case, we perform an induction on the depth of the derivation.

Base Case: If $\Gamma, x':T' \vdash P$ can be derived from one of the axioms then either $P = \mathsf{0}$ or $P = \checkmark$. In both cases $\Gamma \vdash P$ follows again directly by T-NIL$_\mathrm{B}$ or T-SUCC$_\mathrm{B}$.

Induction Hypothesis: $\forall P \in (\mathcal{P}_\mathrm{a}^{=,\sim} : \mathbb{T}_\mathrm{B}) \;.\; \Gamma, x':T' \vdash P \wedge x' \notin \mathsf{fn}(P)$ imply $\Gamma \vdash P$

Induction Step: We have to distinguish nine cases—one for each inference rule in Figure 6.4. All cases follow simply by applying the respective rule on $\Gamma \vdash P$ and then concluding by the induction hypothesis and by the strengthening property for type assignments shown above. We show the case for Rule T-IN$_\mathrm{B}$ as example.

In this case, P is an input guarded process $y(x_1, \ldots, x_n).P'$ and the first step of the derivation is:

$$\frac{\dfrac{\cdots}{\Gamma, x':T' \vdash y:\sharp(T_1, \ldots, T_n)} \cdots \quad \dfrac{\cdots}{\Gamma, x':T', x_1:T_1, \ldots, x_n:T_n \vdash P'} \cdots}{\Gamma, x':T' \vdash y(x_1, \ldots, x_n).P'} \text{ T-IN}_\mathrm{B}$$

If we apply T-IN$_\mathrm{B}$ to $\Gamma \vdash P$ we have to show the subgoals $\Gamma \vdash y:\sharp(T_1, \ldots, T_n)$ and $\Gamma, x_1:T_1, \ldots, x_n:T_n \vdash P'$. The first follows from $\Gamma, x':T' \vdash y:\sharp(T_1, \ldots, T_n)$ and the weakening property for type assignments. The second subgoal follows from $\Gamma, x':T', x_1:T_1, \ldots, x_n:T_n \vdash P'$ by the induction hypothesis.

□

The weakening property shows that a type judgement remains valid if we add additional type assignment as long as these are not in conflict with the given type environment.

Lemma 6.2.11 (Weakening)**.** *If $\Gamma \vdash P$ then $\Gamma, x':T' \vdash P$, and if $\Gamma \vdash x:T$ then $\Gamma, x':T' \vdash x:T$ for any type T' and any name x' such that $\Gamma(x')$ is not defined or equal to T'.*

Proof. Since type environments are sets, if $\Gamma(x') = T'$ then $\Gamma, x':T' = \Gamma$ and the lemma holds trivially. We consider the other case.

$\Gamma \vdash x:T$ can only derived by the Rule T-NAME$_\mathrm{B}$. We conclude that $\Gamma(x) = T$, i.e., $\Gamma = \Gamma', x:T$. Then $\Gamma', x:T, x':T' \vdash x:T$ follows directly by T-NAME$_\mathrm{B}$ again.

For the other case, i.e., $\Gamma \vdash P$, we perform an induction on the depth of the derivation.

Base Case: If $\Gamma \vdash P$ can be derived from one of the axioms then either $P = \mathsf{0}$ or $P = \checkmark$. In both cases $\Gamma, x':T' \vdash P$ follows again directly by T-NIL$_\mathrm{B}$ or T-SUCC$_\mathrm{B}$.

Induction Hypothesis:

$$\forall x' \in \mathcal{N} \;.\; T' \in \mathbb{T}_\mathrm{B} \;.\; \forall P \in (\mathcal{P}_\mathrm{a}^{=,\sim} : \mathbb{T}_\mathrm{B}) \;.\; \Gamma \vdash P \wedge x' \notin \mathsf{n}(\Gamma) \text{ imply } \Gamma, x':T' \vdash P$$

Induction Step: Again, we consider one case for each inference rule in Figure 6.4. All cases simply follow by applying the respective rule on $\Gamma, x' : T' \vdash P$ and then concluding by the induction hypothesis. We show the case for Rule T-IN$_\mathrm{B}$ as example.

In this case, P is an input guarded process $y(x_1, \ldots, x_n).P'$ and the first step of the derivation is:

$$\frac{\overset{\cdots}{\Gamma \vdash y : \sharp(T_1, \ldots, T_n)} \cdots \quad \overset{\cdots}{\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P'} \cdots}{\Gamma \vdash y(x_1, \ldots, x_n).P'} \text{ T-In}_\text{B}$$

If $x' \in \{ x_1, \ldots, x_n \}$, we apply α-conversion to avoid a name clash between P and $x' : T'$. If we apply T-In$_\text{B}$ to $\Gamma, x' : T' \vdash P$ we have to show the subgoals $\Gamma, x' : T' \vdash y : \sharp(T_1, \ldots, T_n)$ and $\Gamma, x' : T', x_1 : T_1, \ldots, x_n : T_n \vdash P'$. The first follows from $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$ and the weakening property for type assignments. The second subgoal follows from $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P'$ by the induction hypothesis.

□

Before we show that the typing rules agree with the reduction relation, we prove that they agree with structural congruence. Unfortunately, this is not true for the match prefix.

Example 6.2.12. Let $P \in (\mathcal{P}_\text{a}^{=,\sim} : \mathbb{T}_\text{B})$ and $Q \triangleq [a = a]\, P$. Obviously, $P \equiv Q$. Moreover, $\Gamma \vdash Q$ always implies $\Gamma \vdash P$, because of Rule T-Mat$_\text{B}$. But $\Gamma \vdash P$ does not necessarily imply $\Gamma \vdash Q$, as for instance if $\Gamma = \emptyset$ and $P = \mathsf{0}$, because there is no type assignment for a.

Because of the rule $[a = b]\, P \equiv P$, it is possible to introduce free names to a typed term for which we do not have any type constrains. To handle this problem, we introduced closed type environments in Section 6.2.1. Hence, we prove agreement with the rules of structural congruence only with respect to closed type environments.

Lemma 6.2.13. *If $\Gamma \vdash P$, $P \equiv Q$, and Γ is closed for Q then $\Gamma \vdash Q$.*

Proof. We show the condition for a single application of the rules of structural congruence in Figure 2.1 at Page 18. The lemma then follows by induction on the depth of the derivation of $P \equiv Q$. As described above, we equate typed processes that are equivalent modulo α-conversion. Of the remaining rules of structural congruence the only interesting case is $[a = b]\, P \equiv P$. All other cases are simple consequences of the Rules T-Nil$_\text{B}$, T-Par$_\text{B}$, and T-Res$_\text{B}$. Note that for the last case $P \mid (\nu x{:}T)\, Q \equiv (\nu x{:}T)\, (P \mid Q)$ the side condition $x \notin \mathsf{fn}(P)$ ensures that the derivation of $\Gamma \vdash P \mid (\nu x{:}T)\, Q$ can be constructed from the derivation of $\Gamma \vdash (\nu x{:}T)\, (P \mid Q)$, because it ensures that the additional type assignment on x is superfluous to prove that P is well-typed and can be removed by Lemma 6.2.10.

In the case of the rule for the match prefix, either $P = [a = a]\, Q$ or $Q = [a = a]\, P$ for some name a. In the first case, $\Gamma \vdash Q$ follows from $\Gamma \vdash [a = a]\, Q$ and T-Mat$_\text{B}$. In the second case, i.e., if $Q = [a = a]\, P$, we apply T-Mat$_\text{B}$ which results in three subgoals. The first two are both $\Gamma \vdash a : T$ for some arbitrary type T. Since Γ is closed for Q, there is some T' such that $\Gamma\,(a) = T'$. Thus, we choose $T = T'$. Then $\Gamma \vdash a : T$ follows by Lemma 6.2.9. The last subgoal $\Gamma \vdash P$ holds by assumption. □

Of course, whenever some typed process respects some type environment then this type environment is closed for the typed process, i.e., provides a type for all free names.

Lemma 6.2.14. *If $\Gamma \vdash P$ then Γ is closed for P.*

Proof. Assume the contrary, i.e., assume $\Gamma \vdash P$ and $x \notin \mathsf{fn}(\Gamma)$ but $x \in \mathsf{fn}(P)$. Apart from the matching operator free names can occur as links or values of outputs. Hence, P has a (potentially guarded) subterm which is either of the form

1. $\overline{y}\langle z_1, \ldots, z_n\rangle$ or $\overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$ for some names $y, o, z_1, \ldots, z_n \in \mathcal{N}$,
2. $y(x_1, \ldots, x_n).P'$, or $y \cdot o(x_1, \ldots, x_n).P'$, or $y^\star(x_1, \ldots, x_n).P'$ for some $y, o \in \mathcal{N}$, some bound names $x_1, \ldots, x_n \in \mathcal{N}$, and some P', or
3. $[a = b]\, P'$ for some names $a, b \in \mathcal{N}$ and some P'

such that $x \in \{ y, o, z_1, \ldots, z_n, a, b \}$. None of the typing rules in Figure 6.4 allows to derive a type judgement for a term without requiring a type judgement for all its subterms. Hence, in the derivation of $\Gamma \vdash P$ there is some subgoal $\Gamma' \vdash \overline{y}\langle z_1, \ldots, z_n\rangle$, $\Gamma' \vdash \overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$, $\Gamma' \vdash y(x_1, \ldots, x_n).P'$, $\Gamma' \vdash y \cdot o(x_1, \ldots, x_n).P'$, $\Gamma' \vdash y^\star(x_1, \ldots, x_n).P'$, or $\Gamma' \vdash [a = b]\, P'$ for some Γ' that is obtained from Γ during the derivation up to the mentioned subgoal. The typing rules in Figure 6.4 can add type assignments to Γ during this partial derivation but do so only for bound names in the Rules T-Res$_\mathrm{B}$, T-In$_\mathrm{B}$, T-InPS$_\mathrm{B}$, and T-Rep$_\mathrm{B}$ (the other rules do not add type assignments). We conclude that $x \notin \mathsf{fn}(\Gamma')$.

Consider the case that $x = y$. Then applying the Rule T-Out$_\mathrm{B}$ on the subgoal $\Gamma' \vdash \overline{y}\langle z_1, \ldots, z_n\rangle$, T-In$_\mathrm{B}$ on the subgoal $\Gamma' \vdash y(x_1, \ldots, x_n).P$, or T-Rep$_\mathrm{B}$ on the subgoal $\Gamma' \vdash y^\star(x_1, \ldots, x_n).P$, which are the only applicable rules, results in the subgoal $\Gamma' \vdash y : \sharp(T_1, \ldots, T_n)$ for some types $T_1, \ldots, T_n$. Moreover, applying the Rule T-OutPS$_\mathrm{B}$ on the subgoal $\Gamma' \vdash \overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$ or T-InPS$_\mathrm{B}$ on the subgoal $\Gamma' \vdash y \cdot o(x_1, \ldots, x_n).P$ results in the subgoal $\Gamma' \vdash y : \mathfrak{v}_\mathsf{n}$. But, since $y \notin \mathsf{fn}(\Gamma')$ and because of Lemma 6.2.9, neither $\Gamma' \vdash y : \sharp(T_1, \ldots, T_n)$ nor $\Gamma' \vdash y : \mathfrak{v}_\mathsf{n}$ can hold. Hence, the judgement $\Gamma \vdash P$ cannot be derived, which contradicts the assumption.

If $x = o$, applying the Rule T-OutPS$_\mathrm{B}$ on the subgoal $\Gamma' \vdash \overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$ or T-InPS$_\mathrm{B}$ on $\Gamma' \vdash y \cdot o(x_1, \ldots, x_n).P$ results in the subgoal $\Gamma' \vdash o : \sharp(T_1, \ldots, T_n)$ for some types $T_1, \ldots, T_n$. Again, by $o \notin \mathsf{fn}(\Gamma')$ and Lemma 6.2.9, $\Gamma' \vdash o : \sharp(T_1, \ldots, T_n)$ has to fail. Hence, the judgement $\Gamma \vdash P$ cannot be derived, which contradicts the assumption.

If $x \in \{ z_1, \ldots, z_n \}$ then P contains an output $\overline{y}\langle z_1, \ldots, z_n\rangle$ or $\overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$. Applying T-Out$_\mathrm{B}$ or T-OutPS$_\mathrm{B}$ on the corresponding subgoal results in the subgoals $\Gamma' \vdash z_1 : T_1$, …, $\Gamma' \vdash z_n : T_n$. But, since $x \notin \mathsf{fn}(\Gamma')$, $x \in \{ z_1, \ldots, z_n \}$, and because of Lemma 6.2.9, one of these subgoals has to fail. Hence, the judgement $\Gamma \vdash P$ cannot be derived, which contradicts again the assumption.

In the remaining case $x \in \{ a, b \}$ and P contains a subterm $[a = b]\, P'$. Applying T-Mat$_\mathrm{B}$ on the subgoal $\Gamma' \vdash [a = b]\, P'$, which is the only applicable rule, results in the subgoals $\Gamma' \vdash a : T$ and $\Gamma' \vdash b : T$ for some type T. But, since $x \notin \mathsf{fn}(\Gamma')$, $x \in \{ a, b \}$, and because of Lemma 6.2.9, one of these subgoals has to fail. Hence, the judgement $\Gamma \vdash P$ cannot be derived, which contradicts the assumption. □

The next lemma shows that type judgements are robust under substitution if the substitution preserves the type of the substituted name, i.e., if the substituted name and its replacement have the same type.

Lemma 6.2.15. *Assume* $\Gamma(x) = \Gamma(z)$. *Then* $\Gamma \vdash P$ *implies* $\Gamma \vdash \{ {}^{z}/_{x} \} P$.

Proof. We construct the derivation of $\Gamma \vdash \{ {}^{z}/_{x} \} P$ from the derivation of $\Gamma \vdash P$, by showing that both proof trees have the same structure, i.e., apply the same typing rules in the same order. Analysing the rules in Figure 6.4, we observe that the rules T-Nil$_{\mathrm{B}}$, T-Succ$_{\mathrm{B}}$, T-Par$_{\mathrm{B}}$, and T-Tau$_{\mathrm{B}}$ do not consider specific names, i.e., can be applied in exactly the same way in both proof trees. The same holds for the third subgoal of T-Mat$_{\mathrm{B}}$. Moreover, we observe that the typing rules may add type assignments to the environments of their subgoals but do not remove any, i.e., for all subgoals with a type environment Γ' in both derivations we have $\Gamma'(x) = \Gamma'(z)$.

P and $\{ {}^{z}/_{x} \} P$ differ only if x is free in P. So, T-Res$_{\mathrm{B}}$ can also be applied in exactly the same way in both proof trees. For the same reason, the respective second subgoals of the Rules T-In$_{\mathrm{B}}$ and T-Rep$_{\mathrm{B}}$ as well as the last subgoal of T-InPS$_{\mathrm{B}}$ are unaffected.

All remaining subgoals are of the form $\Gamma'' \vdash y{:}T'$. By Lemma 6.2.9 and $\Gamma(x) = \Gamma(z)$, $\Gamma \vdash x : T$ iff $\Gamma \vdash z : T$. By Lemma 6.2.11, this condition remains valid if during the derivation new assignments are added to Γ, i.e., $\Gamma \vdash x : T$ iff $\Gamma \vdash z : T$ implies that $\Gamma, \Gamma' \vdash x : T$ iff $\Gamma, \Gamma' \vdash z : T$ for all Γ'. Hence, for all leaves proven by T-Name$_{\mathrm{B}}$ in the proof tree of $\Gamma \vdash P$ we can prove the corresponding leaf in the proof tree of $\Gamma \vdash \{ {}^{z}/_{x} \} P$ again by T-Name$_{\mathrm{B}}$. □

Subject reduction, finally, ensures the agreement of a type system with the reduction relation of a process calculus [SW01]. Accordingly, we show that the typing rules in Figure 6.4 agree with the reduction relation of $\pi_{\mathrm{a}}^{=,\sim}$.

Lemma 6.2.16 (Subject Reduction)**.** *If* $\Gamma \vdash P$, $P \longmapsto P'$, *and* Γ *is closed for* P' *then* $\Gamma \vdash P'$.

Proof. We perform an induction on the depth of the derivation of $P \longmapsto P'$.

Base Case: The reduction semantics of the target languages $\pi_{\mathrm{a}}^{\sim}$, $\pi_{\mathrm{p}}^{\sim}$, and $\pi_{\mathrm{a}}^{=,\sim}$ in Figure 2.4 contains the Axioms Pi-Tau$_{\mathrm{a,p}}^{\sim}$, Pi-Com$_{\mathrm{a,p}}^{\sim}$, Pi-ComPS$_{\mathrm{p}}^{\sim}$, and Pi-Rep$_{\mathrm{a,p}}^{\sim}$. The first rule requires that $P = \tau.Q$ and $P' = Q$ for some Q that, depending on the considered encoding, is in $\mathcal{P}_{\mathrm{a}}^{\sim} : \mathbb{T}_{\mathrm{B}}$, $\mathcal{P}_{\mathrm{p}}^{\sim} : \mathbb{T}_{\mathrm{B}}$, or $\mathcal{P}_{\mathrm{a}}^{=,\sim} : \mathbb{T}_{\mathrm{B}}$. Hence, $\Gamma \vdash P'$ follows from $\Gamma \vdash P$ and T-Tau$_{\mathrm{B}}$.

The Rule Pi-Com$_{\mathrm{a,p}}^{\sim}$ requires that $P = y(x_1, \ldots, x_n).Q \mid \overline{y}\langle z_1, \ldots, z_n \rangle$ and $P' = \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q$. In this case, the derivation of $\Gamma \vdash P$ starts with

$$\frac{D_1 \quad D_2}{\Gamma \vdash y(x_1, \ldots, x_n).Q \mid \overline{y}\langle z_1, \ldots, z_n \rangle}\text{T-Par}_{\mathrm{B}}$$

where

$$D_1 = \frac{\dfrac{\cdots}{\Gamma \vdash y{:}\sharp(T_1, \ldots, T_n)} \cdots \quad \dfrac{\cdots}{\Gamma, x_1{:}T_1, \ldots, x_n{:}T_n \vdash Q} \cdots}{\Gamma \vdash y(x_1, \ldots, x_n).Q}\text{T-In}_{\mathrm{B}}$$

and

$$D_2 = \dfrac{\dfrac{\cdots}{\Gamma \vdash y : \sharp(T'_1, \ldots, T'_n)} \cdots \quad \dfrac{\cdots}{\Gamma \vdash z_1 : T'_1} \cdots \quad \cdots \quad \dfrac{\cdots}{\Gamma \vdash z_n : T'_n} \cdots}{\Gamma \vdash \overline{y}\langle z_1, \ldots, z_n \rangle} \text{T-Out}_\text{B}$$

By $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$, $\Gamma \vdash y : \sharp(T'_1, \ldots, T'_n)$, and Lemma 6.2.9, we have $T_i = T'_i$ for all $1 \leq i \leq n$. Moreover, because of $\Gamma \vdash z_1 : T'_1$, $\ldots \Gamma \vdash z_n : T'_n$, and again Lemma 6.2.9, we know that $\Gamma(z_i) = T_i$ for all $1 \leq i \leq n$. With Lemma 6.2.15 and $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash Q$ we conclude $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P'$. Finally, by Lemma 6.2.10, we have $\Gamma \vdash P'$, because $x_i \notin \mathsf{fn}(P')$ for all $1 \leq i \leq n$.

The Rule Pi-ComPS$_\text{p}$ requires that $P = y_1 \cdot y_2(x_1, \ldots, x_n).Q \mid \overline{y_1 \cdot y_2}\langle z_1, \ldots, z_n \rangle$ and $P' = \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q$. In this case, the derivation of $\Gamma \vdash P$ starts again with

$$\dfrac{D_1 \quad D_2}{\Gamma \vdash y(x_1, \ldots, x_n).Q \mid \overline{y}\langle z_1, \ldots, z_n \rangle} \text{T-Par}_\text{B}$$

where

$$D_1 = \dfrac{\dfrac{\cdots}{\Gamma \vdash y_1 : \mathfrak{b}_\mathfrak{n}} \cdots \quad \dfrac{\cdots}{\Gamma \vdash y_2 : \sharp(T_1, \ldots, T_n)} \cdots \quad D'_1}{\Gamma \vdash y_1 \cdot y_2(x_1, \ldots, x_n).Q} \text{T-InPS}_\text{B}$$

$$D_2 = \dfrac{\dfrac{\cdots}{\Gamma \vdash y_1 : \mathfrak{b}_\mathfrak{n}} \cdots \quad \dfrac{\cdots}{\Gamma \vdash y_2 : \sharp(T'_1, \ldots, T'_n)} \cdots \quad D'_2}{\Gamma \vdash \overline{y_1 \cdot y_2}\langle z_1, \ldots, z_n \rangle} \text{T-OutPS}_\text{B}$$

and again

$$D'_1 = \dfrac{\cdots}{\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash Q} \cdots$$

$$D'_2 = \dfrac{\cdots}{\Gamma \vdash z_1 : T'_1} \cdots \quad \cdots \quad \dfrac{\cdots}{\Gamma \vdash z_n : T'_n} \cdots$$

By $\Gamma \vdash y_2 : \sharp(T_1, \ldots, T_n)$, $\Gamma \vdash y_2 : \sharp(T'_1, \ldots, T'_n)$, and Lemma 6.2.9, we have $T_i = T'_i$ for all $1 \leq i \leq n$. Moreover, because of $\Gamma \vdash z_1 : T'_1$, $\ldots \Gamma \vdash z_n : T'_n$, and again Lemma 6.2.9, we know that $\Gamma(z_i) = T_i$ for all $1 \leq i \leq n$. With Lemma 6.2.15 and $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash Q$ we conclude $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P'$. Finally, by Lemma 6.2.10, we have $\Gamma \vdash P'$, because $x_i \notin \mathsf{fn}(P')$ for all $1 \leq i \leq n$.

The Rule Pi-Rep$^{\sim}_{\text{a,p}}$ requires that $P = y^\star(x_1, \ldots, x_n).Q \mid \overline{y}\langle z_1, \ldots, z_n \rangle$ and $P' = \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q \mid y^\star(x_1, \ldots, x_n).Q$. The derivation of $\Gamma \vdash P$ starts with

$$\dfrac{D_1 \quad D_2}{\Gamma \vdash y^\star(x_1, \ldots, x_n).Q \mid \overline{y}\langle z_1, \ldots, z_n \rangle} \text{T-Par}_\text{B}$$

where

$$D_1 = \dfrac{\dfrac{\cdots}{\Gamma \vdash y : \sharp(T_1, \ldots, T_n)} \cdots \quad \dfrac{\cdots}{\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash Q} \cdots}{\Gamma \vdash y^\star(x_1, \ldots, x_n).Q} \text{T-Rep}_\text{B}$$

and again

$$D_2 = \dfrac{\dfrac{\cdots}{\Gamma \vdash y : \sharp(T'_1, \ldots, T'_n)} \cdots \quad \dfrac{\cdots}{\Gamma \vdash z_1 : T'_1} \cdots \quad \cdots \quad \dfrac{\cdots}{\Gamma \vdash z_n : T'_n} \cdots}{\Gamma \vdash \overline{y}\langle z_1, \ldots, z_n \rangle} \text{T-Out}_\text{B}$$

By $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$, $\Gamma \vdash y : \sharp(T'_1, \ldots, T'_n)$, and Lemma 6.2.9, we have $T_i = T'_i$ for all $1 \leq i \leq n$. Moreover, because of $\Gamma \vdash z_1 : T'_1, \ldots \Gamma \vdash z_n : T'_n$, and again Lemma 6.2.9, we know that $\Gamma(z_i) = T_i$ for all $1 \leq i \leq n$. With Lemma 6.2.15 and $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash Q$ we conclude $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q$. And, by Lemma 6.2.10, we have $\Gamma \vdash \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q$, because $x_i \notin \mathsf{fn}(\{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q)$ for all $1 \leq i \leq n$. Finally, $\Gamma \vdash P'$ follows by T-Par$_\text{B}$ and because of $\Gamma \vdash \{ {}^{z_1}/_{x_1}, \ldots, {}^{z_n}/_{x_n} \} Q$ for the left and $\Gamma \vdash y^\star(x_1, \ldots, x_n).Q$ for the right hand side.

Induction Hypothesis: $\Gamma \vdash P, P \longmapsto P'$, and Γ is closed for P' implies $\Gamma \vdash P'$ for all $P \in (\mathcal{P}_\text{a}^{\sim} : \mathbb{T}_\text{B})$, $P \in \left(\mathcal{P}_\text{p}^{\sim} : \mathbb{T}_\text{B}\right)$, and $P \in (\mathcal{P}_\text{a}^{=,\sim} : \mathbb{T}_\text{B})$

Induction Step: There are three cases; one for each of the reduction Rules Pi-Res$_{\text{m,s,a,p}}$, Pi-Par$_{\text{m,s,a,p}}$, and Pi-Cong$_{\text{m,s,a,p}}$. The first two cases follow by the Rule T-Res$_\text{B}$ or T-Par$_\text{B}$ and the induction hypothesis. In the last case, $P \equiv Q$, $Q \longmapsto Q'$, and $Q' \equiv P'$ for some $Q, Q' \in (\mathcal{P}_\text{a}^{\sim} : \mathbb{T}_\text{B})$, $Q, Q' \in \left(\mathcal{P}_\text{p}^{\sim} : \mathbb{T}_\text{B}\right)$, or $Q, Q' \in (\mathcal{P}_\text{a}^{=,\sim} : \mathbb{T}_\text{B})$, respectively. Without loss of generality let us assume that this is the only application of Pi-Cong$_{\text{m,s,a,p}}$ in $P \longmapsto P'$. Let R, R' be such that $Q \equiv R$, $Q' \equiv R'$, and neither R nor R' contains unguarded subterms guarded by an already resolved match $[a = a]$. Then, by Pi-Cong$_{\text{m,s,a,p}}$, also $P \equiv R$, $R \longmapsto R'$, and $R' \equiv P'$. This time, R and R' do not have a match that is not already in P or P', respectively. Moreover, by Lemma 6.2.14, $\Gamma \vdash P$ implies that Γ is closed for P, i.e., provides a type for each free name of P. By assumption, Γ is also closed for P'. Since the only rule of structural congruence that allows to introduce free names is the rule that introduces matches, Γ is also closed for R, and R'. By Lemma 6.2.13, then $\Gamma \vdash P$ and $P \equiv R$ imply $\Gamma \vdash R$. By the induction hypothesis $\Gamma \vdash R$ and $R \longmapsto R'$ imply $\Gamma \vdash R'$. Finally, by Lemma 6.2.13, $\Gamma \vdash R'$ and $R' \equiv P'$ imply $\Gamma \vdash P'$.

□

It remains to show that the typed encodings $\mathcal{T}_\text{B}^1[\![\cdot]\!]_\text{a}^\text{s}$, $\mathcal{T}_\text{B}^2[\![\cdot]\!]_\text{p}^\text{m}$, and $\mathcal{T}_\text{B}^3[\![\cdot]\!]_\text{a}^\text{m}$ are well-typed with respect to some appropriate type environments. Type environments cover the types of the free names of the term we analyse. Since $[\![\cdot]\!]_\text{p}^\text{m}$ and $[\![\cdot]\!]_\text{a}^\text{m}$ use translated source term names only as values and introduces all other names under restriction except for the outermost occurrences of the request channels p_o and p_i, for all source terms $S \in \mathcal{P}_\text{m}$, appropriate type environment are

$$\Gamma_{[\![S]\!]_\text{a}^\text{m}} = \{ p_o : \mathfrak{o}, p_i : \mathfrak{i} \} \cup \{ \varphi_\text{a}^\text{m}(x) : \mathfrak{v}_\mathfrak{n} \mid x \in \mathsf{fn}(S) \} \quad \text{and}$$
$$\Gamma_{[\![S]\!]_\text{p}^\text{m}} = \{ p_o : \mathfrak{o}', p_i : \mathfrak{i}' \} \cup \{ \varphi_\text{a}^\text{m}(x) : \mathfrak{v}_\mathfrak{n} \mid x \in \mathsf{fn}(S) \}.$$

Note that, because of Lemma 6.2.10, it is no problem if a type environment contains unnecessary type assignments as long as these are not in conflict with a type assignment that can be derived for the corresponding typed term.

Lemma 6.2.17. *The encoding* $\mathcal{T}_{\mathrm{B}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ *is well-typed with respect to* $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}}$.

Proof. An encoding is well-typed if each encoded term is well-typed. Hence, we perform an induction over the structure of source terms. Note that no source term can contain infinitely many free names. We conclude that the type environment is finite for each source term $S \in \mathcal{P}_{\mathrm{m}}$. Let $\Gamma = \Gamma_{[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}}$.

Base Case: In $\mathcal{P}_{\mathrm{m}}$ there are two terms without subterms, namely $\mathsf{0}$ and $\checkmark$. $\mathcal{T}_{\mathrm{B}}^{3}[\![\mathsf{0}]\!]_{\mathrm{a}}^{\mathrm{m}} = (\nu l{:}\mathfrak{l})\, l(t,f)\,.\,\big((\nu v_t{:}\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\big)$ and $\mathcal{T}_{\mathrm{B}}^{3}[\![\checkmark]\!]_{\mathrm{a}}^{\mathrm{m}} = \checkmark$. The second case, i.e., the type judgement $\Gamma \vdash \checkmark$, follows directly by T-Succ$_{\mathrm{B}}$. For the first case, remember that $\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))$. The proof is given by the derivation

$$\dfrac{\dfrac{\dfrac{}{\Gamma, l{:}\mathfrak{l} \vdash l{:}\mathfrak{l}}\text{T-Name}_{\mathrm{B}} \quad \dfrac{D}{\Gamma' \vdash (\nu v_t{:}\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle}\text{T-Res}_{\mathrm{B}}}{\Gamma, l{:}\mathfrak{l} \vdash l(t,f)\,.\,\big((\nu v_t{:}\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\big)}\text{T-In}_{\mathrm{B}}}{\Gamma \vdash (\nu l{:}\mathfrak{l})\, l(t,f)\,.\,\big((\nu v_t{:}\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\big)}\text{T-Res}_{\mathrm{B}}$$

where

$$D = \dfrac{\dfrac{}{\Gamma', v_t{:}\mathfrak{v}_\top \vdash t{:}\sharp(\mathfrak{v}_\top)}\text{T-Name}_{\mathrm{B}} \quad \dfrac{}{\Gamma', v_t{:}\mathfrak{v}_\top \vdash v_t{:}\mathfrak{v}_\top}\text{T-Name}_{\mathrm{B}}}{\Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top)\,, f{:}\sharp(\mathfrak{v}_\bot)\,, v_t{:}\mathfrak{v}_\top \vdash \overline{t}\langle v_t\rangle}\text{T-Out}_{\mathrm{B}}$$

and $\Gamma' = \Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top)\,, f{:}\sharp(\mathfrak{v}_\bot)$.

Induction Hypothesis:

$$\forall S \in \mathcal{P}_{\mathrm{m}}\,.\,\Gamma_{[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}} \vdash \mathcal{T}_{\mathrm{B}}^{3}[\![S]\!]_{\mathrm{a}}^{\mathrm{m}} \tag{IH}$$

Induction Step: We have to prove that $\Gamma \vdash \mathcal{T}_{\mathrm{B}}^{3}[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}$ for the cases when S is a restricted term, a parallel composition, a sum, or a replicated input. The corresponding derivations are huge but very simple. For each step in each case there applies exactly one rule of Figure 6.4. To show how the induction hypothesis is applied in these derivations we present the derivation of the first case. Note that $\Gamma_{[\![S']\!]_{\mathrm{a}}^{\mathrm{m}}} = \Gamma_{[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}}, \varphi_{\mathrm{a}}^{\mathrm{m}}(x){:}\mathfrak{v}_{\mathfrak{n}}$ and $\Gamma = \Gamma_{[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}}$.

$$\dfrac{\dfrac{}{\Gamma, \varphi_{\mathrm{a}}^{\mathrm{m}}(x){:}\mathfrak{v}_{\mathfrak{n}} \vdash \mathcal{T}_{\mathrm{B}}^{3}[\![S']\!]_{\mathrm{a}}^{\mathrm{m}}}(IH) \text{ and Lemma 6.2.11}}{\Gamma \vdash (\nu \varphi_{\mathrm{a}}^{\mathrm{m}}(x){:}\mathfrak{v}_{\mathfrak{n}})\, \mathcal{T}_{\mathrm{B}}^{3}[\![S']\!]_{\mathrm{a}}^{\mathrm{m}}}\text{T-Res}_{\mathrm{B}}$$

The derivations of the other cases are postponed to the appendix in Section A.1.1.

For sake of completeness note that name clashes do not cause any problems here, because syntactically equal names in $\mathcal{T}_{\mathrm{B}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ have also the same type. However, we obtain the same result if we apply α-conversion to avoid name clashes in $\mathcal{T}_{\mathrm{B}}^{3}[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}$. □

The proof of well-typedness of $\mathcal{T}^2_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is very similar.

Lemma 6.2.18. *The encoding $\mathcal{T}^2_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is well-typed with respect to $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$.*

The encoding function $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ does not introduces free names but can be typed only with respect to well-structured source terms.

Lemma 6.2.19. *For all source terms $S \in \mathcal{P}_{\mathrm{s}}$ that are well-structured with respect to $\mathbb{T}_{\mathrm{s}}$ and $\mathcal{T}_S$, the encoding $\mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is well-typed with respect to $\Gamma_{[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}} = \widetilde{\mathcal{T}_S}$.*

The proof of these two Lemmata are postponed to the appendix in Section A.1.1.

With the subject reduction lemma we conclude that all target terms are well-typed. Unfortunately, as already discussed above, $\mathcal{T}^3_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is not a function from $\mathcal{P}_{\mathrm{m}}$ into $\mathcal{P}^{=,\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}}$ but leads to an intermediate language that allows for polyadic communication, i.e., to $\mathcal{P}^{=,\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}}$.

Lemma 6.2.20. *All target terms of $\mathcal{T}^1_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $\mathcal{T}^2_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $\mathcal{T}^3_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, i.e., all terms $P_1 \in (\mathcal{P}^{\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^1_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $P_2 \in (\mathcal{P}^{\sim}_{\mathrm{p}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^2_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, and $P_3 \in (\mathcal{P}^{=,\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^3_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, are modulo structural congruence well-typed with respect to $\Gamma_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ (and well-structured source terms), $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, and $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, respectively.*

Proof. By Lemmata 6.2.19, 6.2.18, and 6.2.17, all encoded source terms are well-typed. The target languages of the first two encodings do not contain a match prefix. Hence, by Lemma 6.2.16, all target terms $P_1 \in (\mathcal{P}^{\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^1_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ and $P_2 \in (\mathcal{P}^{\sim}_{\mathrm{p}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^2_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$ are well-typed with respect to $\Gamma_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ (and well-structured source terms) and $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, respectively.

We observe that in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ match is used only in the translation of the parallel operator on (bound) occurrences of translated source term names of type $\mathfrak{v}_{\mathfrak{n}}$. The side condition "modulo structural congruence" allows us to abstract from unnecessary matches, i.e., from matches that do not result from the encoding function but are introduced by the rule $[a = a]\, P \equiv P$ of structural congruence. Thus, for all target terms $P_3 \in (\mathcal{P}^{=,\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})\restriction_{\mathcal{T}^3_{\mathrm{B}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ without unnecessary matches the type environment $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ is closed. By Lemma 6.2.16, we conclude that for all P_3 there is some $P_3' \equiv P_3$ that is well-typed with respect to $\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$. □

In the next section we show that the basic type system can be extended to cover also the abbreviations of channel with multiplicity greater one.

6.2.3. Types with Behaviour

As already explained by Milner in [Mil93b], the type system described above cannot type the translation of polyadic into monadic communication which is discussed in Section 5.4.

Example 6.2.21. Consider an output $\overline{y}\langle z_1, z_2\rangle$ of multiplicity two. Its translation into $\pi^{=}_{\mathrm{a}}$ is given by the term:

$$(\nu u)\,(\overline{y}\langle u\rangle \mid u(u_1)\,.\,(\overline{u_1}\langle z_1\rangle \mid u(u_2)\,.\overline{u_2}\langle z_2\rangle))$$

Moreover, let us assume that we are able to assign a type to z_1, say T_1, and to z_2, say T_2. With respect to the type system given above, we assign the type $\sharp(T_1)$ to u_1 and $\sharp(T_2)$ to u_2. If $T_1 = T_2$ then the type of u is simply $\sharp(\sharp(T_1))$ and y is typed by $\sharp(\sharp(\sharp(T_1)))$. But if $T_1 \neq T_2$, e.g. because z_1 is a value and z_2 is a link or simply because we want to distinguish z_1 from z_2, we cannot assign a type to u or y.

The problem is that u is used two times to transmit different values. Intuitively, its type should be something like $\sharp(\sharp(T_1)) \vee \sharp(\sharp(T_2))$, to outline that over u two different kinds of arguments can be transmitted. However, allowing such a type would cause other problems and obviously it reduces the amount of information we can derive from a well-typed process. It is much easier to reason about a link that always transmits values of a specific kind than to reason about a link that may unpredictably choose to transmit either this or that kind of value. Moreover, a type like $\sharp(\sharp(T_1)) \vee \sharp(\sharp(T_2))$ does not reveal all information available for the process described in the example above. There, u is not unpredictable but always sends first a value of type $\sharp(\sharp(T_1))$ and then a value of type $\sharp(\sharp(T_2))$. The protocol that underlies the translation of polyadic communication determines the behaviour of u in a very strict way. In this sense the behaviour of u can be considered as somehow static or better predetermined. [Yos96] and [QW05] present two different but related ways to transfer predetermined behaviour in a type system. The resulting types are not as static as the basic types in Section 6.2.2, because they change over time but these changes are completely predetermined. [Yos96] and [QW05] mainly consider the translation of the polyadic communication in the case of a synchronous target language, although both give hints on how to adapt this concept to the asynchronous case. Moreover, both represent the behaviour of types as graphs. However, we mainly focus on the concept in [QW05], because it results in a type system that is a little bit simpler. More precisely, we shortly revisit the type system presented in [QW00], which directly focuses on the case of an asynchronous target language, and then will adapt it to our needs.

[QW00] introduces two new kinds of graph based link types, called *m-sorts*. The "m" stands for monadic and sorts refers to the type system for the polyadic pi-calculus introduced in [Mil93b]. We will adapt this notation and consequently denote the resulting type system as *monadic type system* and its types as *monadic types*. Note that we introduce the monadic type system as an extension of the basic type system in Section 6.2.2. This allows us to reuse some notation and proofs as well as the simple presentation of types as basic types if possible. Moreover, we will again reduce the general concept to our needs, i.e., abandon type constructors but use a fixed set of types instead.

The first kind of m-sorts, called primary m-sorts, is defined in [QW00] by the set

$$\mathcal{S}_1^m = \{ \circ^s \mid s \in \mathcal{S} \} \cup \left\{ s^i \mid 1 \leq i \leq |\lambda(s)|, s \in \mathcal{S} \right\} \cup \{ \bullet \},$$

where $\mathcal{S}$ denotes the set of sorts, i.e., types, of the polyadic source language and $\lambda(s)$ returns the types of the arguments of a link type s, i.e., if $s = \sharp(T_1, \ldots, T_n)$ then $\lambda(s) = T_1, \ldots, T_n$, such that $|\lambda(s)|$ returns the multiplicity of a link type. Primary m-sorts are used to type names that are used to transmit arguments of different kinds

as the u in Example 6.2.21. More precisely, the monadic type system assigns different primary m-sorts to different occurrences of such names. At the beginning, to denote the creation of such a link, $\circ^s$ is assigned. In this state the name can be used only as argument of other links, as by y in $\overline{y}\langle u\rangle$ in Example 6.2.21. Then, for $u(u_1)$, its type becomes s^1 to represent $\sharp(\sharp(T_1))$, and then it becomes s^2 to represent $\sharp(\sharp(T_2))$ for $u(u_2)$ in Example 6.2.21. Finally, the type $\bullet$ is assigned to u to represent its destruction and ensure that it cannot be used any further—neither as link nor as send value.

Secondary m-sorts are used to type names like u_1 and u_2 in Example 6.2.21 that are used to transmit the arguments of the original polyadic communication. They are defined in [QW00] by the set

$$\mathcal{S}_2^m = \left\{ \circ^{s^i} \mid 1 \leq i \leq |\lambda(s)|, s \in \mathcal{S} \right\} \cup \left\{ \delta^{s^i} \mid 1 \leq i \leq |\lambda(s)|, s \in \mathcal{S} \right\}.$$

Again the type $\circ^{s^i}$ is used to type a name for being used as argument in a communication. Consequently, in Example 6.2.21 the type $\circ^{s^1}$ is assigned to u_1 for $u(u_1)$ and $\circ^{s^2}$ is assigned to u_2 for $u(u_2)$, where s is the type of y in the polyadic source language, i.e., $s = \sharp(T_1, T_2)$ in Example 6.2.21. Then, to transmit the original arguments, the type δ^{s^i} is assigned. This result in the type δ^{s^1} for u_1 in $\overline{u_1}\langle z_1\rangle$ and δ^{s^2} for u_2 in $\overline{u_2}\langle z_2\rangle$. Note that, by the translation of polyadic communication, the names used to transmit the original arguments—as u_1 and u_2 here—are always first transmitted and then used exactly once as link. Because of that it is not necessary here to explicitly mark destruction.

The behaviour of primary and secondary m-sorts can best be presented in labelled graphs whose nodes are the primary m-sorts and whose edges are labelled by the secondary m-sorts and original types as explained in [QW00]. As an example, the behaviour of the type of the auxiliary names u, u_1, and u_2 in the translation of $\overline{y}\langle z_1, z_2\rangle$ in Example 6.2.21 is represented by the chain

$$\circ^s \longrightarrow s^1 \xrightarrow{\delta^{s^1} T_1} s^2 \xrightarrow{\delta^{s^2} T_2} \bullet$$

where $s = \sharp(T_1, T_2)$.

By Example 6.2.21, we know that the basic type system is not well suited to type the links that are typed in [QW00] by primary m-sorts, because this would force us to identify the parameters of the respective polyadic channel by the same type. Consequently, we introduce new link types to capture this kind of type. More precisely, we use four different primary m-sorts to type the translation of the polyadic communication on receiver locks, output requests, input requests, and the second chain locks of replicated input, respectively. However, at this stage, there is no need to use secondary m-sorts. The basic type system is sufficient to type the auxiliary links that are typed in [QW00] with secondary m-sorts. The reason for this additional kind of types in [QW00] is to prove linearity of the secondary m-sorts and encapsulation of the corresponding links with respect to the links typed by primary m-sorts. We decide here to separate these concepts and, accordingly, discuss linear types in Section 6.2.4. So, we type names like the u_1 and u_2 in Example 6.2.21 by types similar to basic types.

There are five different kinds of polyadic communication with multiplicity greater than one in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$: (1) communication on sum locks and second chain locks of replicated inputs of multiplicity two, (2) communication on input request channels of multiplicity three, (3) communication on output request channels of multiplicity four, and (4) communication on receiver locks of multiplicity five. To type these links, we first unfold the abbreviation of polyadic communication as explained in Definition 5.4.1. Then the auxiliary links used to transmit the values of the polyadic communication are typed by basic types, but the auxiliary links carrying these links are assigned primary m-sorts as described above. Finally, the type of the original polyadic channel is changed into a link type carrying a single parameter which is the primary m-sort. To avoid confusion between the link types of the basic type system and the link types of the auxiliary links introduced to unfold polyadic communications we use another link type symbol, namely $\natural(\cdot)$. Moreover, because of that new symbol, it is easier to show that the auxiliary links are indeed used only to unfold polyadic communications in the encoding functions. Note that we ignore the difference between $\sharp(\cdot)$ and $\natural(\cdot)$ in the typing rules below.

Definition 6.2.22 (Translated Basic Types)**.** Let $T \in \mathbb{T}_{\mathrm{B}}$ be a basic type, then its translation into the corresponding monadic type, denoted by $\widehat{T}$, is defined as:

$$\widehat{T} = \begin{cases} T, & \text{if } T \in \mathbb{V}_{\mathrm{B}} \\ \sharp\left(\widehat{T'}\right), & \text{if } T = \sharp(T') \wedge T' \in \mathbb{T}_{\mathrm{B}} \\ \sharp\left(\circ \triangleright \sharp\left(\natural\left(\widehat{T_1}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet\right), & \text{if } T = \sharp(T_1, \ldots, T_n) \wedge n > 1 \end{cases}$$

We extend the definition of basic types to the translation of type assignments, i.e., $\widehat{x{:}T} = x{:}\widehat{T}$ for $x \in \mathcal{N}$ and $T \in \mathbb{T}_{\mathrm{B}}$.

Accordingly, type environments Γ in the basic type system are translated into the corresponding type environment of the monadic type system, denoted by $\widehat{\Gamma}$, by replacing all type assignments $y{:}T$ in Γ by $y{:}\widehat{T}$.

As an example we consider the translation of an output $\overline{c_{r2}}\langle r_o, r_i\rangle$ on a second chain lock of replicated inputs and the corresponding input $c_{r2}(r_o, r_i).P$. As described in Definition 5.4.1,

$$\overline{c_{r2}}\langle r_o, r_i\rangle = (\nu u_{\sim,c})\left(\overline{c_{r2}}\langle u_{\sim,c}\rangle \mid u_{\sim,c}(u_o).\left(\overline{u_o}\langle r_o\rangle \mid u_{\sim,c}(u_i).\overline{u_i}\langle r_i\rangle\right)\right)$$

and

$$c_{r2}(r_o, r_i).P = c_{r2}(u_{\sim,c}).\left((\nu u_o)\left(\overline{u_{\sim,c}}\langle u_o\rangle \mid u_o(r_o).\left((\nu u_i)\left(\overline{u_{\sim,c}}\langle u_i\rangle \mid u_i(r_i).P'\right)\right)\right)\right),$$

where P' represents the term P after unfolding all remaining polyadic communications. In the basic type system we allocate the type $\mathfrak{o}$ to r_o, $\mathfrak{i}$ to r_i, and $\sharp(\mathfrak{o}, \mathfrak{i})$ to c_{r2}. Let $T_o = \mathfrak{o}$ and $T_i = \mathfrak{i}$, i.e., $\sharp(\mathfrak{o}, \mathfrak{i}) = \sharp(T_o, T_i)$ and obviously $T_o \neq T_i$. Hence, by Definition 6.2.22, $\overline{c_{r2}}\langle r_o, r_i\rangle$ is translated into

$$(\nu u_{\sim,c}{:}\circ \triangleright \sharp(\natural(\mathfrak{o})) \triangleright \sharp(\natural(\mathfrak{i})) \triangleright \bullet)\left(\overline{c_{r2}}\langle u_{\sim,c}\rangle \mid u_{\sim,c}(u_o).\left(\overline{u_o}\langle r_o\rangle \mid u_{\sim,c}(u_i).\overline{u_i}\langle r_i\rangle\right)\right),$$

where the type $\circ \triangleright \sharp(\natural(\mathfrak{o})) \triangleright \sharp(\natural(\mathfrak{i})) \triangleright \bullet$ depicts that $u_{\sim,c}$ is first transmitted over c_{r2} then used to transmit u_o, which is of the type $\natural(\mathfrak{o})$, and finally used to transmit u_i of type $\natural(\mathfrak{i})$. Consequently, the input $c_{r2}(r_o, r_i).P$ is translated into

$$c_{r2}(u_{\sim,c}).\left((\nu u_o : \natural(\mathfrak{o}))\left(\overline{u_{\sim,c}}\langle u_o\rangle \mid u_o(r_o).\left((\nu u_i : \natural(\mathfrak{i}))\left(\overline{u_{\sim,c}}\langle u_i\rangle \mid u_i(r_i).\widehat{P}\right)\right)\right)\right).$$

Since c_{r2} is now used to swap the channel $u_{\sim,c}$ of type $\circ \triangleright \sharp(\natural(\mathfrak{o})) \triangleright \sharp(\natural(\mathfrak{i})) \triangleright \bullet$, its type becomes $\sharp(\circ \triangleright \sharp(\natural(\mathfrak{o})) \triangleright \sharp(\natural(\mathfrak{i})) \triangleright \bullet)$. In the same manner the types of receiver locks and the links of input and output requests, as well as the necessary auxiliary channels, are derived.

Again, because of the very strict name schema, the encoding function $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ can completely be typed by a finite number of different types depicted in Figure 6.5. The set of monadic types inherits all basic value types and the basic link types $\sharp(\mathfrak{v}_\top)$, $\sharp(\mathfrak{v}_\bot)$, $\mathfrak{s}$, and $\sharp(\mathfrak{v}_\mathfrak{n})$.

In the same manner also the monadic types of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ in Figure 6.6 and of the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ in Figure 6.7 are derived. Again, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ leads to finitely many types. But, in the case of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, the number of monadic types depends on the number of types in the source language as it was already the case for the basic type system. For each link type T_S in the source the unfolding of polyadic communication introduces two auxiliary links u_{T_S} and $u_{\sim,\varphi^{\mathrm{s}}_{\mathrm{a}}(z)}$ that are typed by $\natural(T_S)$ and $\mathfrak{n}_{\circ,T_S} = \circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(T_S)) \triangleright \bullet$, respectively.

Again, the types of the monadic type system are the unification of the types in Figures 6.5, 6.6, and 6.7.

Definition 6.2.23 (Monadic Types)**.** Let $\mathbb{T}_{\mathrm{s}}$ be the types of the type system of the source language π_{s}. The types of the monadic type system, called *monadic types*, are given by the monadic value types $\mathbb{V}_{\mathrm{M}}$, the monadic link types $\mathbb{L}_{\mathrm{M}}$, and the monadic types $\mathbb{T}_{\mathrm{M}}$. The monadic value types are defined as for the basic type system by the set

$$\mathbb{V}_{\mathrm{M}} \triangleq \mathbb{V}_{\mathrm{B}} = \{\, \mathfrak{v}_\mathfrak{n}, \mathfrak{v}_\top, \mathfrak{v}_\bot, \mathfrak{v}_\mathfrak{s}, \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \,\} \cup \{\, V_S \mid V_S \in \mathbb{T}_{\mathrm{s}} \wedge \forall T_S \in \mathbb{T}_{\mathrm{s}} \,.\, V_S \neq \sharp(T_S) \,\}$$

The set of monadic link types is defined as

$$\begin{aligned}
\mathbb{L}_{\mathrm{M}} = {} & \{\, \sharp(\mathfrak{v}_\top), \sharp(\mathfrak{v}_\bot), \mathfrak{l}, \mathfrak{s}, \mathfrak{r}, \mathfrak{o}, \mathfrak{i}, \sharp(\mathfrak{i}), \sharp(\mathfrak{o}), \sharp(\mathfrak{v}_\mathfrak{n}), \sharp(\mathfrak{c}_\circ), \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), \mathfrak{r}', \mathfrak{o}', \mathfrak{i}', \mathfrak{t}_\mathfrak{o}, \mathfrak{t}_\mathfrak{i} \,\} \\
& \cup \{\, \natural(\mathfrak{v}_\mathfrak{n}), \natural(\sharp(\mathfrak{v}_\top)), \natural(\sharp(\mathfrak{v}_\bot)), \natural(\mathfrak{l}), \natural(\mathfrak{s}), \natural(\mathfrak{r}), \natural(\mathfrak{o}), \natural(\mathfrak{i}), \mathfrak{l}_\circ, \mathfrak{c}_\circ, \mathfrak{i}_\circ, \mathfrak{o}_\circ, \mathfrak{r}_\circ \,\} \\
& \cup \{\, \natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})), \natural(\mathfrak{r}'), \mathfrak{t}_{\mathfrak{i},\circ}, \mathfrak{i}'_\circ, \mathfrak{t}_{\mathfrak{o},\circ}, \mathfrak{o}'_\circ, \mathfrak{r}'_\circ \,\} \\
& \cup \{\, \natural(V_S) \mid V_S \in \mathbb{T}_{\mathrm{s}} \wedge \forall T_S \in \mathbb{T}_{\mathrm{s}} \,.\, V_S \neq \sharp(T_S) \,\} \\
& \cup \left\{\, \sharp(\mathfrak{n}_{\circ,T_S}), \natural(T_S), \mathfrak{n}_{\circ,T_S} \mid \sharp(T'_S) \in \mathbb{T}_{\mathrm{s}} \wedge T_S = \widetilde{T'_S} \,\right\}.
\end{aligned}$$

As in the basic type system, $\mathbb{T}_{\mathrm{M}} = \mathbb{V}_{\mathrm{M}} \cup \mathbb{L}_{\mathrm{M}}$.

The Figures 6.5, 6.6, and 6.7 also specify the set of type assignments $\mathcal{T}^3_{\mathrm{M}}$, $\mathcal{T}^2_{\mathrm{M}}$, and $\mathcal{T}^1_{\mathrm{M}}$, respectively. Accordingly, the typed variant of a target term $P_3 \in \mathcal{P}^{=}_{\mathrm{a}} \restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, $P_2 \in \mathcal{P}_{\mathrm{p}} \restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $P_1 \in \mathcal{P}_{\mathrm{a}} \restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ in the monadic type system is given (after unfolding all

Description	Names	Type
source term names	$\varphi_{\mathrm{a}}^{\mathrm{m}}(x)\,, \varphi_{\mathrm{a}}^{\mathrm{m}}(y)\,,$ $\varphi_{\mathrm{a}}^{\mathrm{m}}(z)\,, y, y', z$	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\sharp(\mathfrak{v}_{\top})$
	f	$\sharp(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\mathfrak{l}_{\circ})$
sender locks	s	$\mathfrak{s} = \sharp(\mathfrak{v}_{\mathfrak{s}})$
receiver locks	r	$\mathfrak{r} = \sharp(\mathfrak{r}_{\circ})$
output requests	$p_o, m_o, p_{o,up},$ $m_{o,up}, r_o, r_{o,up}$	$\mathfrak{o} = \sharp(\mathfrak{o}_{\circ})$
input requests	$p_i, m_i, p_{i,up},$ $m_{i,up}, r_i, r_{i,up}$	$\mathfrak{i} = \sharp(\mathfrak{i}_{\circ})$
chain locks	c_o	$\sharp(\mathfrak{i})$
	c_i	$\sharp(\mathfrak{o})$
	c_{r1}	$\sharp(\mathfrak{v}_{\mathfrak{n}})$
	c_{r2}	$\sharp(\mathfrak{c}_{\circ})$
auxiliary links	u_n	$\natural(\mathfrak{v}_{\mathfrak{n}})$
	u_t	$\natural(\sharp(\mathfrak{v}_{\top}))$
	u_f	$\natural(\sharp(\mathfrak{v}_{\bot}))$
	u_l	$\natural(\mathfrak{l})$
	u_s	$\natural(\mathfrak{s})$
	u_r	$\natural(\mathfrak{r})$
	u_o	$\natural(\mathfrak{o})$
	u_i	$\natural(\mathfrak{i})$
	$u_{\sim,l}$	$\mathfrak{l}_{\circ} = \circ \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\top}))) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\bot}))) \triangleright \bullet$
	$u_{\sim,c}$	$\mathfrak{c}_{\circ} = \circ \triangleright \sharp(\natural(\mathfrak{o})) \triangleright \sharp(\natural(\mathfrak{i})) \triangleright \bullet$
	$u_{\sim,i}$	$\mathfrak{i}_{\circ} = \circ \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{r})) \triangleright \bullet$
	$u_{\sim,o}$	$\mathfrak{o}_{\circ} = \circ \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \bullet$
	$u_{\sim,r}$	$\mathfrak{r}_{\circ} =$ $\circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \bullet$

Figure 6.5.: Monadic Types in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$.

Description	Names	Type
source term names	$\varphi_{\mathrm{p}}^{\mathrm{m}}(x)\,, \varphi_{\mathrm{p}}^{\mathrm{m}}(y)\,,$ $\varphi_{\mathrm{p}}^{\mathrm{m}}(z)\,, y, z$	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\sharp(\mathfrak{v}_{\top})$
	f	$\sharp(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\mathfrak{l}_{\circ})$
sender and receiver locks	s_1, r_1, v, w	$\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$
	s_2	$\mathfrak{s} = \sharp(\mathfrak{v}_{\mathfrak{s}})$
	r_2	$\mathfrak{r}' = \sharp(\mathfrak{r}'_{\circ})$
output requests	$p_o, p_{o,up}$	$\mathfrak{o}' = \sharp(\mathfrak{o}'_{\circ})$
input requests	$p_i, p_{i,up}$	$\mathfrak{i}' = \sharp(\mathfrak{i}'_{\circ})$
tags	o	$\mathfrak{t}_{\mathfrak{o}} = \sharp(\mathfrak{t}_{\mathfrak{o},\circ})$
	i	$\mathfrak{t}_{\mathfrak{i}} = \sharp(\mathfrak{t}_{\mathfrak{i},\circ})$
auxiliary links	u_n	$\natural(\mathfrak{v}_{\mathfrak{n}})$
	u_t	$\natural(\sharp(\mathfrak{v}_{\top}))$
	u_f	$\natural(\sharp(\mathfrak{v}_{\bot}))$
	u_l	$\natural(\mathfrak{l})$
	$u_{s,r}$	$\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))$
	u_s	$\natural(\mathfrak{s})$
	$u_{r'}$	$\natural(\mathfrak{r}')$
	$u_{\sim,l}$	$\mathfrak{l}_{\circ} = \circ \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\top}))) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\bot}))) \triangleright \bullet$
	$u_{\sim,t_i}$	$\mathfrak{t}_{\mathfrak{i},\circ} = \circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \sharp(\natural(\mathfrak{r}')) \triangleright \bullet$
	$u_{\sim,i'}$	$\mathfrak{i}'_{\circ} =$ $\circ \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \sharp(\natural(\mathfrak{r}')) \triangleright \bullet$
	$u_{\sim,t_o}$	$\mathfrak{t}_{\mathfrak{o},\circ} =$ $\circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \bullet$
	$u_{\sim,o'}$	$\mathfrak{o}'_{\circ} = \circ \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright$ $\sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \bullet$
	$u_{\sim,r}$	$\mathfrak{r}'_{\circ} = \circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{l})) \triangleright$ $\sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(\mathfrak{v}_{\mathfrak{n}})) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))) \triangleright \bullet$

Figure 6.6.: Monadic Types in $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$.

Description	Names	Type
source term names	$\varphi^{\mathrm{s}}_{\mathrm{a}}(x)$	V_S
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$	$\sharp(\mathfrak{n}_{\circ,T_S})$
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(z)$	$\sharp(\mathfrak{n}_{\circ,T_S})$
		V_S
		$\sharp(\mathfrak{n}_{\circ,T_S})$
auxiliary values	v_t	$\mathfrak{v}_\top$
	v_f	$\mathfrak{v}_\bot$
	v_s	$\mathfrak{v}_\mathfrak{s}$
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$
booleans	t	$\sharp(\mathfrak{v}_\top)$
	f	$\sharp(\mathfrak{v}_\bot)$
sum locks	l, l'	$\mathfrak{l} = \sharp(\mathfrak{l}_\circ)$
sender locks	s	$\mathfrak{s} = \sharp(\mathfrak{v}_\mathfrak{s})$
receiver locks	r	$\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$
auxiliary links	u_{T_S}	$\natural(T_S)$
	u_t	$\natural(\sharp(\mathfrak{v}_\top))$
	u_f	$\natural(\sharp(\mathfrak{v}_\bot))$
	u_l	$\natural(\mathfrak{l})$
	u_s	$\natural(\mathfrak{s})$
	$u_{\sim,l}$	$\mathfrak{l}_\circ = \circ \triangleright \sharp(\natural(\sharp(\mathfrak{v}_\top))) \triangleright \sharp(\natural(\sharp(\mathfrak{v}_\bot))) \triangleright \bullet$
	$u_{\sim,T_S}$	$\mathfrak{n}_{\circ,T_S} = \circ \triangleright \sharp(\natural(\mathfrak{l})) \triangleright \sharp(\natural(\mathfrak{s})) \triangleright \sharp(\natural(T_S)) \triangleright \bullet$

Figure 6.7.: Monadic Types in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$.

polyadic communications) as $\mathcal{T}^3_{\mathrm{M}}(P_3)$, $\mathcal{T}^2_{\mathrm{M}}(P_2)$, and $\mathcal{T}^1_{\mathrm{M}}(P_1)$, respectively. We extend the translation of basic types to the translation of typed terms in the basic type system to typed terms in the monadic type system.

Definition 6.2.24 (Translated Basic Typed Terms)**.** Let P be a typed process in the basic type system, i.e., $P \in (\mathcal{P}^{\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})$, $P \in \big(\mathcal{P}^{\sim}_{\mathrm{p}} : \mathbb{T}_{\mathrm{B}}\big)$, or $P \in (\mathcal{P}^{=,\sim}_{\mathrm{a}} : \mathbb{T}_{\mathrm{B}})$. Without loss of generality let us assume that P is free of name clashes, i.e., $\mathsf{fn}(P) \cap \mathsf{bn}(P) = \emptyset$ and no name is bound twice in P, and let the set $\{\, u, u_1, \ldots, u_n \mid n \in \mathbb{N} \,\}$ contain only fresh names, i.e., $\{\, u, u_1, \ldots, u_n \mid n \in \mathbb{N} \,\} \cap \mathsf{n}(P) = \emptyset$. Then, there exists an untyped version P' of P and a set of type assignments $\mathcal{T}_{\mathrm{B}}$ in the basic type system for all restricted names in P such that $P = \mathcal{T}_{\mathrm{B}}(P')$. Moreover, let Γ be a type environment that is not in conflict with $\mathcal{T}_{\mathrm{B}}$ and provides a type for each free name of P.

Then the translation of P with respect to Γ into the corresponding typed process of $\mathcal{P}_{\mathrm{a}} : \mathbb{T}_{\mathrm{M}}$, $\mathcal{P}_{\mathrm{p}} : \mathbb{T}_{\mathrm{M}}$, or $\mathcal{P}^{=}_{\mathrm{a}} : \mathbb{T}_{\mathrm{M}}$, denoted by $\widehat{P}$, is obtained from P by replacing recursively

any output $\overline{y_1}\langle z_{1,1},\ldots,z_{1,n_1}\rangle$ with $n_1 > 1$ and $y_1 : \sharp(T_{1,1},\ldots,T_{1,n_1}) \in \mathcal{T}_{\mathrm{B}} \cup \Gamma$ by

$$\Big(\nu u : \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_{1,1}}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_{1,n_1}}\Big)\Big) \triangleright \bullet\Big) \\ (\overline{y_1}\langle u\rangle \mid u(u_1).(\overline{u_1}\langle z_{1,1}\rangle \mid \ldots \mid u(u_{n_1}).(\overline{u_{n_1}}\langle z_{1,n_1}\rangle)\ldots))$$

any output $\overline{y_{2,1}\cdot y_{2,2}}\langle z_{2,1},\ldots,z_{2,n_2}\rangle$ with $n_2 > 1$ and $y_{2,2} : \sharp(T_{2,1},\ldots,T_{2,n_2}) \in \mathcal{T}_{\mathrm{B}} \cup \Gamma$ by

$$\Big(\nu u : \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_{2,1}}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_{2,n_2}}\Big)\Big) \triangleright \bullet\Big) \\ (\overline{y_{2,1}\cdot y_{2,2}}\langle u\rangle \mid u(u_1).(\overline{u_1}\langle z_{2,1}\rangle \mid \ldots \mid u(u_{n_2}).(\overline{u_{n_2}}\langle z_{2,n_2}\rangle)\ldots))$$

any input $y_3(x_{3,1},\ldots,x_{3,n_3}).P'$ with $n_3 > 1$ and $y_3 : \sharp(T_{3,1},\ldots,T_{3,n_3}) \in \mathcal{T}_{\mathrm{B}} \cup \Gamma$ by

$$y_3(u).\Big(\Big(\nu u_1 : \natural\Big(\widehat{T_{3,1}}\Big)\Big)\Big(\overline{u}\langle u_1\rangle \mid u_1(x_{3,1}).\Big(\ldots \\ \Big(\nu u_{n,3} : \natural\Big(\widehat{T_{3,n_3}}\Big)\Big)\Big(\overline{u}\langle u_{n_3}\rangle \mid u_{n_3}(x_{3,n_3}).\widehat{P'}\Big)\ldots\Big)\Big)\Big)$$

any input $y_{4,1}\cdot y_{4,2}(x_{4,1},\ldots,x_{4,n_4}).P'$ with $n_4 > 1$ and $y_{4,2} : \sharp(T_{4,1},\ldots,T_{4,n_4}) \in \mathcal{T}_{\mathrm{B}} \cup \Gamma$ by

$$y_{4,1}\cdot y_{4,2}(u).\Big(\Big(\nu u_1 : \natural\Big(\widehat{T_{4,1}}\Big)\Big)\Big(\overline{u}\langle u_1\rangle \mid u_1(x_{4,1}).\Big(\ldots \\ \Big(\nu u_{n,4} : \natural\Big(\widehat{T_{4,n_4}}\Big)\Big)\Big(\overline{u}\langle u_{n_4}\rangle \mid u_{n_4}(x_{4,n_4}).\widehat{P'}\Big)\ldots\Big)\Big)\Big)$$

and any replicated input $y_5^\star(x_{5,1},\ldots,x_{5,n_5}).P'$ with $n_5 > 1$ and $y_5 : \sharp(T_{5,1},\ldots,T_{5,n_5}) \in \mathcal{T}_{\mathrm{B}} \cup \Gamma$ by

$$y_5^\star(u).\Big(\Big(\nu u_1 : \natural\Big(\widehat{T_{5,1}}\Big)\Big)\Big(\overline{u}\langle u_1\rangle \mid u_1(x_{5,1}).\Big(\ldots \\ \Big(\nu u_{n,5} : \natural\Big(\widehat{T_{5,n_5}}\Big)\Big)\Big(\overline{u}\langle u_{n_5}\rangle \mid u_{n_5}(x_{5,n_5}).\widehat{P'}\Big)\ldots\Big)\Big)\Big)$$

Note that by Definition 5.4.1 the names of auxiliary links used to unfold the polyadic communications in the three encoding functions differ from the names used above. However the associated typed terms differ only by α-conversion, because all auxiliary links are restricted.

Observation 6.2.25. $\widehat{\mathcal{T}^3_{\mathrm{B}}(P_3)} \equiv_\alpha \mathcal{T}^3_{\mathrm{M}}(P'_3)$ with respect to $\mathcal{T}^3_{\mathrm{B}}$, where P'_3 is the result of unfolding all polyadic communications with respect to Definition 5.4.1. Similarly, $\widehat{\mathcal{T}^2_{\mathrm{B}}(P_2)} \equiv_\alpha \mathcal{T}^2_{\mathrm{M}}(P'_2)$ with respect to $\mathcal{T}^2_{\mathrm{B}}$ and $\widehat{\mathcal{T}^1_{\mathrm{B}}(P_1)} \equiv_\alpha \mathcal{T}^1_{\mathrm{M}}(P'_1)$ with respect to $\mathcal{T}^1_{\mathrm{B}}$.

Moreover, note that we define the translation of typed terms with respect to a type environment, because the typed term provides only the types of restricted channels and we need a basic link type as starting point for each translation of a polyadic communication. However, in the following we mainly consider the translation of type judgements which consist of a typed term and the corresponding type environment. If not explicitly stated otherwise, we silently translate the typed terms of type judgements with respect

to the type environment of the type judgement potentially extended by arbitrary types for all free names that do not occur in the type environment. Of course, in the case of well-typed terms, the type environment provides a type for all free names of the term as shown by Lemma 6.2.14. The sets $\mathcal{T}^3_{\mathrm{M}}$, $\mathcal{T}^2_{\mathrm{M}}$, and $\mathcal{T}^1_{\mathrm{M}}$ provide a type for all names that occur in the target terms of the encodings.

To capture the m-sorts like types as $\mathfrak{l}_\circ$ and their behaviour, type judgements are augmented with two additional sets, ranged over by $\Delta, \Delta', \Delta_1, \ldots$ and $\Psi, \Psi', \Psi_1, \ldots$, respectively. Given a type judgement $\Gamma \vdash P$ in the basic type system, $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ is the corresponding type judgement in the monadic type system, where initially $\Delta = \Psi = \emptyset$.[2] Δ and Ψ are very similar to type environments but instead of type assignments $x : T$ they contain assignment like elements $u : M$, there M is a part of a m-sort like monadic link type, i.e., in contrast to type assignments M is often not a type. These sets are used to capture the current state of m-sort like types. Therefore, Δ contains the respective information for input capabilities, while Ψ provides the corresponding information used to type outputs. Note that all m-sort like types in the type environment are complete.

Definition 6.2.26 (Monadic Type Judgements)**.** Let Γ be a type environment, Δ and Ψ be auxiliary sets that contain assignment like elements $u : M$ there $u \in \mathcal{N}$, and let P be a typed term with respect to monadic types. Then, $\Gamma; \Delta; \Psi \vdash P$ is a *monadic type judgement* if $(\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi)) \subseteq \mathsf{n}(\Gamma)$ and for all $u : M \in \Delta \cup \Psi$ there exists some $u : T \in \Gamma$ such that M is a part of T, i.e., $T = T_1 \triangleright \ldots \triangleright T_n$ with $n > 3$, $T_1 = \circ$, and $T_n = \bullet$ and either $M = T_i$ for some $1 < i < n$ or $M = T_{i_1} \triangleright \ldots \triangleright T_{i_m}$ for some $i_1, \ldots, i_m \in \{\, 1, \ldots, n \,\}$ such that $i_k < i_{k+1}$ for all $1 \leq k < m$.

Moreover, for all type environments or auxiliary sets M within monadic type judgements M_1, M_2 is defined only if $\mathsf{n}(M_1) \cap \mathsf{n}(M_2) = \emptyset$. In this case $M_1, M_2 = M_1 \cup M_2$.

Note that in the basic type system, we allow to add the same type assignment several times to the same type environment. By the way, by doing so we do not need to apply α-conversion in the proofs of Lemma 6.2.17, Lemma 6.2.18, or Lemma 6.2.19. However, in the type system defined within this subsection we require that two parts of a type environment or an auxiliary set do not share a name. This avoids confusion between the type environment and the auxiliary sets, if two or more unfoldings of polyadic communication are considered simultaneously.

Intuitively, Δ and Ψ capture information about the current state of primary m-sort like types. That is necessary to check the use of links typed by a m-sort like type. The axioms are not influenced by these information, because they are used to check the type of constants that do not contain names or to check the type of a name which does not require information about the actual state of its type. Hence, we recycle the typing Rule T-Name$_{\mathrm{B}}$ to derive judgements on type assignments. Moreover, we require that the protocol associated with a m-sort like type is finished before a constant is reached, i.e., $\mathsf{0}$ and $\checkmark$ are well-typed if $\Delta = \Psi = \emptyset$ (see Figure 6.8 at Page 191).

The Rule T-Res$_{\mathrm{B}}$ for restriction in the monadic type system is split into two rules; one for restriction on m-sort like types and the other for all remaining types (including

[2] In [QW00] Ψ is used to denote the original type environment. Hence, Γ and Ψ are swapped.

link types whose parameter is a m-sort like type). The rule covering restriction on m-sorts is denoted by T-Res-$\mathrm{M_M}$. Such a restriction marks the beginning of a translation of polyadic communication at the side of a sender. The new auxiliary link with the m-sort like type is transmitted and then used to receive other auxiliary links. Hence, Δ is initialised by T-Res-$\mathrm{M_M}$ to check the type of the continuation. To do so, the preceding $\circ$ is cut off. Moreover, T-Res-$\mathrm{M_M}$ allows to cut off more from the beginning of the m-sort like type, to enable typing of derivatives of well-typed terms. The subsequent rules will ensure that after each input the first part of the remaining m-sort is cut off. T-Res-$\mathrm{B_M}$ is similar to T-$\mathrm{Res_B}$, i.e., requires that the continuation is well-typed with respect to the type environment extended by an additional type assignment covering the type constraint of the typed restriction. To make the use of T-Res-$\mathrm{B_M}$ and T-Res-$\mathrm{M_M}$ unambiguous, T-Res-$\mathrm{B_M}$ is augmented with a side condition depicting that the restricted link is not typed by a m-sort like type. To cover structural congruence both rules have to allow for not empty sets Δ and Ψ in the precondition. We require that these sets remain unchanged except for the additional element added by T-Res-$\mathrm{M_M}$ to Δ.

Neither τ-prefix nor match occur in the translation of polyadic communication. Thus, the Rule T-$\mathrm{Tau_M}$ is obtained from T-$\mathrm{Tau_B}$ by adding the precondition $\Delta = \Psi = \emptyset$. Because of the match rule in structural congruence, we have to allow for arbitrary sets Δ and Ψ in the precondition of T-$\mathrm{Mat_M}$. But we require that these sets are not changed and are inherit by the subgoal of the subterm. The check of the type assignments for the names in the match is similar to T-$\mathrm{Mat_B}$.

The Rule T-$\mathrm{Out_B}$ is also split into two cases. The first rule, denoted by T-Out-$\mathrm{B_M}$, adds in comparison to T-$\mathrm{Out_B}$ the precondition $\Delta = \Psi = \emptyset$ and is used to type outputs, where the link is not typed by a m-sort. Note that this rule does not distinguish between link types $\sharp(\cdot)$ or link types $\natural(\cdot)$ of auxiliary links, i.e., implicitly T-Out-$\mathrm{B_M}$ represents two rules

$$\frac{\Gamma \vdash y{:}\sharp(T) \quad \Gamma \vdash z{:}T}{\Gamma;\emptyset;\emptyset \vdash \overline{y}\langle z\rangle} \quad \text{and} \quad \frac{\Gamma \vdash y{:}\natural(T) \quad \Gamma \vdash z{:}T}{\Gamma;\emptyset;\emptyset \vdash \overline{y}\langle z\rangle}.$$

T-Out-$\mathrm{M_M}$ covers the case that the link is typed by a m-sort. Here, the first subgoal of T-$\mathrm{Out_B}$ is missing. To be permitted to send the respective value, Ψ has to assign the corresponding link type that is a part of the respective m-sort. Note that in both cases $\Delta = \emptyset$ and Ψ contains at most one element that is consumed by T-Out-$\mathrm{M_M}$.

The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ uses polyadic synchronisation on links combined from two channel names in the translation of the parallel operator. Over these links three values are swapped in the variant of the encoding with polyadic communication. Hence, links for polyadic synchronisation are used only to transmit a value that is typed by a m-sort like type. Accordingly, we use a variant of T-Out-$\mathrm{B_M}$ to capture outputs on links of the form $y \cdot o$. The result is the Rule T-$\mathrm{OutPS_M}$. As in T-$\mathrm{OutPS_B}$, y has to be typed by $\mathfrak{v}_\mathfrak{n}$ and the information on the link type is carried by the tag o.

T-$\mathrm{In_B}$ is split into three cases. The first rule, denoted by T-In-$\mathrm{B_M}$, covers inputs where neither the link nor the value are typed by a m-sort. Because such inputs appear in the translation of polyadic inputs such that the respective continuation may output on a link typed by a m-sort, T-In-$\mathrm{B_M}$ allows that Ψ is not empty. But again $\Delta = \emptyset$ is

required and it is not allowed to change Ψ. T-In-B_M does not distinguish between link types $\sharp(\cdot)$ and link types $\natural(\cdot)$ of auxiliary links. Apart from that, T-In-B_M is similar to T-In_B. The translation of a polyadic input starts with an input whose value is typed by a m-sort. This case is covered by T-In-$M1_M$. Since this point marks the starting point of a translated polyadic communication, the rule requires that no such protocol is still active, i.e., $\Delta = \Psi = \emptyset$. The only difference with T-In_B is that T-In-$M1_M$ initialises Ψ and, thus, enables the transmission of the first value over the link typed by the m-sort. T-In-$M2_M$ is used to cover inputs within the translation of a polyadic sender. Accordingly, Δ contains the active part of the m-sort for the link. Its first part has to be a link type corresponding to the received value, which replaces the first subgoal in T-In_B. For the remaining subgoal the first part of the actual state of the m-sort like type is cut off. Intuitively, it is consumed by this rule. For both, the precondition as well as the continuation, $\Psi = \emptyset$.

The Rule T-$InPS_M$ is similar to T-In-$M1_M$ but covers polyadic synchronisation of links combined from two channel names. The Rules T-Rep-B_M and T-Rep-M_M are similar to T-In-B_M and T-In-$M1_M$. The set of all typing rules of the monadic type system is given in Figure 6.8.

We observe that in the translation of a polyadic sender the actual state of a m-sort is expressed by cutting off the already consumed parts in T-Res-M_M and T-In-$M2_M$. However, also T-Out-M_M requires that the respective m-sort is already cut down to its actual state but this cutting is done by no rule described so far. In fact, it is performed by T-Par_M, the counterpart of T-Par_B, covering parallel compositions. Outputs on links typed by a m-sort are necessary to translate polyadic inputs and polyadic replicated inputs, as for the translation of $c_{r2}(r_o, r_i).P$ into

$$c_{r2}(u_{\sim,c}).\left((\nu u_o : \natural(\mathfrak{o}))\left(\overline{u_{\sim,c}}\langle u_o\rangle \mid u_o(r_o).\left((\nu u_i : \natural(\mathfrak{i}))\left(\overline{u_{\sim,c}}\langle u_i\rangle \mid u_i(r_i).\widehat{P}\right)\right)\right)\right).$$

By T-In-$M1_M$, Ψ is initialised by the m-sort without the preceding $\circ$. Then T-Par_M distributes the remaining m-sort such that the first part is assigned to the output capability and the rest to the remaining term. This behaviour is captured by the precondition $\Psi_P \cdot \Psi_Q$. It ensures, that either $\Psi_P \cdot \Psi_Q = \Psi_P, \Psi_Q$, or Ψ_P and Ψ_Q distribute some m-sort like types in the elements $y{:}T$ of Ψ into the part necessary on the left hand side and the part necessary on the right hand side of the parallel composition.

Definition 6.2.27. Let Ψ_P and Ψ_Q be two sets that contain elements of the form $y{:}T$, where $y \in \mathcal{N}$ and T is part of a m-sort like monadic type. Then $\Psi_P \cdot \Psi_Q$ holds if either $\mathsf{n}(\Psi_P) \cap \mathsf{n}(\Psi_Q) = \emptyset$ and $\Psi_P \cdot \Psi_Q = \Psi_P, \Psi_Q$, or if $\mathsf{n}(\Psi_P) \cap \mathsf{n}(\Psi_Q) \neq \emptyset$ and

$$\begin{aligned}\Psi_P \cdot \Psi_Q = &\{\, y_P{:}T_P \mid y_P \in (\mathsf{n}(\Psi_P) \setminus \mathsf{n}(\Psi_Q)) \wedge y_P{:}T_P \in \Psi_P \,\} \\ &\cup \{\, y_Q{:}T_Q \mid y_Q \in (\mathsf{n}(\Psi_Q) \setminus \mathsf{n}(\Psi_P)) \wedge y_Q{:}T_Q \in \Psi_Q \,\} \\ &\cup \{\, y{:}T_1 \triangleright \ldots \triangleright T_n \triangleright \bullet \mid y \in (\mathsf{n}(\Psi_P) \cap \mathsf{n}(\Psi_Q)) \,\}\end{aligned}$$

such that, for all $y \in (\mathsf{n}(\Psi_P) \cap \mathsf{n}(\Psi_Q))$ with $y{:}T_1 \triangleright \ldots \triangleright T_n \triangleright \bullet \in \Psi_P \cdot \Psi_Q$, there exists some $i_1, \ldots, i_m$ and $j_1, \ldots, j_o$ such that

$$\text{T-Name}_\text{B}\ \frac{}{\Gamma, x{:}T \vdash x{:}T} \qquad \text{T-Nil}_\text{M}\ \frac{}{\Gamma;\emptyset;\emptyset \vdash \mathsf{0}} \qquad \text{T-Succ}_\text{M}\ \frac{}{\Gamma;\emptyset;\emptyset \vdash \checkmark}$$

$$\text{T-Res-B}_\text{M}\ \frac{\Gamma, x{:}T;\Delta;\Psi \vdash P}{\Gamma;\Delta;\Psi \vdash (\nu x{:}T)\,P}\quad T \neq \circ \triangleright T'$$

$$\text{T-Res-M}_\text{M}\ \frac{\Gamma, x{:}T \triangleright T';\Delta,\Delta';\Psi \vdash P}{\Gamma;\Delta;\Psi \vdash (\nu x{:}T \triangleright T')\,P}\quad \Delta' = \begin{cases} x{:}T', & \text{if } T' \neq \bullet \\ \emptyset, & \text{else} \end{cases}$$

$$\text{T-Par}_\text{M}\ \frac{\Gamma;\Delta_P;\Psi_P \vdash P \quad \Gamma;\Delta_Q;\Psi_Q \vdash Q}{\Gamma;\Delta_P,\Delta_Q;\Psi_P \cdot \Psi_Q \vdash P \mid Q}$$

$$\text{T-Mat}_\text{M}\ \frac{\Gamma \vdash a{:}T \quad \Gamma \vdash b{:}T \quad \Gamma;\Delta;\Psi \vdash P}{\Gamma;\Delta;\Psi \vdash [a = b]\,P} \qquad \text{T-Tau}_\text{M}\ \frac{\Gamma;\emptyset;\emptyset \vdash P}{\Gamma;\emptyset;\emptyset \vdash \tau.P}$$

$$\text{T-Out-B}_\text{M}\ \frac{\Gamma \vdash y{:}\sharp(T) \quad \Gamma \vdash z{:}T}{\Gamma;\emptyset;\emptyset \vdash \overline{y}\langle z\rangle} \qquad \text{T-Out-M}_\text{M}\ \frac{\Gamma \vdash z{:}T}{\Gamma;\emptyset;y{:}\sharp(T) \vdash \overline{y}\langle z\rangle}$$

$$\text{T-OutPS}_\text{M}\ \frac{\Gamma \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma \vdash o{:}\sharp(\circ \triangleright T) \quad \Gamma \vdash z{:}\circ \triangleright T}{\Gamma;\emptyset;\emptyset \vdash \overline{y \cdot o}\langle z\rangle}$$

$$\text{T-In-B}_\text{M}\ \frac{\Gamma \vdash y{:}\sharp(T) \quad \Gamma, x{:}T;\emptyset;\Psi \vdash P}{\Gamma;\emptyset;\Psi \vdash y(x)\,.P}\quad T \neq \circ \triangleright T'$$

$$\text{T-In-M1}_\text{M}\ \frac{\Gamma \vdash y{:}\sharp(\circ \triangleright T) \quad \Gamma, x{:}\circ \triangleright T;\emptyset;x{:}T \vdash P}{\Gamma;\emptyset;\emptyset \vdash y(x)\,.P}$$

$$\text{T-In-M2}_\text{M}\ \frac{\Gamma, x{:}T;\Delta;\emptyset \vdash P}{\Gamma;y{:}\sharp(T) \triangleright T';\emptyset \vdash y(x)\,.P}\quad \Delta = \begin{cases} \emptyset, & \text{if } T' = \bullet \\ y{:}T', & \text{else} \end{cases}$$

$$\text{T-InPS}_\text{M}\ \frac{\Gamma \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma \vdash o{:}\sharp(\circ \triangleright T) \quad \Gamma, x{:}\circ \triangleright T;\emptyset;x{:}T \vdash P}{\Gamma;\emptyset;\emptyset \vdash y \cdot o(x)\,.P}$$

$$\text{T-Rep-B}_\text{M}\ \frac{\Gamma \vdash y{:}\sharp(T) \quad \Gamma, x{:}T;\emptyset;\emptyset \vdash P}{\Gamma;\emptyset;\emptyset \vdash y^\star(x)\,.P}\quad T \neq \circ \triangleright T'$$

$$\text{T-Rep-M}_\text{M}\ \frac{\Gamma \vdash y{:}\sharp(\circ \triangleright T) \quad \Gamma, x{:}\circ \triangleright T;\emptyset;x{:}T \vdash P}{\Gamma;\emptyset;\emptyset \vdash y^\star(x)\,.P}$$

Figure 6.8.: Typing Rules of the Monadic Type System.

1. $m > 0$, $o > 0$, $\{ i_1, \ldots, i_m, j_1, \ldots, j_o \} = \{ 1, \ldots, n \}$, and $n = m + o$,
2. for all $1 \leq k < m$ and all $1 \leq l < o$, we have $i_k < i_{k+1}$ and $j_l < j_{l+1}$,
3. if $m > 1$ then $y{:}T_{i_1} \triangleright \ldots \triangleright T_{i_m} \triangleright \bullet \in \Psi_P$ and else $y{:}T_{i_1} \in \Psi_P$, and
4. if $o > 1$ then $y{:}T_{j_1} \triangleright \ldots \triangleright T_{j_o} \triangleright \bullet \in \Psi_Q$ and else $y{:}T_{j_1} \in \Psi_Q$.

Note that the definition above allows also to distribute a part of a m-sort like type by cutting it into its atomic pieces and distribute this pieces arbitrarily among both sides of a parallel operator as long as the order of pieces for each side is not changed. This is necessary to capture reordering of terms modulo structural congruence if there are more than two unguarded subterms containing an output that transmits a value typed by a part of the respective m-sort like type. However, by Definition 5.4.1, this does not happen in the considered encodings. Also note that, in the case of remainders containing only a single atomic piece, the terminating $\bullet$ is omitted. Except from that Δ_P, Δ_Q and $\Psi_P \cdot \Psi_Q$ ensure that for each name typed by a m-sort like type the information about the actual state of that type can only be used in one part of the parallel composition. This shows that links typed by m-sort like types are linearised (see Lemma 6.2.58), i.e., in each term there is not more than a single unguarded out- or input on that link.

The monadic type system consists of the monadic types and the typing rule in Figure 6.8. Remember that, in contrast to the basic type system, the monadic type system uses the monadic type judgements of Definition 6.2.26.

Definition 6.2.28 (Monadic Type System)**.** The *monadic type system* is given by the monadic types in Definition 6.2.23 and the typing rules in Figure 6.8.

Because the monadic type system inherits the typing rule for type assignments of the basic type systems, the validity of Lemma 6.2.9 is not affected. The same holds for strengthening and weakening of type judgements for type assignments. We show that also the remaining properties of the basic type system are preserved by the extension of the basic to the monadic type system.

Lemma 6.2.29 (Strengthening)**.** *If* $\Gamma, x{:}T; \Delta; \Psi \vdash P$ *and* $x \notin \mathsf{fn}(P)$ *then* $\Gamma; \Delta; \Psi \vdash P$*.*

Lemma 6.2.30 (Weakening)**.** *If* $\Gamma; \Delta; \Psi \vdash P$ *then* $\Gamma, x : T; \Delta; \Psi \vdash P$ *for any monadic type* T *and any name* x *such that* $\Gamma(x)$ *is not defined or equal to* T*.*

The proof of strengthening and weakening for the monadic type system are similar to the respective proofs in the basic type system (Lemma 6.2.10 and Lemma 6.2.11). To demonstrate the influence of the additional sets in the type judgements, we present the respective inductions in the appendix in Section A.1.2. Again, well-typedness of a typed term is preserved modulo structural congruence with respect to closed type environments.

Lemma 6.2.31. *If* $\Gamma; \Delta; \Psi \vdash P$*,* $P \equiv Q$*, and* Γ *is closed for* Q *then* $\Gamma; \Delta; \Psi \vdash Q$*.*

The proof is (similar to the linear type system) by a case differentiation on the rules of structural congruence and an induction on the number of structural congruence rules applied to obtain $P \equiv Q$. We postpone it to Section A.1.2. Remarkably, Δ and Ψ contain only information about free names of the term.

Lemma 6.2.32. *If* $\Gamma; \Delta; \Psi \vdash P$ *then* $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Proof. We perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{ \mathcal{P}_\mathrm{a}, \mathcal{P}_\mathrm{p}, \mathcal{P}_\mathrm{a}^= \}$ be the set of processes of the target language of the considered encoding.

Base Case: If $\Gamma; \Delta; \Psi \vdash P$ can be derived from one of the axioms then $\Delta = \Psi = \emptyset$.

Induction Hypothesis: $\Gamma; \Delta; \Psi \vdash P$ implies $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$

Induction Step: We perform a case split on the inference rules in Figure 6.8.

Case of T-Res-B$_\mathrm{M}$ **:** In this case, $P = (\nu x{:}T)\, P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{M}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, and $\Gamma, x{:}T; \Delta; \Psi \vdash P'$. By the induction hypothesis, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P')$. Moreover, by Definition 6.2.26, $x \notin \mathsf{n}(\Gamma)$ and, thus, $x \notin \mathsf{n}(\Delta) \cup \mathsf{n}(\Psi)$. Then, since $\mathsf{fn}(P) = \mathsf{fn}(P') \setminus \{ x \}$, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Case of T-Res-M$_\mathrm{M}$ **:** Again $P = (\nu x{:}T)\, P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{M}$, and $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$ but here $\Gamma, x : T; \Delta, \Delta'; \Psi \vdash P'$, where Δ' is either $x : T$ or $\emptyset$. By the induction hypothesis, $\mathsf{n}(\Delta, \Delta') \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P')$. Moreover, by Definition 6.2.26, $x \notin \mathsf{n}(\Gamma)$ and, thus, $x \notin \mathsf{n}(\Delta) \cup \mathsf{n}(\Psi)$. Then, since $\mathsf{fn}(P) = \mathsf{fn}(P') \setminus \{ x \}$, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Case of T-Par$_\mathrm{M}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, $\Delta = \Delta_1, \Delta_2$, $\Psi = \Psi_1 \cdot \Psi_2$, $\Gamma; \Delta_1; \Psi_1 \vdash P_1$, and $\Gamma; \Delta_2; \Psi_2 \vdash P_2$. Thus, by the induction hypothesis, $\mathsf{n}(\Delta_1) \cup \mathsf{n}(\Psi_1) \subseteq \mathsf{fn}(P_1)$ and $\mathsf{n}(\Delta_2) \cup \mathsf{n}(\Psi_2) \subseteq \mathsf{fn}(P_2)$. By Definition 6.2.27, $\mathsf{n}(\Psi) = \mathsf{n}(\Psi_1) \cup \mathsf{n}(\Psi_2)$. Since $\mathsf{fn}(P) = \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Case of T-Mat$_\mathrm{M}$ **:** In this case, $P = [a = b]\, P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, and $\Gamma; \Delta; \Psi \vdash P'$. By the induction hypothesis, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P')$. Then, since $\mathsf{fn}(P) = \mathsf{fn}(P')$, $\mathsf{n}(\Delta) \cup \mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Case of T-Tau$_\mathrm{M}$, T-Out-B$_\mathrm{M}$, T-OutPS$_\mathrm{M}$, T-In-M1$_\mathrm{M}$, T-InPS$_\mathrm{M}$, T-Rep-B$_\mathrm{M}$, **or** T-Rep-M$_\mathrm{M}$ **:** Here, $\Delta = \Psi = \emptyset$.

Case of T-Out-M$_\mathrm{M}$ **:** In this case, $\Delta = \emptyset$, $\mathsf{n}(\Psi) = \{ y \}$ and $y \in \mathsf{fn}(P)$.

Case of T-In-B$_\mathrm{M}$ **:** In this case, $P = y(x).P'$ for some $x, y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, $\Delta = \emptyset$, and $\Gamma, x{:}T; \Delta; \Psi \vdash P'$. By the induction hypothesis, $\mathsf{n}(\Psi) \subseteq \mathsf{fn}(P')$. Moreover, by Definition 6.2.26, $x \notin \mathsf{n}(\Gamma)$ and, thus, $x \notin \mathsf{n}(\Psi)$. Then, since $\mathsf{fn}(P) = \mathsf{fn}(P') \setminus \{ x \}$, $\mathsf{n}(\Psi) \subseteq \mathsf{fn}(P)$.

Case of T-In-M2$_\mathrm{M}$ **:** In this case, $\mathsf{n}(\Delta) = \{ y \}$, $\Psi = \emptyset$, and $y \in \mathsf{fn}(P)$.

□

As for type judgements in the basic type system, a monadic type judgement can be proved only if for all free names a type is provided by the type environment.

Lemma 6.2.33. *If* $\Gamma;\Delta;\Psi \vdash P$ *then* Γ *is closed for* P*.*

Proof. Assume the contrary, i.e., assume $\Gamma;\Delta;\Psi \vdash P$ and $x \notin \mathsf{n}(\Gamma)$ but $x \in \mathsf{fn}(P)$. Hence, P has a (potentially guarded) subterm which is either of the form

1. $\overline{y}\langle z\rangle$ or $\overline{y \cdot o}\langle z\rangle$ for some names $y, o, z \in \mathcal{N}$,
2. $y(x').P'$, or $y \cdot o(x').P'$, or $y^\star(x').P'$ for some $o, x', y \in \mathcal{N}$ and some P', or
3. $[a = b]\,P'$ for some $a, b \in \mathcal{N}$ and some P'

such that $x \in \{ a, b, o, y, z \}$. None of the typing rules in Figure 6.8 allows to derive a type judgement for a term without requiring a type judgement for all its subterms. Thus, in the derivation of $\Gamma;\Delta;\Psi \vdash P$ there is some subgoal $\Gamma';\Delta';\Psi' \vdash \overline{y}\langle z\rangle$, $\Gamma';\Delta';\Psi' \vdash \overline{y \cdot o}\langle z\rangle$, $\Gamma';\Delta';\Psi' \vdash y(x').P'$, $\Gamma';\Delta';\Psi' \vdash y \cdot o(x').P'$, or $\Gamma';\Delta';\Psi' \vdash y^\star(x').P'$ for some Γ' that is obtained from Γ during the derivation up to the mentioned subgoal. Moreover, the typing rules in Figure 6.8 can add type assignments to Γ during derivations but do so only for bound names. We conclude that, $x \notin \mathsf{n}(\Gamma')$. By Definition 6.2.26, then also $x \notin \mathsf{n}(\Delta') \cup \mathsf{n}(\Psi')$.

Consider the case that $x = y$. Then we can apply only T-Out-B$_\mathrm{M}$ on the subgoal $\Gamma';\Delta';\Psi' \vdash \overline{y}\langle z\rangle$, because T-Out-M$_\mathrm{M}$ requires $y \in \mathsf{n}(\Psi')$. As result we obtain the subgoal $\Gamma' \vdash y : \sharp(T)$ for some type $T \in \mathbb{T}_\mathrm{M}$. The Rule T-OutPS$_\mathrm{M}$ on the subgoal $\Gamma';\Delta';\Psi' \vdash \overline{y \cdot o}\langle z\rangle$ leads to $\Gamma' \vdash y : \mathfrak{v}_\mathfrak{n}$. If $\Gamma';\Delta';\Psi' \vdash y(x').P'$ we can apply T-In-B$_\mathrm{M}$ or T-In-M1$_\mathrm{M}$. Both lead again to $\Gamma' \vdash y : \sharp(T)$ for some type $T \in \mathbb{T}_\mathrm{M}$. T-InPS$_\mathrm{M}$ on $\Gamma';\Delta';\Psi' \vdash y \cdot o(x').P'$ leads to $\Gamma' \vdash y : \mathfrak{v}_\mathfrak{n}$. On $\Gamma';\Delta';\Psi' \vdash y^\star(x').P$ we can apply T-Rep-B$_\mathrm{M}$ or T-Rep-M$_\mathrm{M}$. Both result in the subgoal $\Gamma' \vdash y : \sharp(T)$. Hence, for all cases we have to show either $\Gamma' \vdash y : \sharp(T)$ for some type $T \in \mathbb{T}_\mathrm{M}$ or $\Gamma' \vdash y : \mathfrak{v}_\mathfrak{n}$. But, since $y \notin \mathsf{n}(\Gamma')$ and because of Lemma 6.2.9, neither $\Gamma' \vdash y : \sharp(T)$ nor $\Gamma' \vdash y : \mathfrak{v}_\mathfrak{n}$ can hold. Hence, the judgement $\Gamma;\Delta;\Psi \vdash P$ cannot be derived, which contradicts the assumption.

If $x = o$, applying the Rule T-OutPS$_\mathrm{B}$ on the subgoal $\Gamma';\Delta';\Psi' \vdash \overline{y \cdot o}\langle z\rangle$ or T-InPS$_\mathrm{B}$ on the subgoal $\Gamma';\Delta';\Psi' \vdash y \cdot o(x').P$ results in the subgoal $\Gamma' \vdash o : \sharp(T)$ for some type $T \in \mathbb{T}_\mathrm{M}$. Again, by $o \notin \mathsf{n}(\Gamma')$ and Lemma 6.2.9, $\Gamma' \vdash o : \sharp(T)$ has to fail. Hence, the judgement $\Gamma;\Delta;\Psi \vdash P$ cannot be derived, which contradicts the assumption.

If $x = z$ then P contains an output $\overline{y}\langle z\rangle$ or $\overline{y \cdot o}\langle z\rangle$. Applying Rule T-Out-B$_\mathrm{M}$, Rule T-Out-M$_\mathrm{M}$, or Rule T-OutPS$_\mathrm{M}$ on the corresponding subgoal results in the subgoal $\Gamma' \vdash z : T$ for some $T \in \mathbb{T}_\mathrm{B}$. Since $x \notin \mathsf{n}(\Gamma')$ and because of Lemma 6.2.9, this subgoal has to fail. Hence, the judgement $\Gamma;\Delta;\Psi \vdash P$ cannot be derived, which contradicts again the assumption.

If $x \in \{ a, b \}$ then P contains a subterm $[a = b]\,P'$. Applying T-Mat$_\mathrm{M}$ on $\Gamma';\Delta';\Psi' \vdash [a = b]\,P'$ results in the subgoals $\Gamma' \vdash a : T$ and $\Gamma' \vdash b : T$ for some type $T \in \mathbb{T}_\mathrm{M}$. But since $x \notin \mathsf{n}(\Gamma')$, $x \in \{ a, b \}$, and because of Lemma 6.2.9, one of these subgoals has to fail. Hence, the judgement $\Gamma;\Delta;\Psi \vdash P$ cannot be derived, which contradicts the assumption. □

Monadic type judgements are robust under substitution if the substitution preserves the type of the substituted name and the substitution is also applied on the auxiliary sets Δ and Ψ. We define such substitutions on auxiliary sets M as $\{\, z/x \,\}\, M = \{\, y{:}T \mid y{:}T \in M \wedge y' = \{\, z/x \,\}\,(y) \,\}$.

Lemma 6.2.34. *Assume* $\Gamma(x) = \Gamma(z)$. *Then* $\Gamma; \Delta; \Psi \vdash P$ *implies* $\Gamma; \{\, z/x \,\}\, \Delta; \{\, z/x \,\}\, \Psi \vdash \{\, z/x \,\}\, P$.

Proof. We construct the derivation of $\Gamma; \{\, z/x \,\}\, \Delta; \{\, z/x \,\}\, \Psi \vdash \{\, z/x \,\}\, P$ from the derivation of $\Gamma; \Delta; \Psi \vdash P$, by showing that both proof trees have the same structure, i.e., apply the same typing rules in the same order. Note that $\{\, z/x \,\}\, \Delta$ and $\{\, z/x \,\}\, \Psi$ ensure that the preconditions of the Rules T-Par$_\mathrm{M}$, T-Out-M$_\mathrm{M}$, and T-In-M2$_\mathrm{M}$ are satisfied for the derivation of $\Gamma; \{\, z/x \,\}\, \Delta; \{\, z/x \,\}\, \Psi \vdash \{\, z/x \,\}\, P$ if they are satisfied for $\Gamma; \Delta; \Psi \vdash P$. Moreover, Lemma 6.2.32 ensures that the substitution has only an effect on the auxiliary sets if it has an effect on P, i.e., if $x \in \mathsf{fn}(P)$. Analysing the rules in Figure 6.8, we observe that the Rules T-Nil$_\mathrm{M}$, T-Succ$_\mathrm{M}$, T-Par$_\mathrm{M}$, and T-Tau$_\mathrm{M}$ do not consider specific names of terms, i.e., can be applied in exactly the same way in both proof trees. The same holds for the third subgoal of T-Mat$_\mathrm{M}$. Moreover, we observe that the typing rules may add type assignments to the environments of their subgoals but do not remove any, i.e., for all subgoals with a type environment Γ' in both derivations we have $\Gamma'(x) = \Gamma'(z)$.

P and $\{\, z/x \,\}\, P$ differ only if x is free in P. So, T-Res-B$_\mathrm{M}$ and T-Res-M$_\mathrm{M}$ can also be applied in exactly the same way in both proof trees. For the same reason, the respective second subgoals of the Rules T-In-B$_\mathrm{M}$, T-In-M1$_\mathrm{M}$, T-Rep-B$_\mathrm{M}$, and T-Rep-M$_\mathrm{M}$ as well as the last subgoal of T-InPS$_\mathrm{M}$ and the only subgoal of T-In-M2$_\mathrm{M}$ are unaffected.

All remaining subgoals are of the form $\Gamma'' \vdash y{:}T'$. By Lemma 6.2.9 and $\Gamma(x) = \Gamma(z)$, $\Gamma \vdash x : T$ iff $\Gamma \vdash z : T$. By Lemma 6.2.11, this condition remains valid if during the derivation new assignments are added to Γ, i.e., $\Gamma \vdash x : T$ iff $\Gamma \vdash z : T$ implies that $\Gamma, \Gamma' \vdash x : T$ iff $\Gamma, \Gamma' \vdash z : T$ for all Γ'. Hence, for all leaves proven by T-Name$_\mathrm{B}$ in the proof tree of $\Gamma; \Delta; \Psi \vdash P$ we can prove the corresponding leaf in the proof tree of $\Gamma; \{\, z/x \,\}\, \Delta; \{\, z/x \,\}\, \Psi \vdash \{\, z/x \,\}\, P$ again by T-Name$_\mathrm{B}$. □

Of course, the most important property is again subject reduction. Note that in contrast to type environments the typing rules can remove elements from the sets Δ and Ψ and do also manipulate elements of these sets. Moreover, we have to require that if the current state is of a link typed by a m-sort like type is captured by both sets, Δ and Ψ, then these two sets have to describe the same state of the link. In this case we say Δ and Ψ are consistent.

Definition 6.2.35 (Consistent Auxiliary Sets)**.** Let Δ and Ψ be two sets containing elements of the form $y{:}T$, where $y \in \mathcal{N}$ and T is a part of a monadic type. Then Δ and Ψ are *consistent* if, for all $y \in \mathsf{n}(\Delta) \cap \mathsf{n}(\Psi)$ such that $y{:}T_1 \in \Delta$ and $y{:}T_2 \in \Psi$, either $T_1 = T_2 \triangleright T'$ or the first part of T_1 and the first part of T_2 are equal.

A reduction of an input may consume or change an element of Δ if this input was typed with respect to the Rule T-In-M2$_\mathrm{M}$. Moreover, a reduction of a (replicated)

input may add an element to Ψ if this (replicated) input was typed with respect to the Rule T-In-M1$_{\mathrm{M}}$ or T-Rep-M$_{\mathrm{M}}$. Accordingly, a reduction of an output may consume or change an element of Ψ if this output was typed with respect to the Rule T-Out-M$_{\mathrm{M}}$. If Δ' and Ψ' are obtained from Δ and Ψ by such a reduction, we call them derivatives of Δ, Ψ, and the corresponding Γ.

Definition 6.2.36 (Derivatives of Auxiliary Sets)**.** Let $\Gamma;\Delta;\Psi \vdash P$. Then Δ' and Ψ' are derivatives of Γ, Δ, and Ψ if

1. $\Delta' = \Delta$ or there exists some $y_1 : T_1 \in \Delta$ such that $\Delta' = (\Delta \setminus \{\, y_1{:}T_1 \,\}) \cup \{\, y_1{:}T_1'' \mid T_1 = T_1' \triangleright T_1'' \wedge T_1'' \neq \bullet \,\}$, and
2. $\Psi' = \Psi$, or there exists some $y_3 : T_3 \in \Psi$ such that $\Psi' = (\Psi \setminus \{\, y_3{:}T_3 \,\}) \cup \{\, y_3{:}T_3'' \mid T_3 = T_4 \triangleright T_4' \triangleright T_4'' \wedge ((T_4'' \neq \bullet \wedge T_3'' = T_4' \triangleright T_4'') \vee (T_4'' = \bullet \wedge T_3'' = T_4')) \,\}$, or $\Psi' = \Psi, y_2{:}T_2$ for some $y_2 \notin \mathsf{n}(\Psi)$ and $\Gamma(y_2) = \circ \triangleright T_2$.

Note that consistency is preserved by subject reduction. Moreover, by Lemma 6.2.39, for every translated typed term $\widehat{P}$ such that P is well-typed in the basic type system with respect to Γ, we can derive $\Gamma;\emptyset;\emptyset \vdash \widehat{P}$. So, the Δ and Ψ obtained during derivations of translated typed terms are always consistent.

Lemma 6.2.37 (Subject Reduction)**.** *If* $\Gamma;\Delta;\Psi \vdash P$, $P \longmapsto P'$, Γ *is closed for* P', *and* Δ *and* Ψ *are consistent then* $\Gamma;\Delta';\Psi' \vdash P'$ *such that* Δ' *and* Ψ' *are consistent derivatives of* Γ, Δ, *and* Ψ.

Proof. We perform an induction on the depth of the derivation of $P \longmapsto P'$. Let $\mathcal{P} \in \{\, \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \,\}$ be the set of processes of the target language of the considered encoding.

Base Case: The reduction semantics of π_{a}, π_{p}, and $\pi_{\mathrm{a}}^{=}$ in Figure 2.3 contains the Axioms Pi-Tau$_{\mathrm{a,p}}$, Pi-Com$_{\mathrm{a,p}}$, Pi-ComPS$_{\mathrm{p}}$, and Pi-Rep$_{\mathrm{a,p}}$. The first rule requires that $P = \tau.Q$ and $P' = Q$ for some $Q \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$. Hence, $\Gamma;\Delta;\Psi \vdash P'$ follows from $\Gamma;\Delta;\Psi \vdash P$ and T-Tau$_{\mathrm{M}}$.

The Rule Pi-Com$_{\mathrm{a,p}}$ requires that $P = y(x).Q \mid \overline{y}\langle z\rangle$ and $P' = \{\, {}^{z}/_{x} \,\}\, Q$. In this case, the derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with

$$\frac{D_1 \quad D_2}{\Gamma;\Delta;\Psi \vdash y(x).Q \mid \overline{y}\langle z\rangle}\,\text{T-Par}_{\mathrm{M}}$$

where $\Gamma;\Delta_1;\Psi_1 \vdash y(x).Q$ is the goal of D_1 and $\Gamma;\Delta_2;\Psi_2 \vdash \overline{y}\langle z\rangle$ is the goal of D_2 such that $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$. On the left hand side we have to apply next one of the input Rules T-In-B$_{\mathrm{M}}$, T-In-M1$_{\mathrm{M}}$, or T-In-M2$_{\mathrm{M}}$ and on the right hand side one of the Rules T-Out-B$_{\mathrm{M}}$ or T-Out-M$_{\mathrm{M}}$.

Case of T-In-B$_{\mathrm{M}}$ **:** If D_1 is shown by T-In-B$_{\mathrm{M}}$, we have

$$D_1 = \frac{\dfrac{}{\Gamma \vdash y{:}\sharp(T)}\,\text{T-Name}_{\mathrm{B}} \quad \dfrac{\cdots}{\Gamma, x{:}T;\Delta_1;\Psi_1 \vdash Q}\,\cdots}{\Gamma;\Delta_1;\Psi_1 \vdash y(x).Q}\,\text{T-In-B}_{\mathrm{M}}$$

for some $T \in \mathbb{T}_{\mathrm{M}}$ that does not contain $\triangleright$. By Lemma 6.2.9, $\Gamma \vdash y : \sharp(T)$ implies $\Gamma(y) = \sharp(T)$. Because of that and by Definition 6.2.26, y does not occur in Δ or Ψ. Since $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$ and by Definition 6.2.27, this implies that y does also not occur in Δ_1, Δ_2, Ψ_1, or Ψ_2. Thus, we cannot apply T-Out-M$_{\mathrm{M}}$ for D_2. Hence,

$$D_2 = \dfrac{\dfrac{}{\Gamma \vdash y : \sharp(T')}\text{T-Name}_{\mathrm{B}} \quad \dfrac{}{\Gamma \vdash z : T'}\text{T-Name}_{\mathrm{B}}}{\Gamma; \Delta_2; \Psi_2 \vdash \overline{y}\langle z \rangle}\text{T-Out-B}_{\mathrm{M}}$$

where $\Delta_2 = \Psi_2 = \emptyset$. By Definition 6.2.27, then $\Delta = \Delta_1$ and $\Psi = \Psi_1$. By Lemma 6.2.9, $T = T'$. Moreover, because of $\Gamma \vdash z : T'$ and again Lemma 6.2.9, we know that $\Gamma(z) = T$. With Lemma 6.2.34 and $\Gamma, x : T; \Delta_1; \Psi_1 \vdash Q$ we conclude $\Gamma, x : T; \Delta; \Psi \vdash P'$. Finally, by Lemma 6.2.10, we have $\Gamma; \Delta; \Psi \vdash P'$, because $x \notin \mathsf{fn}(P')$.

Case of T-In-M1$_{\mathrm{M}}$ **:** If D_1 is shown by T-In-M1$_{\mathrm{M}}$, we have $\Delta_1 = \Psi_1 = \emptyset$ and

$$D_1 = \dfrac{\dfrac{}{\Gamma \vdash y : \sharp(\circ \triangleright T)}\text{T-Name}_{\mathrm{B}} \quad \dfrac{\cdots}{\Gamma, x : \circ \triangleright T; \emptyset; x : T \vdash Q}\cdots}{\Gamma; \emptyset; \emptyset \vdash y(x).Q}\text{T-In-M1}_{\mathrm{M}}$$

By Lemma 6.2.9, $\Gamma \vdash y : \sharp(\circ \triangleright T)$ implies $\Gamma(y) = \sharp(\circ \triangleright T)$. Because of that and by Definition 6.2.26, y does not occur in $\Delta = \Delta_2$ or $\Psi = \Psi_2$. Thus, we cannot apply T-Out-M$_{\mathrm{M}}$ for D_2. Hence,

$$D_2 = \dfrac{\dfrac{}{\Gamma \vdash y : \sharp(T')}\text{T-Name}_{\mathrm{B}} \quad \dfrac{}{\Gamma \vdash z : T'}\text{T-Name}_{\mathrm{B}}}{\Gamma; \Delta_2; \Psi_2 \vdash \overline{y}\langle z \rangle}\text{T-Out-B}_{\mathrm{M}}$$

where $\Delta_2 = \Psi_2 = \emptyset$. By Definition 6.2.27, then $\Delta = \Psi = \emptyset$. By Lemma 6.2.9, $T' = \circ \triangleright T$. Moreover, because of $\Gamma \vdash z : T'$ and again Lemma 6.2.9, we know that $\Gamma(z) = \circ \triangleright T$. With Lemma 6.2.34 and $\Gamma, x : \circ \triangleright T; \emptyset; x : T \vdash Q$ we conclude $\Gamma, x : \circ \triangleright T; \emptyset; z : T \vdash P'$. Finally, by Lemma 6.2.10, we have $\Gamma; \emptyset; z : T \vdash P'$, because $x \notin \mathsf{fn}(P')$. Note that $\emptyset$ and $z : T$ are consistent and are derivatives of Γ, $\emptyset$, and $\emptyset$.

Case of T-In-M2$_{\mathrm{M}}$ **:** If D_1 is shown by T-In-M2$_{\mathrm{M}}$, we have $\Delta_1 = y : \sharp(T) \triangleright T'$, $\Psi_1 = \emptyset$, and

$$D_1 = \dfrac{\dfrac{\cdots}{\Gamma, x : T; \Delta_1'; \emptyset \vdash Q}\cdots}{\Gamma; y : \sharp(T) \triangleright T'; \emptyset \vdash y(x).Q}\text{T-In-M2}_{\mathrm{M}}$$

where Δ_1' is $\emptyset$ if $T' = \bullet$ and else $\Delta_1' = y : T'$. By Definition 6.2.26, $y \in \mathsf{n}(\Delta_1)$ implies $\Gamma(y) = \circ \triangleright T_y$ for some T_y. Thus, by Lemma 6.2.9, we cannot derive a judgement like $\Gamma \vdash y : \sharp(T'')$, i.e., cannot apply T-Out-B$_{\mathrm{M}}$ on D_2. Hence,

$$D_2 = \dfrac{\dfrac{}{\Gamma \vdash z : T''}\text{T-Name}_{\mathrm{B}}}{\Gamma; \Delta_2; \Psi_2 \vdash \overline{y}\langle z \rangle}\text{T-Out-M}_{\mathrm{M}}$$

where $\Delta_2 = \emptyset$ and $\Psi_2 = y : \sharp(T'')$. Thus, $\Delta = \Delta_1$ and $\Psi = \Psi_2$. Since Δ and Ψ are consistent and by Definition 6.2.35, $T'' = T$. Because of $\Gamma \vdash z : T''$ and Lemma 6.2.9, $\Gamma(z) = T$. With Lemma 6.2.34 and $\Gamma, x : T; \Delta_1'; \emptyset \vdash Q$ we conclude $\Gamma, x : T; \Delta_1'; \emptyset \vdash P'$. Finally, by Lemma 6.2.10, we have $\Gamma; \Delta_1'; \emptyset \vdash P'$, because $x \notin \mathsf{fn}(P')$. Note that Δ_1' and $\emptyset$ are consistent and are derivatives of Γ, Δ_1, and Ψ_2.

The other two cases are similar.

The Rule Pi-ComPS$_\mathrm{p}$ requires that $P = y \cdot o(x).Q \mid \overline{y \cdot o}\langle z \rangle$ and $P' = \{ {}^{z}/_{x} \} Q$. In this case, the derivation of $\Gamma; \Delta; \Psi \vdash P$ starts again with

$$\frac{D_1 \quad D_2}{\Gamma; \Delta; \Psi \vdash y \cdot o(x).Q \mid \overline{y \cdot o}\langle z \rangle} \text{T-Par}_\mathrm{M}$$

where $\Gamma; \Delta_1; \Psi_1 \vdash y \cdot o(x).Q$ is the goal of D_1 and $\Gamma; \Delta_2; \Psi_2 \vdash \overline{y \cdot o}\langle z \rangle$ is the goal of D_2 such that $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$. On the left hand side we have to apply next T-InPS$_\mathrm{M}$ and on the right hand side T-OutPS$_\mathrm{M}$. Hence, we have $\Delta_1 = \Psi_1 = \emptyset$ and

$$D_1 = \frac{\dfrac{}{\Gamma \vdash y : \mathfrak{v}_\mathfrak{n}} N \quad \dfrac{}{\Gamma \vdash o : \sharp(\circ \triangleright T)} N \quad \dfrac{\cdots}{\Gamma, x : \circ \triangleright T; \emptyset; x : T \vdash Q} \cdots}{\Gamma; \emptyset; \emptyset \vdash y \cdot o(x).Q} \text{T-InPS}_\mathrm{M}$$

where $N = \text{T-Name}_\mathrm{B}$. Moreover, $\Delta_2 = \Psi_2 = \emptyset$ and

$$D_2 = \frac{\dfrac{}{\Gamma \vdash y : \mathfrak{v}_\mathfrak{n}} N \quad \dfrac{}{\Gamma \vdash o : \sharp(T')} N \quad \dfrac{}{\Gamma \vdash z : T'} \text{T-Name}_\mathrm{B}}{\Gamma; \emptyset; \emptyset \vdash \overline{y}\langle z \rangle} \text{T-OutPS}_\mathrm{M}$$

where $N = \text{T-Name}_\mathrm{B}$. By Definition 6.2.27, then $\Delta = \Psi = \emptyset$. By Lemma 6.2.9, $T' = \circ \triangleright T$. Moreover, because of $\Gamma \vdash z : T'$ and again Lemma 6.2.9, we know that $\Gamma(z) = \circ \triangleright T$. With Lemma 6.2.34 and $\Gamma, x : \circ \triangleright T; \emptyset; x : T \vdash Q$ we conclude $\Gamma, x : \circ \triangleright T; \emptyset; z : T \vdash P'$. Finally, by Lemma 6.2.29, we have $\Gamma; \emptyset; z : T \vdash P'$, because $x \notin \mathsf{fn}(P')$. Again, $\emptyset$ and $z : T$ are consistent and are derivatives of Γ, $\emptyset$, and $\emptyset$.

The Rule Pi-Rep$_\mathrm{a,p}$ requires that $P = y^\star(x).Q \mid \overline{y}\langle z \rangle$ and $P' = \{ {}^{z}/_{x} \} Q \mid y^\star(x).Q$. The derivation of $\Gamma; \Delta; \Psi \vdash P$ starts with

$$\frac{D_1 \quad D_2}{\Gamma; \Delta; \Psi \vdash y^\star(x).Q \mid \overline{y}\langle z \rangle} \text{T-Par}_\mathrm{M}$$

where $\Gamma; \Delta_1; \Psi_1 \vdash y^\star(x).Q$ is the goal of D_1 and $\Gamma; \Delta_2; \Psi_2 \vdash \overline{y}\langle z \rangle$ is the goal of D_2 such that $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$. On the left hand side we have to apply next T-Rep-B$_\mathrm{M}$ or T-Rep-M$_\mathrm{M}$. In the first case the rest of the proof is similar to the Case of T-In-B$_\mathrm{M}$ for Pi-Com$_\mathrm{a,p}$ and in the other case the rest of the proof is similar to the Case of T-In-M1$_\mathrm{M}$ for Pi-Com$_\mathrm{a,p}$.

Induction Hypothesis: $\Gamma; \Delta; \Psi \vdash P$, $P \longmapsto P'$, Γ is closed for P', and Δ and Ψ are consistent imply $\Gamma; \Delta'; \Psi' \vdash P'$ such that Δ' and Ψ' are consistent derivatives of Γ, Δ, and Ψ, for all $P \in (\mathcal{P} : \mathbb{T}_\mathrm{M})$.

Induction Step: There are three cases; one for each of the reduction Rules Pi-Res$_{\mathrm{m,s,a,p}}$, Pi-Par$_{\mathrm{m,s,a,p}}$, and Pi-Cong$_{\mathrm{m,s,a,p}}$.

Case of Pi-Par$_{\mathrm{m,s,a,p}}$ **:** In this case, $P = P_1 \mid P_2$, $P_1 \longmapsto P_1'$, and $P' = P_1' \mid P_2$ for some $P_1, P_1', P_2 \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$. The derivation of $\Gamma; \Delta; \Psi \vdash P$ starts with

$$\dfrac{\dfrac{\cdots}{\Gamma; \Delta_1; \Psi_1 \vdash P_1} \cdots \quad \dfrac{\cdots}{\Gamma; \Delta_2; \Psi_2 \vdash P_2} \cdots}{\Gamma; \Delta; \Psi \vdash P_1 \mid P_2}\,\text{T-Par}_{\mathrm{M}}$$

where $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$. By the induction hypothesis, we have $\Gamma; \Delta_1'; \Psi_1' \vdash P_1'$ such that Δ_1' and Ψ_1' are consistent derivatives of Γ, Δ_1, and Ψ_1. Then, by Definition 6.2.35 and Definition 6.2.36, $\Delta' = \Delta_1', \Delta_2$ and $\Psi' = \Psi_1' \cdot \Psi_2$ are also consistent derivatives of Γ, Δ, and Ψ. Hence,

$$\dfrac{\dfrac{\cdots}{\Gamma; \Delta_1'; \Psi_1' \vdash P_1'}\,IH \quad \dfrac{}{\Gamma; \Delta_2; \Psi_2 \vdash P_2}\,A}{\Gamma; \Delta'; \Psi' \vdash P_1' \mid P_2}\,\text{T-Par}_{\mathrm{M}}$$

where IH is the induction hypothesis and A means by assumption, i.e., by the derivation above.

Case of Pi-Res$_{\mathrm{m,s,a,p}}$ **:** In this case, $P = (\nu x{:}T)\, P_1$, $P_1 \longmapsto P_2$, and $P' = (\nu x{:}T)\, P_2$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_{\mathrm{M}}$, and $P_1, P_2 \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$. The derivation of $\Gamma; \Delta; \Psi \vdash P$ starts with

$$\dfrac{\dfrac{\cdots}{\Gamma, x{:}T; \Delta_1; \Psi \vdash P_1} \cdots}{\Gamma; \Delta; \Psi \vdash (\nu x{:}T)\, P_1}\,R$$

where either $R = \text{T-Res-B}_{\mathrm{M}}$ and $\Delta_1 = \Delta$ or $R = \text{T-Res-M}_{\mathrm{M}}$, $T = T_1 \triangleright T_2$, and $\Delta_1 = \Delta, x : T_2$ or again $\Delta_1 = \Delta$. By the induction hypothesis, $\Gamma, x : T; \Delta_2; \Psi_2 \vdash P_2$ such that Δ_2 and Ψ_2 are consistent derivatives of $\Gamma, x{:}T$, Δ_1, and Ψ. Hence,

$$\dfrac{\dfrac{}{\Gamma, x{:}T; \Delta_2; \Psi_2 \vdash P_2}\,IH}{\Gamma; \Delta_2'; \Psi_2 \vdash (\nu x{:}T)\, P_2}\,R$$

if we find an appropriate Δ_2'. If $R = \text{T-Res-B}_{\mathrm{M}}$ then $\Delta_1 = \Delta$ and, so, $\Delta_2' = \Delta_2$. Else, if $R = \text{T-Res-M}_{\mathrm{M}}$, we have $T = T_1 \triangleright T_2$ and $\Delta_1 = \Delta, x{:}T_2$ or $\Delta_1 = \Delta$. By Definition 6.2.36, $\Delta_1 = \Delta$ implies $x \notin \mathsf{n}(\Delta_2)$. In this case $\Delta_2' = \Delta_2$ again. Else, by Definition 6.2.36, $\Delta_1 = \Delta, x{:}T_2$ implies $\Delta_2 = \Delta_1$ or $\Delta_2 = \Delta, x{:}T_2'$, where $T_2 = T_2'' \triangleright T_2'$. In both cases, we can choose $\Delta_2' = \Delta$.

Case of Pi-Cong$_{\mathrm{m,s,a,p}}$ **:** In the this case, $P \equiv Q$, $Q \longmapsto Q'$, and $Q' \equiv P'$ for some $Q, Q' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$. Without loss of generality let us assume that this is the only application of Pi-Cong$_{\mathrm{m,s,a,p}}$ in $P \longmapsto P'$. Let R, R' be such that $Q \equiv R$, $Q' \equiv R'$, and neither R nor R' contains unguarded subterms guarded by a match prefix $[a = a]$. Then, by Pi-Cong$_{\mathrm{m,s,a,p}}$, also $P \equiv R$, $R \longmapsto R'$, and

$R' \equiv P'$. This time, R and R' do not have a match that is not already in P or P', respectively. Moreover, by Lemma 6.2.33, $\Gamma; \Delta; \Psi \vdash P$ implies that Γ is closed for P, i.e., provides a type for each free name of P. By assumption, Γ is also closed for P'. Since the only rule of structural congruence that allows to introduce free names is the rule that introduces matches, Γ is also closed for R and R'. By Lemma 6.2.31, then $\Gamma; \Delta; \Psi \vdash P$ and $P \equiv R$ imply $\Gamma; \Delta; \Psi \vdash R$. By the induction hypothesis $\Gamma; \Delta; \Psi \vdash R$ and $R \longmapsto R'$ imply $\Gamma; \Delta'; \Psi' \vdash R'$. Finally, by Lemma 6.2.31, $\Gamma; \Delta'; \Psi' \vdash R'$ and $R' \equiv P'$ imply $\Gamma; \Delta'; \Psi' \vdash P'$.

□

Finally, we show that the translation of basic types in Definition 6.2.22 and of terms typed in the basic type system in Definition 6.2.24 preserves well-typedness. For this, we consider first preservation of type judgements on type assignments.

Lemma 6.2.38 (Preservation of Well-Typed Assignments)**.** $\Gamma \vdash x{:}T \quad$ *iff* $\quad \widehat{\Gamma} \vdash x{:}\widehat{T}$.

Proof. By Lemma 6.2.9, $\Gamma \vdash x : T$ iff $\Gamma(x) = T$. By Definition 6.2.22 and Definition 6.2.24, $\Gamma(x) = T$ iff $\widehat{\Gamma}(x) = \widehat{T}$. And again by Lemma 6.2.9, $\widehat{\Gamma}(x) = \widehat{T}$ iff $\widehat{\Gamma} \vdash x{:}\widehat{T}$. □

With this lemma we can now show that a translated typed term is well-typed in the monadic type system if and only if its origin is well-typed in the basic type system.

Lemma 6.2.39 (Preservation of Well-Typed Processes)**.** *Let* $P \in (\mathcal{P}_{\mathrm{a}}^{\sim} : \mathbb{T}_{\mathrm{B}})$, $P \in (\mathcal{P}_{\mathrm{p}}^{\sim} : \mathbb{T}_{\mathrm{B}})$, *or* $P \in (\mathcal{P}_{\mathrm{a}}^{=,\sim} : \mathbb{T}_{\mathrm{B}})$ *and* $\widehat{P}$ *its translation with respect to* Γ. *Then*

$$\Gamma \vdash P \quad \textit{iff} \quad \widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}$$

and $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ *implies* $\Delta = \Psi = \emptyset$.

Proof. We perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \}$ be the set of processes of the target language of the considered encoding and $\mathcal{P}^{\sim} \in \{ \mathcal{P}_{\mathrm{a}}^{\sim}, \mathcal{P}_{\mathrm{p}}^{\sim}, \mathcal{P}_{\mathrm{a}}^{=,\sim} \}$ the respective set of processes of the intermediate language allowing for polyadic communication.

Base Case: If $\Gamma \vdash P$ can be derived from one of the axioms in the basic type system then either $P = \mathsf{0}$ or $P = \checkmark$. In both cases $\widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}$ follows again directly from T-Nil$_{\mathrm{M}}$ or T-Succ$_{\mathrm{M}}$.

If $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ can be derived from one of the axioms in the monadic type system then $\Delta = \Psi = \emptyset$ and again either $\widehat{P} = \mathsf{0}$ or $\widehat{P} = \checkmark$. In both cases $\Gamma \vdash P$ follows by T-Nil$_{\mathrm{B}}$ or T-Succ$_{\mathrm{B}}$.

Induction Hypothesis: $\forall P \in (\mathcal{P} : \mathbb{T}_{\mathrm{B}}) \ . \ \Big(\Gamma \vdash P \quad \text{iff} \quad \widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}\Big)$ and $\Big(\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ implies $\Delta = \Psi = \emptyset\Big)$.

Induction Step: We consider first the goal: $\Gamma \vdash P$ implies $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$. The basic type system defines nine inference rules.

Case of T-Res$_\mathrm{B}$ **:** In this case, $P = (\nu x{:}T)\,P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{B}$, $P' \in (\mathcal{P}^{\sim}{:}\mathbb{T}_\mathrm{B})$, and $\Gamma, x:T \vdash P'$. Then $\widehat{P} = \left(\nu x{:}\widehat{T}\right)\widehat{P'}$ and, by the induction hypothesis, we have $\widehat{\Gamma}, x:\widehat{T};\emptyset;\emptyset \vdash \widehat{P'}$. Thus, we conclude $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$ by T-Res-B$_\mathrm{M}$.

Case of T-Par$_\mathrm{B}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}^{\sim}{:}\mathbb{T}_\mathrm{B})$ and $\Gamma \vdash P_1$ as well as $\Gamma \vdash P_2$. Then $\widehat{P} = \widehat{P_1} \mid \widehat{P_2}$ and, by the induction hypothesis, we have $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P_1}$ as well as $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P_2}$. Hence, because $\emptyset = \emptyset,\emptyset$ and, by Definition 6.2.27, also $\emptyset = \emptyset \cdot \emptyset$, we conclude $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$ by T-Par$_\mathrm{M}$.

Case of T-Mat$_\mathrm{B}$ **:** In this case, $P = [a = b]\,P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}^{\sim}{:}\mathbb{T}_\mathrm{B})$, $\Gamma \vdash a{:}T$, $\Gamma \vdash b{:}T$, and $\Gamma \vdash P'$ for some $T \in \mathbb{T}_\mathrm{B}$. Then $\widehat{P} = [a = b]\,\widehat{P'}$ and, by the induction hypothesis, we have $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P'}$. By Lemma 6.2.38, $\widehat{\Gamma} \vdash a:\widehat{T}$ and $\widehat{\Gamma} \vdash b{:}\widehat{T}$. We conclude $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$ by T-Mat$_\mathrm{M}$.

Case of T-Tau$_\mathrm{B}$ **:** In this case, $P = \tau.P'$ for some $P' \in (\mathcal{P}^{\sim}{:}\mathbb{T}_\mathrm{B})$ and $\Gamma \vdash P'$. Then $\widehat{P} = \tau.\widehat{P'}$ and, by the induction hypothesis, we have $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P'}$. Thus, we conclude $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$ by T-Tau$_\mathrm{M}$.

Case of T-Out$_\mathrm{B}$ **:** In this case, $P = \overline{y}\langle z_1, \ldots, z_n\rangle$ for some $n \geq 0$ and some names $y, z_1, \ldots, z_n \in \mathcal{N}$ and $\Gamma \vdash y{:}\sharp(T_1, \ldots, T_n)$, $\Gamma \vdash z_1{:}T_1$, $\ldots$, $\Gamma \vdash z_n{:}T_n$ for some $T_1, \ldots, T_n \in \mathbb{T}_\mathrm{B}$. If $n = 1$ then $\widehat{P} = P$ and, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y:\sharp\left(\widehat{T_1}\right)$ and $\widehat{\Gamma} \vdash z_1{:}\widehat{T_1}$. We conclude $\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}$ by T-Out-B$_\mathrm{M}$.

Else, $n > 1$. By Lemma 6.2.38,

$$\widehat{\Gamma} \vdash y{:}\sharp\left(\circ \triangleright \sharp\left(\natural\left(\widehat{T_1}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet\right), \text{ and} \tag{6.1}$$

$$\widehat{\Gamma} \vdash z_1{:}\widehat{T_1}, \ldots, \widehat{\Gamma} \vdash z_n{:}\widehat{T_n}. \tag{6.2}$$

and

$$\widehat{P} = (\nu u{:}T_u)\,(\overline{y}\langle u\rangle \mid u(u_1).\,(\overline{u_1}\langle z_1\rangle \mid \ldots \mid u(u_n).\,(\overline{u_n}\langle z_n\rangle)\ldots))$$

for $T_u = \circ \triangleright \sharp\left(\natural\left(\widehat{T_1}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet$. Then

$$\dfrac{\dfrac{D_1 \quad D_2}{\widehat{\Gamma}, u{:}T_u; u:\sharp\left(\natural\left(\widehat{T_1}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet;\emptyset \vdash Q}\,\text{T-Par}_\mathrm{M}}{\widehat{\Gamma};\emptyset;\emptyset \vdash \widehat{P}}\,\text{T-Res-M}_\mathrm{M}$$

where $Q = \overline{y}\langle u\rangle \mid u(u_1).\,(\overline{u_1}\langle z_1\rangle \mid \ldots \mid u(u_n).\,(\overline{u_n}\langle z_n\rangle)\ldots)$.

$$D_1 = \dfrac{\dfrac{}{\widehat{\Gamma}, u{:}T_u \vdash y:\sharp(T_u)}\,(6.1) \quad \dfrac{}{\widehat{\Gamma}, u{:}T_u \vdash u{:}T_u}\,\text{T-Name}_\mathrm{B}}{\widehat{\Gamma}, u{:}\circ \triangleright \sharp\left(\natural\left(\widehat{T_1}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet;\emptyset;\emptyset \vdash \overline{y}\langle u\rangle}\,\text{T-Out-B}_\mathrm{M}$$

Let $Q_1 = (\overline{u_1}\langle z_1\rangle \mid \ldots \mid u(u_n)\,.\,(\overline{u_n}\langle z_n\rangle)\ldots)$ and

$$T'_u = \sharp\Big(\sharp\Big(\widehat{T_2}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\sharp\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet,$$

then

$$D_2 = \dfrac{\dfrac{D'_2 \quad D_3}{\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\Big(\widehat{T_1}\Big)\,; u{:}T'_u; \emptyset \vdash Q_1}\text{T-Par}_\text{M}}{\widehat{\Gamma}, u{:}T_u; u{:}\sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet; \emptyset \vdash u(u_1)\,.Q_1}\text{T-In-M2}_\text{M}$$

where

$$D'_2 = \dfrac{\dfrac{}{\widehat{\Gamma}_1 \vdash u_1{:}\natural\Big(\widehat{T_1}\Big)}\text{T-Name}_\text{B} \quad \dfrac{}{\widehat{\Gamma}_1 \vdash z_1{:}\widehat{T_1}}(*)}{\widehat{\Gamma}_1; \emptyset; \emptyset \vdash \overline{u_1}\langle z_1\rangle}\text{T-Out-B}_\text{M}$$

$\widehat{\Gamma}_1 = \widehat{\Gamma}, u{:}T_u, u_1{:}\natural\Big(\widehat{T_1}\Big)$ and the subgoal $\widehat{\Gamma}_1 \vdash z_1{:}\widehat{T_1}$ marked by $(*)$ follows by (6.2) above and Lemma 6.2.11. Repeating the argumentation in D_2, i.e., applying Rule T-In-M2$_\text{M}$ then T-Par$_\text{M}$ and on the output-subgoal T-Name$_\text{B}$ and $(*)$, $n-2$ more times results in the subgoal

$$\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\Big(\widehat{T_1}\Big), \ldots, u_{n-1}{:}\natural\Big(\widehat{T_{n-1}}\Big)\,; u{:}\sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet; \emptyset \vdash u(u_n)\,.\overline{u_n}\langle z_n\rangle$$

Let $\widehat{\Gamma}_{n-1} = \widehat{\Gamma}, u{:}T_u, u_1{:}\natural\Big(\widehat{T_1}\Big), \ldots, u_{n-1}{:}\natural\Big(\widehat{T_{n-1}}\Big)$. We conclude with

$$D_{n+1} = \dfrac{\dfrac{D'_{n+1} \quad D''_{n+1}}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\Big(\widehat{T_n}\Big)\,; \emptyset; \emptyset \vdash \overline{u_n}\langle z_n\rangle}\text{T-Out-B}_\text{M}}{\widehat{\Gamma}_{n-1}; u{:}\sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet; \emptyset \vdash u(u_n)\,.\overline{u_n}\langle z_n\rangle}\text{T-In-M2}_\text{M}$$

where

$$D'_{n+1} = \dfrac{}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\Big(\widehat{T_n}\Big) \vdash u_n{:}\natural\Big(\widehat{T_n}\Big)}\text{T-Name}_\text{B}$$

and

$$D''_{n+1} = \dfrac{}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\Big(\widehat{T_n}\Big) \vdash z_n{:}\widehat{T_n}}\text{(6.2) and Lemma 6.2.11}$$

Case of T-OutPS$_\text{B}$ **:** Similar to the case before but instead of link y we have the link $y \cdot o$ in P, instead of (6.1) we have

$$\widehat{\Gamma} \vdash y{:}\mathfrak{v}_\mathfrak{n}, \text{ and} \tag{6.1.1}$$

$$\widehat{\Gamma} \vdash o{:}\sharp\Big(\circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet\Big) \tag{6.1.2}$$

and D_1 becomes

$$D_1 = \dfrac{\dfrac{}{\Gamma' \vdash y : \mathfrak{v}_\mathfrak{n}}(6.1.1) \quad \dfrac{}{\Gamma' \vdash o : \sharp(T_u)}(6.1.2) \quad \dfrac{}{\widehat{\Gamma}, u : T_u \vdash u : T_u} N}{\widehat{\Gamma}, u : \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet; \emptyset; \emptyset \vdash \overline{y \cdot o}\langle u \rangle} O$$

where $\Gamma' = \widehat{\Gamma}, u : T_u$, $N = \text{T-Name}_\text{B}$, and $O = \text{T-OutPS}_\text{M}$.

Case of T-In$_\text{B}$ **:** In this case, $P = y(x_1, \ldots, x_n).P'$ for some $n \geq 1$, some $y, x_1, \ldots, x_n \in \mathcal{N}$, and $P' \in (\mathcal{P}^{\sim} : \mathbb{T}_\text{B})$ and $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$, and $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P'$ for some $T_1, \ldots, T_n \in \mathbb{T}_\text{B}$. If $n = 1$ then $\widehat{P} = P$ and, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \sharp\Big(\widehat{T_1}\Big)$. By the induction hypothesis, we have $\widehat{\Gamma}, x_1 : \widehat{T_1}; \emptyset; \emptyset \vdash \widehat{P'}$ and $\widehat{T_1} \neq \circ \triangleright T'$ for all T'. We conclude $\widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}$ by T-In-B$_\text{M}$.

Else, $n > 1$. By Lemma 6.2.38,

$$\widehat{\Gamma} \vdash y : \sharp\Big(\circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet\Big)\,. \tag{6.3}$$

Let $\Gamma' = \Gamma, x_1 : T_1, \ldots, x_n : T_n$. Then

$$\widehat{P} = y(u).\Big(\Big(\nu u_1 : \natural\Big(\widehat{T_1}\Big)\Big)\Big(\overline{u}\langle u_1\rangle \mid u_1(x_1).\Big(\ldots \Big(\nu u_n : \natural\Big(\widehat{T_n}\Big)\Big)\Big(\overline{u}\langle u_n\rangle \mid u_n(x_n).\widehat{P'}\Big)\ldots\Big)\Big)\Big)$$

and, by the induction hypothesis, we have $\widehat{\Gamma'}; \emptyset; \emptyset \vdash \widehat{P'}$. Let $T_u = \circ\triangleright\sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big)\triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet$. Then:

$$D = \dfrac{\dfrac{}{\widehat{\Gamma} \vdash y : \sharp(T_u)}(6.3) \quad D_1}{\widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}} \text{T-In-M1}_\text{M}$$

Let $Q_1 = \overline{u}\langle u_1\rangle \mid u_1(x_1).\Big(\ldots\Big(\nu u_n : \natural\Big(\widehat{T_n}\Big)\Big)\Big(\overline{u}\langle u_n\rangle \mid u_n(x_n).\widehat{P'}\Big)\ldots\Big)$ and $\Psi_1 = u : \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet$. Then

$$D_1 = \dfrac{\dfrac{D_1' \quad D_1''}{\widehat{\Gamma}, u : T_u, u_1 : \natural\Big(\widehat{T_1}\Big); \emptyset; \Psi_1 \vdash Q_1} \text{T-Par}_\text{M}}{\widehat{\Gamma}, u : T_u; \emptyset; \Psi_1 \vdash \Big(\nu u_1 : \natural\Big(\widehat{T_1}\Big)\Big) Q_1} \text{T-Res-B}_\text{M}$$

Let $Q_2 = \overline{u}\langle u_2\rangle \mid u_2(x_2).\Big(\ldots\Big(\nu u_n : \natural\Big(\widehat{T_n}\Big)\Big)\Big(\overline{u}\langle u_n\rangle \mid u_n(x_n).\widehat{P'}\Big)\ldots\Big)$. By Definition 6.2.27, we can distribute Ψ_1 into $u : \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big)$ and $\Psi_2 = u :$

$\sharp\left(\natural\left(\widehat{T_2}\right)\right) \triangleright \ldots \triangleright \sharp\left(\natural\left(\widehat{T_n}\right)\right) \triangleright \bullet$ such that $\Psi_1 = u{:}\sharp\left(\natural\left(\widehat{T_1}\right)\right) \cdot \Psi_2$. Thus,

$$D_1' = \dfrac{\dfrac{}{\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right) \vdash u_1{:}\natural\left(\widehat{T_1}\right)}\text{T-Name}_\text{B}}{\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right); \emptyset; u{:}\sharp\left(\natural\left(\widehat{T_1}\right)\right) \vdash \overline{u}\langle u_1\rangle}\text{T-Out-M}_\text{M}$$

and

$$D_1'' = \dfrac{\dfrac{}{\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right) \vdash u_1{:}\natural\left(\widehat{T_1}\right)}\text{T-Name}_\text{B} \quad D_2}{\widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right); \emptyset; \Psi_2 \vdash u_1(x_1)\,.\left(\nu u_2{:}\natural\left(\widehat{T_2}\right)\right) Q_2}\text{T-In-B}_\text{M}$$

where D_2 introduces the subgoal $\widehat{\Gamma}_1; \emptyset; \Psi_2 \vdash \left(\nu u_2{:}\natural\left(\widehat{T_2}\right)\right) Q_2$ with $\widehat{\Gamma}_1 = \widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right), x_1{:}\widehat{T_1}$. Repeating the argumentation for D_1 again $n-2$ more times results in the subgoal

$$\widehat{\Gamma}_{n-1}; \emptyset; \Psi_n \vdash \left(\nu u_n{:}\natural\left(\widehat{T_n}\right)\right)\left(\overline{u}\langle u_n\rangle \mid u_n(x_n)\,.\widehat{P'}\right),$$

where $\widehat{\Gamma}_{n-1} = \widehat{\Gamma}, u{:}T_u, u_1{:}\natural\left(\widehat{T_1}\right), x_1{:}\widehat{T_1}, \ldots, u_{n-1}{:}\natural\left(\widehat{T_{n-1}}\right), x_{n-1}{:}\widehat{T_{n-1}}$ and $\Psi_n = u{:}\sharp\left(\natural\left(\widehat{T_n}\right)\right)$. We conclude with

$$D_n = \dfrac{\dfrac{D_n' \quad D_n''}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\left(\widehat{T_n}\right); \emptyset; \Psi_n \vdash \overline{u}\langle u_n\rangle \mid u_n(x_n)\,.\widehat{P'}}\text{T-Par}_\text{M}}{\widehat{\Gamma}_{n-1}; \emptyset; \Psi_n \vdash \left(\nu u_n{:}\natural\left(\widehat{T_n}\right)\right)\left(\overline{u}\langle u_n\rangle \mid u_n(x_n)\,.\widehat{P'}\right)}\text{T-Res-B}_\text{M}$$

where

$$D_n' = \dfrac{\dfrac{}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\left(\widehat{T_n}\right) \vdash u_n{:}\natural\left(\widehat{T_n}\right)}\text{T-Name}_\text{B}}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\left(\widehat{T_n}\right); \emptyset; u{:}\sharp\left(\natural\left(\widehat{T_n}\right)\right) \vdash \overline{u}\langle u_n\rangle}\text{T-Out-M}_\text{M}$$

and

$$D_n'' = \dfrac{\dfrac{}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\left(\widehat{T_n}\right) \vdash u_n{:}\natural\left(\widehat{T_n}\right)}\text{T-Name}_\text{B} \quad D_n'''}{\widehat{\Gamma}_{n-1}, u_n{:}\natural\left(\widehat{T_n}\right); \emptyset; \emptyset \vdash u_n(x_n)\,.\widehat{P'}}\text{T-In-B}_\text{M}$$

Let $\widehat{\Gamma}_n = \widehat{\Gamma}, u : T_u, u_1 : \natural\left(\widehat{T_1}\right), x_1 : \widehat{T_1}, \ldots, u_n : \natural\left(\widehat{T_n}\right), x_n : \widehat{T_n}$. It remains to show, for D_n''', that $\widehat{\Gamma}_n; \emptyset; \emptyset \vdash \widehat{P'}$. By the induction hypothesis, we have $\widehat{\Gamma}_n'; \emptyset; \emptyset \vdash \widehat{P'}$ with $\widehat{\Gamma}_n' = \widehat{\Gamma}, x_1 : \widehat{T_1}, \ldots, x_n : \widehat{T_n}$. Hence, we conclude by Lemma 6.2.29.

Case of T-InPS$_\mathrm{B}$ **:** Similar to the case before but instead of link y we have the link $y \cdot o$ in P, instead of (6.3) we have

$$\widehat{\Gamma} \vdash y : \mathfrak{v}_\mathfrak{n}, \text{ and} \tag{6.3.1}$$

$$\widehat{\Gamma} \vdash o : \sharp\Big(\circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet\Big) \tag{6.1.2}$$

and D becomes

$$D = \dfrac{\dfrac{}{\widehat{\Gamma} \vdash y : \mathfrak{v}_\mathfrak{n}}(6.3.1) \quad \dfrac{}{\widehat{\Gamma} \vdash o : \sharp(T_u)}(6.1.2) \quad D_1}{\widehat{\Gamma}; \emptyset; \emptyset \vdash \widehat{P}}\text{T-InPS}_\mathrm{M}$$

Case of T-Rep$_\mathrm{B}$ **:** Similar to the case of T-In$_\mathrm{B}$. In D apply T-Rep-M$_\mathrm{M}$ instead of T-In-M1$_\mathrm{M}$.

It remains to show that $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ implies $\Gamma \vdash P$ and $\Delta = \Psi = \emptyset$. Here, we distinguish between the inference rules of the monadic type system.

Case of T-Res-B$_\mathrm{M}$ **:** In this case, $\widehat{P} = (\nu x{:}T)\, P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{M}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, $\Delta = \emptyset$, $T \neq \circ \triangleright T'$ for all T', and $\widehat{\Gamma}, x{:}T; \Delta; \Psi \vdash P'$. Because of $\widehat{T} \neq \circ \triangleright T'$ for all $T' \in \mathbb{T}_\mathrm{M}$, T is no m-sort and, thus, there exists some $T'' \in \mathbb{T}_\mathrm{B}$ such that $T = \widehat{T''}$. Consequently, there also exists some $P'' \in (\mathcal{P}^\sim{:}\mathbb{T}_\mathrm{B})$ such that $P' = \widehat{P''}$ and $P = (\nu x{:}T'')\, P''$. Hence, by the induction hypothesis, $\widehat{\Gamma}, x{:}\widehat{T''}; \Delta; \Psi \vdash \widehat{P''}$ implies that $\Gamma, x{:}T'' \vdash P''$ and $\Delta = \Psi = \emptyset$. We conclude $\Gamma \vdash P$ by T-Res$_\mathrm{B}$.

Case of T-Res-M$_\mathrm{M}$ **:** Again $\widehat{P} = (\nu x{:}T)\, P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{M}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, and $\widehat{\Gamma}, x{:}T; \Delta; \Psi \vdash P'$ but here $T = T_1 \triangleright T_2$ for some T_1, T_2. Because of Definition 6.2.24, then $P = \overline{y}\langle z_1, \ldots, z_n\rangle$ (or $P = \overline{y \cdot o}\langle z_1, \ldots, z_n\rangle$) for some $y, z_1, \ldots, z_n \in \mathcal{N}$. Accordingly, $T = \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet$ and

$$P' = \overline{y}\langle u\rangle \mid u(u_1).\,(\overline{u_1}\langle z_1\rangle \mid \ldots \mid u(u_n).\,(\overline{u_n}\langle z_n\rangle)\ldots).$$

By the argumentation in the Case of T-Out$_\mathrm{B}$ above, i.e., by the derivation presented for $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$, we observe that $\widehat{\Gamma}; \Delta; \Psi \vdash \widehat{P}$ implies $\Delta = \Psi = \emptyset$, $\widehat{\Gamma}, u{:}T \vdash y : \sharp(T)$ and $\widehat{\Gamma}_i \vdash z_i : \widehat{T_i}$, where $\widehat{\Gamma}_i = \widehat{\Gamma}, u{:}T, u_1{:}\sharp\Big(\widehat{T_1}\Big), \ldots, u_i{:}\sharp\Big(\widehat{T_i}\Big)$, for all $1 \leq i \leq n$. Moreover, by Definition 6.2.22, $\sharp(T) = \sharp(\widehat{T_1, \ldots, T_n})$. Thus, by Lemma 6.2.11 and Lemma 6.2.38, $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$ and $\Gamma \vdash z_i : T_i$ for all $1 \leq i \leq n$. We conclude $\Gamma \vdash P$ by T-Out$_\mathrm{B}$.

Case of T-Par$_\mathrm{M}$ **:** In this case, $\widehat{P} = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}{:}\mathbb{T}_\mathrm{M})$, $\Delta = \Delta_1, \Delta_2$, $\Psi = \Psi_1 \cdot \Psi_2$, $\widehat{\Gamma}; \Delta_1; \Psi_1 \vdash P_1$, and $\widehat{\Gamma}; \Delta_2; \Psi_2 \vdash P_2$. By Definition 6.2.24, there exists $P_1', P_2' \in (\mathcal{P}^\sim{:}\mathbb{T}_\mathrm{B})$ such that $P_1 = \widehat{P_1'}$, $P_2 = \widehat{P_2'}$, and $P = P_1' \mid P_2'$. Then, by the induction hypothesis, $\widehat{\Gamma}; \Delta_1; \Psi_1 \vdash P_1$ and $\widehat{\Gamma}; \Delta_2; \Psi_2 \vdash P_2$ imply $\Gamma \vdash P_1'$, $\Gamma \vdash P_2'$, and $\Delta_1 = \Delta_2 = \Psi_1 = \Psi_2 = \emptyset$. Hence, $\Delta = \Delta_1, \Delta_2 = \emptyset$ and $\Psi = \Psi_1 \cdot \Psi_2 = \emptyset$. We conclude $\Gamma \vdash P$ by T-Par$_\mathrm{B}$.

Case of T-MAT$_{\mathrm{M}}$ **:** In this case, $\widehat{P} = [a = b]\, P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, $\widehat{\Gamma} \vdash a : T$, $\widehat{\Gamma} \vdash b : T$, for some $T \in \mathbb{T}_{\mathrm{M}}$,and $\widehat{\Gamma}; \Delta; \Psi \vdash P'$. By Definition 6.2.24, there exists $P'' \in (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$ such that $P' = \widehat{P''}$ and $P = [a = b]\, P''$. Then, by the induction hypothesis, $\widehat{\Gamma}; \Delta; \Psi \vdash P'$ implies $\Gamma \vdash P''$ and $\Delta = \Psi = \emptyset$. By Definition 6.2.22, there exists some $T' \in \mathbb{T}_{\mathrm{B}}$ such that $T = \widehat{T'}$. Hence, by Lemma 6.2.38, $\widehat{\Gamma} \vdash a : T$ and $\widehat{\Gamma} \vdash b : T$ imply $\Gamma \vdash a : T'$ and $\Gamma \vdash b : T'$. We conclude $\Gamma \vdash P$ by T-MAT$_{\mathrm{B}}$.

Case of T-TAU$_{\mathrm{M}}$ **:** In this case, $\widehat{P} = \tau.P'$ for some $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, $\Delta = \Psi = \emptyset$, and $\widehat{\Gamma}; \Delta; \Psi \vdash P'$. By Definition 6.2.24, there exists $P'' \in (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$ such that $P' = \widehat{P''}$ and $P = \tau.P''$. Then, by the induction hypothesis, $\widehat{\Gamma}; \Delta; \Psi \vdash P'$ implies $\Gamma \vdash P''$. We conclude $\Gamma \vdash P$ by T-TAU$_{\mathrm{B}}$.

Case of T-OUT-B$_{\mathrm{M}}$ **:** In this case, $\widehat{P} = \overline{y}\langle z \rangle$ for some $y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\widehat{\Gamma} \vdash z : T$, and either $\widehat{\Gamma} \vdash y : \sharp(T)$ or $\widehat{\Gamma} \vdash y : \natural(T)$ for some $T \in \mathbb{T}_{\mathrm{M}}$. By Definition 6.2.22 and Definition 6.2.24, there exists $T' \in \mathbb{T}_{\mathrm{B}}$ be such that $T = \widehat{T'}$, if we have $\widehat{\Gamma} \vdash y : \sharp(T)$. In the other case, i.e., if $\widehat{\Gamma} \vdash y : \natural(T)$, $P \notin (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$. Hence, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \sharp(T)$ and $\widehat{\Gamma} \vdash z : T$ imply $\Gamma \vdash y : \sharp(T')$ and $\Gamma \vdash z : T'$. We conclude $\Gamma \vdash P$ by T-OUT$_{\mathrm{B}}$.

Case of T-OUT-M$_{\mathrm{M}}$ **:** Here $\widehat{P} = \overline{y}\langle z \rangle$ for some $y, z \in \mathcal{N}$, $\Delta = \emptyset$, and $\widehat{\Gamma} \vdash z : T$ for some $T \in \mathbb{T}_{\mathrm{M}}$ but $\Psi = y : \sharp(T)$. However, by Definition 6.2.26, $\Psi = y : \sharp(T)$ implies $\Gamma(y) = \circ \triangleright T'$ for some T'. Thus, by Definition 6.2.24, $P \notin (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$.

Case of T-OUTPS$_{\mathrm{M}}$ **:** In this case, $\widehat{P} = \overline{y \cdot o}\langle z \rangle$ for some $o, y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\widehat{\Gamma} \vdash y : \mathfrak{v}_{\mathfrak{n}}$, $\widehat{\Gamma} \vdash o : \sharp(T)$, and $\widehat{\Gamma} \vdash z : T$ for some $T \in \mathbb{T}_{\mathrm{M}}$. By Definition 6.2.22, Definition 6.2.24, and because $\widehat{P} = \overline{y \cdot o}\langle z \rangle$, there exists $T' \in \mathbb{T}_{\mathrm{B}}$ be such that $T = \widehat{T'}$. Hence, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \mathfrak{v}_{\mathfrak{n}}$, $\widehat{\Gamma} \vdash o : \sharp(T)$, and $\widehat{\Gamma} \vdash z : T$ imply $\Gamma \vdash y : \mathfrak{v}_{\mathfrak{n}}$, $\Gamma \vdash o : \sharp(T')$, and $\Gamma \vdash z : T'$. We conclude $\Gamma \vdash P$ by T-OUTPS$_{\mathrm{B}}$.

Case of T-IN-B$_{\mathrm{M}}$ **:** In this case, $\widehat{P} = y(x).P'$ for some names $x, y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, $\Delta = \emptyset$, $\widehat{\Gamma}, x : T; \emptyset; \Psi \vdash P'$, and either $\widehat{\Gamma} \vdash y : \sharp(T)$ or $\widehat{\Gamma} \vdash y : \natural(T)$ for some $T \in \mathbb{T}_{\mathrm{M}}$ such that $T \neq \circ \triangleright T'$ for all T'. By Definition 6.2.22 and Definition 6.2.24, there exists $P'' \in (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$ such that $P' = \widehat{P''}$ and $P = y(x).P''$ and there exists $T'' \in \mathbb{T}_{\mathrm{B}}$ be such that $T = \widehat{T''}$, if we have $\widehat{\Gamma} \vdash y : \sharp(T)$. In the other case, i.e., if $\widehat{\Gamma} \vdash y : \natural(T)$, $P \notin (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$. Then, by the induction hypothesis, $\widehat{\Gamma}, x : T; \Delta; \Psi \vdash P'$ implies $\Gamma, x : T'' \vdash P''$ and $\Psi = \emptyset$. By Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \sharp(T)$ implies $\Gamma \vdash y : \sharp(T'')$. We conclude $\Gamma \vdash P$ by T-IN$_{\mathrm{B}}$.

Case of T-IN-M1$_{\mathrm{M}}$ **:** Again $\widehat{P} = y(x).P'$ for some names $x, y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, and $\Delta = \emptyset$ but here $\widehat{\Gamma} \vdash y : \sharp(T)$, $\Psi = \emptyset$, and $\widehat{\Gamma}, x : T; \emptyset; x : T' \vdash P'$ for some $T \in \mathbb{T}_{\mathrm{M}}$ such that $T = \circ \triangleright T'$. Thus, by Definition 6.2.24, there exists $x_1, \ldots, x_n \in \mathcal{N}$ and $P'' \in (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$ such that $P = y(x_1, \ldots, x_n).P''$,

$$P' = \Big(\nu u_1 : \natural\Big(\widehat{T_1}\Big)\Big)\Big(\overline{u}\langle u_1 \rangle \mid u_1(x_1).\Big(\ldots \Big(\nu u_n : \natural\Big(\widehat{T_n}\Big)\Big)\Big(\overline{u}\langle u_n \rangle \mid u_n(x_n).\widehat{P''}\Big)\ldots\Big)\Big),$$

and $T = \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet$ for some $T_1, \ldots, T_n \in \mathbb{T}_{\mathrm{B}}$. By the argumentation in the Case of T-In$_{\mathrm{B}}$ above, i.e., by the derivation presented for $\widehat{\Gamma}; \emptyset; \emptyset \vdash P$, we observe that $\widehat{\Gamma}; \emptyset; \emptyset \vdash P$ implies $\widehat{\Gamma}_n; \emptyset; \emptyset \vdash \widehat{P''}$, where $\widehat{\Gamma}_n = \widehat{\Gamma}, u : T, u_1 : \natural\Big(\widehat{T_1}\Big), x_1 : \widehat{T_1}, \ldots, u_n : \natural\Big(\widehat{T_n}\Big), x_n : \widehat{T_n}$. By Lemma 6.2.29, then $\widehat{\Gamma}'_n; \emptyset; \emptyset \vdash \widehat{P''}$, where $\widehat{\Gamma}'_n = \widehat{\Gamma}, x_1 : \widehat{T_1}, \ldots, x_n : \widehat{T_n}$. And, by the induction hypothesis, $\widehat{\Gamma}'_n; \emptyset; \emptyset \vdash \widehat{P''}$ implies $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P''$. Moreover, by Definition 6.2.22, $\sharp(T) = \sharp(\widehat{T_1, \ldots, T_n})$. Thus, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \sharp(T)$ implies $\Gamma \vdash y : \sharp(T_1, \ldots, T_n)$. We conclude $\Gamma \vdash P$ by T-In$_{\mathrm{B}}$.

Case of T-In-M2$_{\mathrm{M}}$ **:** Again $\widehat{P} = y(x).P'$ for some names $x, y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, and $\widehat{\Gamma}, x : T; \Delta; \Psi \vdash P'$ for some $T \in \mathbb{T}_{\mathrm{M}}$ but $\Delta = y : \sharp(T) \triangleright T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$, and $\Psi = \emptyset$. However, by Definition 6.2.26, $\Delta = y : \sharp(T) \triangleright T'$ implies $\Gamma(y) = \circ \triangleright T'$ for some T'. Thus, by Definition 6.2.24, $P \notin (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$.

Case of T-InPS$_{\mathrm{M}}$ **:** In this case $\widehat{P} = y \cdot o(x).P'$ for some names $o, x, y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_{\mathrm{M}})$, $\Delta = \Psi = \emptyset$, $\widehat{\Gamma} \vdash y : \mathfrak{v}_{\mathfrak{n}}$, $\widehat{\Gamma} \vdash o : \sharp(T)$, and $\widehat{\Gamma}, x : T; \emptyset; x : T' \vdash P'$ for some $T \in \mathbb{T}_{\mathrm{M}}$ such that $T = \circ \triangleright T'$. Thus, by Definition 6.2.24, there exists $x_1, \ldots, x_n \in \mathcal{N}$ and $P'' \in (\mathcal{P}^{\sim} : \mathbb{T}_{\mathrm{B}})$ such that $P = y \cdot o(x_1, \ldots, x_n).P''$,

$$P' = \Big(\nu u_1 : \natural\Big(\widehat{T_1}\Big)\Big)\Big(\overline{u}\langle u_1\rangle \mid u_1(x_1).\Big(\ldots \Big(\nu u_n : \natural\Big(\widehat{T_n}\Big)\Big)\Big(\overline{u}\langle u_n\rangle \mid u_n(x_n).\widehat{P''}\Big)\ldots\Big)\Big),$$

and $T = \circ \triangleright \sharp\Big(\natural\Big(\widehat{T_1}\Big)\Big) \triangleright \ldots \triangleright \sharp\Big(\natural\Big(\widehat{T_n}\Big)\Big) \triangleright \bullet$ for some $T_1, \ldots, T_n \in \mathbb{T}_{\mathrm{B}}$. By the argumentation in the Case of T-InPS$_{\mathrm{B}}$ (which is similar to the Case of T-In$_{\mathrm{B}}$) above, i.e., by the derivation presented for $\widehat{\Gamma}; \emptyset; \emptyset \vdash P$, we observe that $\widehat{\Gamma}; \emptyset; \emptyset \vdash P$ implies $\widehat{\Gamma}_n; \emptyset; \emptyset \vdash \widehat{P''}$, where $\widehat{\Gamma}_n = \widehat{\Gamma}, u : T, u_1 : \natural\Big(\widehat{T_1}\Big), x_1 : \widehat{T_1}, \ldots, u_n : \natural\Big(\widehat{T_n}\Big), x_n : \widehat{T_n}$. By Lemma 6.2.29, then $\widehat{\Gamma}'_n; \emptyset; \emptyset \vdash \widehat{P''}$, where $\widehat{\Gamma}'_n = \widehat{\Gamma}, x_1 : \widehat{T_1}, \ldots, x_n : \widehat{T_n}$. And, by the induction hypothesis, $\widehat{\Gamma}'_n; \emptyset; \emptyset \vdash \widehat{P''}$ implies $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P''$. Moreover, by Definition 6.2.22, $\sharp(T) = \sharp(\widehat{T_1, \ldots, T_n})$. Thus, by Lemma 6.2.38, $\widehat{\Gamma} \vdash y : \mathfrak{v}_{\mathfrak{n}}$ and $\widehat{\Gamma} \vdash o : \sharp(T)$ imply $\Gamma \vdash y : \mathfrak{v}_{\mathfrak{n}}$ and $\Gamma \vdash o : \sharp(T_1, \ldots, T_n)$. We conclude $\Gamma \vdash P$ by T-InPS$_{\mathrm{B}}$.

Case of T-Rep-B$_{\mathrm{M}}$ **:** Similar to the case of T-In-B$_{\mathrm{M}}$. $\widehat{P} = y^{\star}(x).P'$. Use T-Rep$_{\mathrm{B}}$ instead of T-In$_{\mathrm{B}}$.

Case of T-Rep-M$_{\mathrm{M}}$ **:** Similar to the case of T-In-M1$_{\mathrm{M}}$. $\widehat{P} = y^{\star}(x).P'$. Use the argumentation of the Case of T-Rep$_{\mathrm{B}}$ (which is similar to the Case of T-In$_{\mathrm{B}}$) instead of the Case of T-In$_{\mathrm{B}}$ and T-Rep$_{\mathrm{B}}$ instead of T-In$_{\mathrm{B}}$.

□

We conclude that the encoding functions are well-typed in the monadic type system.

Theorem 6.2.40. *All target terms of* $\mathcal{T}^1_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $\mathcal{T}^2_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $\mathcal{T}^3_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, *i.e., all terms* $P_1 \in (\mathcal{P}_{\mathrm{a}}{:}\mathbb{T}_{\mathrm{M}})\restriction_{\mathcal{T}^1_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $P_2 \in (\mathcal{P}_{\mathrm{p}}{:}\mathbb{T}_{\mathrm{M}})\restriction_{\mathcal{T}^2_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, *and* $P_3 \in (\mathcal{P}^{=}_{\mathrm{a}}{:}\mathbb{T}_{\mathrm{M}})\restriction_{\mathcal{T}^3_{\mathrm{M}}[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, *are modulo structural congruence well-typed with respect to* $\widehat{\Gamma_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}}$ *(and well-structured source terms),* $\widehat{\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}}$, *and* $\widehat{\Gamma_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}}$, *respectively.*

Proof. By Lemma 6.2.20, Observation 6.2.25, Lemma 6.2.37, and Lemma 6.2.39. □

Note that this theorem allows us to forget about the unfolding of polyadic communications and use the type information of the basic type system—or the type system introduced in the following—instead.

6.2.4. Polarity and Multiplicity

The type systems of the last two sections allow us to talk about the components of the encoding functions, which significantly eases the formulation and the proof of invariants. However, type systems can also be used directly to show properties of encodings. Within this section we use *polarities* and *multiplicities* to prove some properties on the usage of the links in the encoding functions. Polarities state how a link can be used [PS96], whereas multiplicities indicate how often a link can be used [San97, KPT99, San99].

A link can have one of three polarities. Links typed by the polarity $\uparrow$ can be used for output only, whereas $\downarrow$ denotes an input-only polarity. If the link should be used for both, we assign the polarity $\updownarrow$. Note that these polarities will replace the symbol $\sharp$ for link types. Furthermore, we augment this new symbol with a superscript and/or subscript to denote multiplicities. Linear links, i.e., links that are used for exactly one out- or input, are assigned multiplicity 1. $+$ is used to denote links that can be used at most once but can also be used not at all. In contrast ω means arbitrarily often, i.e., the link can be used any number of times from zero to any finite number. Moreover, we use the multiplicity $*$ to denote links that are used for exactly one replicated input but no other input. Of course, such links can hardly be considered as linear, since replicated inputs can be used arbitrarily often. Instead $*$ means that all input capabilities of that link are equivalent since they all stem from exactly the same replicated input capability. In [SW01] such links are called *ω-receptive*, because the existence of such replicated input automatically proves that once unguarded it provides a communication partner for each output on that link. Hence, $*$ ensures that all such outputs are eventually processed in exactly the same way, if the respective replicated input is not guarded.

A typical example of linear links, i.e., links that are used exactly once for input and exactly once for output are the auxiliary links that are introduced by the unfolding of polyadic communication and that are not typed by a m-sort like type. We type all such auxiliary links in all three encodings by $\updownarrow^1_1(T)$, where T is the type of the respective value. Note that as soon as a step on the respective link removes the only out- and input the corresponding restriction that introduces this link is superfluous and can be removed modulo structural congruence.

Apart from the protocol to unfold polyadic communications, all three encodings share sum locks and booleans. Remember that the positive instantiation of a sum lock l looks

like $\overline{l}\langle\top\rangle = l(t,f)\,.\overline{t}$, while $\overline{l}\langle\bot\rangle = l(t,f)\,.\overline{f}$ represents a negative instantiation. Hence, whether a sum lock is positively or negatively instantiated depends on the boolean link that is used for output. The boolean links t and f are then used in a test-construct $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q = (\nu t,f)\left(\overline{l}\langle t,f\rangle \mid t.P \mid f.Q\right)$ to guard the respective continuation of the then or the else-case. Here, the restriction ensures that for each pair of boolean links t,f there is exactly one input for each link. The instantiation of the sum lock, however, will only send a message on one of these links. Hence, for each boolean link there is at most one output and exactly one input. Accordingly, we type them by $\updownarrow_1^+(\mathfrak{v}_\top)$ and $\updownarrow_1^+(\mathfrak{v}_\bot)$. Note that within communications on sum locks only the output capability of booleans is communicated, while the linear input capability of the booleans remains in the respective test-construct. Because of this, we have to assign the type $\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))$ to the auxiliary link u_t and $\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))$ to the auxiliary link u_f. The same holds for all other polyadic communications in the three encodings, i.e., in polyadic communications only the output capabilities of the parameters is communicated. The encodings also ensure that there is always at most one instantiation of each sum lock. Moreover, for each consumed instantiation of a sum lock eventually a new instantiation is unguarded. But, unfortunately, these properties are not that easy to check. There is at most one instantiation of each sum lock, because to unguard a new one it is necessary to first consume one within a test-construct. For each so consumed instantiation exactly one new instantiation is eventually unguarded. But to prove that, we have to consider communications on various channels. First the instantiation is consumed by a communication on a sum lock. This starts the protocol behind the unfolding of polyadic communications, which requires in the case of sum locks four more steps. Finally, there is a step on a boolean. Hence to ensure that eventually each consumed sum lock instantiation is restored we need a more complex type system that also covers the relationships between links. In [Nes00] e.g. an ordering of dependencies between links is used to show that all links introduced by the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ are reliable, i.e., do not introduce deadlocks. As discussed in Section 6.5 it is not easy to extend this type system to reason about deadlock-freedom in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. Instead we use invariants to reason about such properties. Consequently, we do not assign a link type with special multiplicities to sum locks, but rely on the ordinary link type $\sharp(\mathfrak{l}_\circ)$. However, because we assign linear types to all auxiliary links and because again over the link $u_{\sim,l}$ of type $\mathfrak{l}_\circ$ only output capabilities are communicated, the m-sort like type $\mathfrak{l}_\circ$ becomes $\circ \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\big) \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\big) \triangleright \bullet$.

The encoded continuation of a source term sender is guarded in all three encodings by a sender lock (second sender lock in $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$). Since such a source term sender has exactly one (possibly empty) continuation, for each encoded source term sender there is exactly one sender lock and initially exactly one input on this lock. However, a successful emulation of a source term step may remove this single input, but since there may be still some guarded outputs on the sender lock, the corresponding restriction cannot be removed as it is the case for the linear types above. Consequently, we assign the type $\updownarrow_+^\omega(\mathfrak{v}_\mathfrak{s})$. Note that the ω for the output capability shows that our type system allows for arbitrary many (possibly guarded) outputs on sender locks.

Receiver locks in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ guard test-constructs by a replicated input. Again, $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ introduces exactly one receiver lock for each translated source term input and exactly one

Description	Names	Type
source term names	$\varphi_{\mathrm{a}}^{\mathrm{m}}(x)\,, \varphi_{\mathrm{a}}^{\mathrm{m}}(y)\,,$ $\varphi_{\mathrm{a}}^{\mathrm{m}}(z)\,, y, y', z$	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\updownarrow_1^+(\mathfrak{v}_{\top})$
	f	$\updownarrow_1^+(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\mathfrak{l}_{\circ})$
sender locks	s	$\mathfrak{s} = \updownarrow_+^{\omega}(\mathfrak{v}_{\mathfrak{s}})$
receiver locks	r	$\mathfrak{r} = \updownarrow_*^{\omega}(\mathfrak{r}_{\circ})$
output requests	$p_o, m_o, p_{o,up},$ $m_{o,up}, r_o, r_{o,up}$	$\mathfrak{o} = \sharp(\mathfrak{o}_{\circ})$, $\mathfrak{o}_* = \updownarrow_*^{\omega}(\mathfrak{o}_{\circ})$
input requests	$p_i, m_i, p_{i,up},$ $m_{i,up}, r_i, r_{i,up}$	$\mathfrak{i} = \sharp(\mathfrak{i}_{\circ})$, $\mathfrak{i}_* = \updownarrow_*^{\omega}(\mathfrak{i}_{\circ})$
chain locks	c_o	$\updownarrow_*^{\omega}(\downarrow_*(\mathfrak{i}_{\circ}))$
	c_i	$\updownarrow_*^{\omega}(\downarrow_*(\mathfrak{o}_{\circ}))$
	c_{r1}	$\updownarrow_*^{\omega}(\mathfrak{v}_{\mathfrak{n}})$
	c_{r2}	$\sharp(\mathfrak{c}_{\circ})$
auxiliary links	u_n	$\updownarrow_1^1(\mathfrak{v}_{\mathfrak{n}})$
	u_t	$\updownarrow_1^1(\uparrow^+(\mathfrak{v}_{\top}))$
	u_f	$\updownarrow_1^1(\uparrow^+(\mathfrak{v}_{\bot}))$
	u_l	$\updownarrow_1^1(\mathfrak{l})$
	u_s	$\updownarrow_1^1(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))$
	u_r	$\updownarrow_1^1(\uparrow^{\omega}(\mathfrak{r}_{\circ}))$
	u_o	$\updownarrow_1^1(\uparrow^{\omega}(\mathfrak{o}_{\circ}))$
	u_i	$\updownarrow_1^1(\uparrow^{\omega}(\mathfrak{i}_{\circ}))$
	$u_{\sim,l}$	$\mathfrak{l}_{\circ} = \circ \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_{\top}))\big) \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_{\bot}))\big) \triangleright \bullet$
	$u_{\sim,c}$	$\mathfrak{c}_{\circ} = \circ \triangleright \sharp\big(\uparrow^1(\uparrow^{\omega}(\mathfrak{o}_{\circ}))\big) \triangleright \sharp\big(\uparrow^1(\uparrow^{\omega}(\mathfrak{i}_{\circ}))\big) \triangleright \bullet$
	$u_{\sim,i}$	$\mathfrak{i}_{\circ} = \circ \triangleright \sharp\big(\uparrow^1(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^1(\uparrow^{\omega}(\mathfrak{r}_{\circ}))\big) \triangleright \bullet$
	$u_{\sim,o}$	$\mathfrak{o}_{\circ} = \circ \triangleright \sharp\big(\uparrow^1(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright$ $\sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^1(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^1(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \bullet$
	$u_{\sim,r}$	$\mathfrak{r}_{\circ} = \circ \triangleright \sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright$ $\sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^1(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^1(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \bullet$

Figure 6.9.: Linear and Monadic Types in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$.

replicated input for each such lock. Accordingly, we type them by $\updownarrow^{\omega}_{*}(\mathfrak{r}_{\circ})$. Remarkably, we can assign the same type also to most of the input and output requests. Inputs on request channels are usually replicated inputs, except for the inputs that collect right requests of a parallel operator. Note that $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ ensures that also for right requests there is eventually exactly one input. But again this property can only be proved by considering the relationships between links, because the respective input can be temporary guarded by a replicated input on a chain lock. Because of that, we use the ordinary link type for request channels that are restricted at the right hand side of a parallel operator encoding. Nonetheless, since the most difficult and challenging task of the algorithm introduced by $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is to guide the flow of requests, the special type of the remaining requests channels is still a very helpful information. Finally, there is also exactly one replicated input for each chain lock except for the second kind of chain locks associated to the translations of replicated inputs. An overview of all names and types used for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is presented in Figure 6.9. To obtain the typed variant of a term P, replace the syntactical representation of the request channels bound for the right hand side in the translations of the parallel operator by a fresh name, say $p_o{}'$ and $p_i{}'$, respectively. Then the typed variant of P is $\mathcal{T}^3_{\mathrm{L}}(P)$, where $\mathcal{T}^3_{\mathrm{L}}$ contains the assignments $p_o : \mathfrak{o}_*$, $p_i : \mathfrak{i}_*$, $p_o{}' : \mathfrak{o}$, and $p_i{}' : \mathfrak{i}$ and is defined for the remaining names by the type assignments given in Figure 6.9. In the following, whenever we write $\mathcal{T}^3_{\mathrm{L}}(P)$ for some $P \in \mathcal{P}^{=}_{\mathrm{a}}$ or $P = [\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ with $S \in \mathcal{P}_{\mathrm{m}}$ we silently assume that $\mathcal{T}^3_{\mathrm{L}}(P)$ is obtained in this way.

The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ introduces additional sender and receiver locks (the respective first parts) to retransmit requests after failed test-constructs, i.e., to compensate for aborted emulation attempts. For both locks $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ introduces exactly one replicated input. Hence, we type them by $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$. Moreover, for all request channels exactly one replicated input is provided. Apart from that, the names in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ are typed as in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Figure 6.10 provides an overview. Also, the definition of types for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ is straightforward. They are given in Figures 6.11. Figure 6.10 and 6.11 define also the sets of type assignments $\mathcal{T}^2_{\mathrm{L}}$ and $\mathcal{T}^1_{\mathrm{L}}$ that are used to obtain typed terms of the respective encoding.

We denote our third type system as linear type system, because it introduces linear types for some of the links. However, to avoid confusion with the *linear types*, which are link types with one of the multiplicities 1 or $*$, the types of the linear type system are denoted as multiplicity types. The set of multiplicity types is again defined by the union of the types in Figures 6.9, 6.10, and 6.11.

Definition 6.2.41 (Multiplicity Types)**.** Let $\mathbb{T}_{\mathrm{s}}$ be the types of the type system of the source language π_{s}. The types of the linear type system, called *multiplicity types*, are given by the multiplicity value types $\mathbb{V}_{\mathrm{L}}$, the multiplicity link types $\mathbb{L}_{\mathrm{L}}$, and the multiplicity types $\mathbb{T}_{\mathrm{L}} = \mathbb{V}_{\mathrm{L}} \cup \mathbb{L}_{\mathrm{L}}$. The multiplicity value types are defined as for the basic type system by the set

$$\mathbb{V}_{\mathrm{L}} \triangleq \mathbb{V}_{\mathrm{B}} = \{ \mathfrak{v}_{\mathfrak{n}}, \mathfrak{v}_{\top}, \mathfrak{v}_{\bot}, \mathfrak{v}_{\mathfrak{s}}, \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \} \cup \{ V_S \mid V_S \in \mathbb{T}_{\mathrm{s}} \wedge \forall T_S \in \mathbb{T}_{\mathrm{s}} \,.\, V_S \neq \sharp(T_S) \}$$

Description	Names	Type
source term names	$\varphi^{\mathrm{m}}_{\mathrm{p}}(x)\,,\varphi^{\mathrm{m}}_{\mathrm{p}}(y)\,,$ $\varphi^{\mathrm{m}}_{\mathrm{p}}(z)\,,y,z$	$\mathfrak{v}_{\mathfrak{n}}$
auxiliary values	v_t	$\mathfrak{v}_{\top}$
	v_f	$\mathfrak{v}_{\bot}$
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$
	v_s	$\mathfrak{v}_{\mathfrak{s}}$
booleans	t	$\updownarrow^{+}_{1}(\mathfrak{v}_{\top})$
	f	$\updownarrow^{+}_{1}(\mathfrak{v}_{\bot})$
sum locks	l, l_1, l_2, l_s, l_r	$\mathfrak{l} = \sharp(\mathfrak{l}_{\circ})$
sender and receiver locks	s_1, r_1, v, w	$\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$
	s_2	$\mathfrak{s} = \updownarrow^{\omega}_{+}(\mathfrak{v}_{\mathfrak{s}})$
	r_2	$\mathfrak{r}' = \updownarrow^{\omega}_{*}(\mathfrak{r}'_{\circ})$
output requests	$p_o, p_{o,up}$	$\mathfrak{o}' = \updownarrow^{\omega}_{*}(\mathfrak{o}'_{\circ})$
input requests	$p_i, p_{i,up}$	$\mathfrak{i}' = \updownarrow^{\omega}_{*}(\mathfrak{i}'_{\circ})$
tags	o	$\mathfrak{t}_{\mathfrak{o}} = \sharp(\mathfrak{t}_{\mathfrak{o},\circ})$
	i	$\mathfrak{t}_{\mathfrak{i}} = \sharp(\mathfrak{t}_{\mathfrak{i},\circ})$
auxiliary links	u_n	$\updownarrow^{1}_{1}(\mathfrak{v}_{\mathfrak{n}})$
	u_t	$\updownarrow^{1}_{1}(\uparrow^{+}(\mathfrak{v}_{\top}))$
	u_f	$\updownarrow^{1}_{1}(\uparrow^{+}(\mathfrak{v}_{\bot}))$
	u_l	$\updownarrow^{1}_{1}(\mathfrak{l})$
	$u_{s,r}$	$\updownarrow^{1}_{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))$
	u_s	$\updownarrow^{1}_{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))$
	$u_{r'}$	$\updownarrow^{1}_{1}(\uparrow^{\omega}(\mathfrak{r}'_{\circ}))$
	$u_{\sim,l}$	$\mathfrak{l}_{\circ} = \circ \triangleright \sharp\big(\uparrow^{1}(\uparrow^{+}(\mathfrak{v}_{\top}))\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{+}(\mathfrak{v}_{\bot}))\big) \triangleright \bullet$
	$u_{\sim,t_i}$	$\mathfrak{t}_{\mathfrak{i},\circ} =$ $\circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{r}'_{\circ}))\big) \triangleright \bullet$
	$u_{\sim,i'}$	$\mathfrak{i}'_{\circ} = \circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright$ $\sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{r}'_{\circ}))\big) \triangleright \bullet$
	$u_{\sim,t_o}$	$\mathfrak{t}_{\mathfrak{o},\circ} = \circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright$ $\sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \bullet$
	$u_{\sim,o'}$	$\circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright$ $\sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright \bullet$
	$u_{\sim,r}$	$\mathfrak{r}'_{\circ} = \circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright$ $\sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^{1}(\mathfrak{v}_{\mathfrak{n}})\big) \triangleright$ $\sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\big) \triangleright \bullet$

Figure 6.10.: Linear Types in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$.

Description	Names	Type
source term names	$\varphi^{\mathrm{s}}_{\mathrm{a}}(x)$	V_S
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$	$\sharp(\mathfrak{n}_{\circ,T_S})$
	$\varphi^{\mathrm{s}}_{\mathrm{a}}(z)$	V_S
		$\sharp(\mathfrak{n}_{\circ,T_S})$
auxiliary values	v_t	$\mathfrak{v}_\top$
	v_f	$\mathfrak{v}_\bot$
	v_s	$\mathfrak{v}_\mathfrak{s}$
	$v_{s,r}$	$\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$
booleans	t	$\updownarrow^+_1(\mathfrak{v}_\top)$
	f	$\updownarrow^+_1(\mathfrak{v}_\bot)$
sum locks	l, l'	$\mathfrak{l} = \sharp(\mathfrak{l}_\circ)$
sender locks	s	$\mathfrak{s} = \updownarrow^\omega_+(\mathfrak{v}_\mathfrak{s})$
receiver locks	r	$\updownarrow^\omega_*(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$
auxiliary links	u_{T_S}	$\updownarrow^1_1(T_S)$
	u_t	$\updownarrow^1_1(\uparrow^+(\mathfrak{v}_\top))$
	u_f	$\updownarrow^1_1(\uparrow^+(\mathfrak{v}_\bot))$
	u_l	$\updownarrow^1_1(\mathfrak{l})$
	u_s	$\updownarrow^1_1(\uparrow^\omega(\mathfrak{v}_\mathfrak{s}))$
	$u_{\sim,l}$	$\mathfrak{l}_\circ = \circ \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\big) \triangleright \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\big) \triangleright \bullet$
	$u_{\sim,T_S}$	$\mathfrak{n}_{\circ,T_S} = \circ \triangleright \sharp\big(\uparrow^1(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^1(\uparrow^\omega(\mathfrak{v}_\mathfrak{s}))\big) \triangleright \sharp\big(\uparrow^1(T_S)\big) \triangleright \bullet$

Figure 6.11.: Linear Types in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$.

Let M denote the set of types in Figures 6.9, 6.10, and 6.11, i.e.,

$$
\begin{aligned}
M = &\{\, \updownarrow^+_1(\mathfrak{v}_\top), \updownarrow^+_1(\mathfrak{v}_\bot), \mathfrak{l}, \mathfrak{s}, \mathfrak{r}, \mathfrak{o}, \mathfrak{i}, \sharp(\mathfrak{i}), \sharp(\mathfrak{o}), \sharp(\mathfrak{v}_\mathfrak{n}), \sharp(\mathfrak{c}_\circ), \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), \mathfrak{r}', \mathfrak{o}', \mathfrak{i}', \mathfrak{t}_\mathfrak{o}, \mathfrak{t}_\mathfrak{i}, \updownarrow^1_1(\mathfrak{v}_\mathfrak{n}) \,\} \\
&\cup \{\, \updownarrow^1_1\big(\uparrow^+(\mathfrak{v}_\top)\big), \updownarrow^1_1(\uparrow^+(\mathfrak{v}_\bot)), \updownarrow^1_1(\mathfrak{l}), \updownarrow^1_1(\uparrow^\omega(\mathfrak{v}_\mathfrak{s})), \updownarrow^1_1(\uparrow^\omega(\mathfrak{r}_\circ)), \updownarrow^1_1(\uparrow^\omega(\mathfrak{o}_\circ)), \updownarrow^1_1(\uparrow^\omega(\mathfrak{i}_\circ)) \,\} \\
&\cup \{\, \mathfrak{l}_\circ, \mathfrak{c}_\circ, \mathfrak{i}_\circ, \mathfrak{o}_\circ, \mathfrak{r}_\circ, \updownarrow^1_1(\uparrow^\omega(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})), \updownarrow^1_1\big(\uparrow^\omega(\mathfrak{r}'_\circ)\big), \mathfrak{t}_{\mathfrak{i},\circ}, \mathfrak{i}'_\circ, \mathfrak{t}_{\mathfrak{o},\circ}, \mathfrak{o}'_\circ, \mathfrak{r}'_\circ \,\} \\
&\cup \{\, \updownarrow^1_1(V_S) \mid V_S \in \mathbb{T}_{\mathrm{s}} \wedge \forall T_S \in \mathbb{T}_{\mathrm{s}} \,.\, V_S \neq \sharp(T_S) \,\} \\
&\cup \left\{\, \sharp(\mathfrak{n}_{\circ,T_S}), \updownarrow^1_1(T_S), \mathfrak{n}_{\circ,T_S} \mid \sharp\big(T'_S\big) \in \mathbb{T}_{\mathrm{s}} \wedge T_S = \widetilde{T'_S} \,\right\}
\end{aligned}
$$

Then, the set of multiplicity link types is defined as

$$
\mathbb{L}_{\mathrm{L}} = M \cup \{\, \uparrow^n(T), \downarrow_m(T) \mid \updownarrow^n_m(T) \in M \,\}.
$$

Moreover, let $\mathbb{T}_{\mathrm{lin}} = \{\, \updownarrow^1_1(T'), \uparrow^1(T'), \downarrow_1(T'), \updownarrow^+_1(T'), \updownarrow^\omega_*(T'), \downarrow_*(T') \mid T' \in \mathbb{T}_{\mathrm{L}} \,\}$ define the set of *linear types*, i.e., the types that contain a multiplicity 1 or $*$. Accordingly, we define:

$$
\begin{aligned}
\mathsf{lin}(\Gamma) &= \{\, x{:}T \mid x{:}T \in \Gamma \wedge T \in \mathbb{T}_{\mathrm{lin}} \,\} \\
\mathsf{lin}^+(\Gamma) &= \{\, x{:}T \mid x{:}T \in \Gamma \wedge T \in \big(\mathbb{T}_{\mathrm{lin}} \cup \{\, \updownarrow^\omega_+(T'), \uparrow^+(T'), \downarrow_+(T') \mid T' \in \mathbb{T}_{\mathrm{L}} \,\}\big) \,\}
\end{aligned}
$$

Note that we consider none of the link types $\sharp(\cdot)$, $\uparrow^{\omega}(\cdot)$, or $\uparrow^{+}(\cdot)$ as linear types even if they carry a linear type as argument. In contrast, for simplicity, we denote the types $\updownarrow^{\omega}_{*}(\cdot)$ and $\downarrow_{*}(\cdot)$ as linear types, although they are associated to unique replicated inputs.

The typing rules of the linear type system in Figure 6.12 are very similar to the typing rules of the monadic type system in Figure 6.8. Indeed, the Rule T-Res-$\mathrm{M_L}$ for restriction is equivalent to the Rule T-Res-$\mathrm{M_M}$ in the monadic type system. The same holds for the Rule T-$\mathrm{Tau_L}$ for τ-prefixes and the Rules T-Out-$\mathrm{M_L}$ and T-In-$\mathrm{M2_L}$ that are designed to cover communications on links typed by a m-sort. The Axioms T-$\mathrm{Name_L}$, T-$\mathrm{Nil_L}$, and T-Succ differ only by the additional side condition $\mathsf{lin}(\Gamma) = \emptyset$. It ensures that the typing by a linear multiplicity 1 or $*$ induces an obligation. A link typed by $\updownarrow^{1}_{1}(T)$ has to be used exactly once for input and exactly once for output. A term containing a name typed by a linear type cannot be well-typed before this obligation is satisfied.

The typing Rule T-Res-$\mathrm{B_L}$ is very similar to the corresponding Rule T-Res-$\mathrm{B_M}$ in the monadic type system, but it allows to "forget" a linear type to allow for restrictions on already consumed links with a linear type. Note that such restrictions are superfluous and can be removed modulo structural congruence.

The typing Rule T-$\mathrm{Par_L}$ for parallel compositions differs from T-$\mathrm{Par_M}$ by some additional requirements on the type environments. In the basic and the monadic type system each subgoal of a typing rule simply inherits the type environment of the goal. In the linear type system this is different. A linear multiplicity and also the multiplicity $+$ forbids that an output or input on the typed link occurs more than once. To ensure this, type environments contain information about capabilities of a multiplicity in $\{\, 1, *, + \,\}$ only if the respective capability did not already occur in the derivation so far. Intuitively, type environments only contain still active type information for linear capabilities and capabilities of multiplicity $+$. Hence, to check the type of a parallel composition, its type environment—or more precisely the parts of the type environment with a multiplicity in $\{\, 1, *, + \,\}$—have to be distributed on the two subgoals. How such a distribution has to look like is formalised by the function $\Gamma_1 + \Gamma_2$.

Definition 6.2.42. Let Γ_1 and Γ_2 be type environments. Then, for all $x \in \mathsf{n}(\Gamma_1) \cup \mathsf{n}(\Gamma_2)$,

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x)\,, & \text{if } x \notin \mathsf{n}(\Gamma_2) \\ \Gamma_2(x)\,, & \text{else if } x \notin \mathsf{n}(\Gamma_1) \\ \Gamma_1(x) + \Gamma_2(x)\,, & \text{else} \end{cases}$$

where

$$\begin{array}{rclcrcl} T + T' & = & T' + T & & \updownarrow^{\omega}_{*}(T) & = & \uparrow^{\omega}(T) + \downarrow_{*}(T) \\ \sharp(T) & = & \sharp(T) + \sharp(T) & & \updownarrow^{\omega}_{*}(T) & = & \uparrow^{\omega}(T) + \updownarrow^{\omega}_{*}(T) \\ \sharp(T) & = & \sharp(T) + \uparrow^{\omega}(T) & & \updownarrow^{\omega}_{+}(T) & = & \uparrow^{\omega}(T) + \downarrow_{+}(T) \\ T_1 \triangleright T_2 & = & T_1 \triangleright T_2 + T_1 \triangleright T_2 & & \updownarrow^{\omega}_{+}(T) & = & \uparrow^{\omega}(T) + \updownarrow^{\omega}_{+}(T) \\ \uparrow^{\omega}(T) & = & \uparrow^{\omega}(T) + \uparrow^{\omega}(T) & & \updownarrow^{+}_{1}(T) & = & \uparrow^{+}(T) + \downarrow_{1}(T) \\ \updownarrow^{1}_{1}(T) & = & \uparrow^{1}(T) + \downarrow_{1}(T) & & & & \end{array}$$

for all $T, T' \in \mathbb{T}_{\mathrm{L}}$ and all parts of a m-sort like type T_1, T_2.

In order to type terms like $\overline{y}\langle z\rangle \mid y(x).\mathsf{0}$ with respect to a linear type for y, the rules above allow to distribute a linear type constraint into its output and input part. The Rule $T + T' = T' + T$ ensures that $\Gamma_1 + \Gamma_2$ is symmetric for all type environments, i.e., $\Gamma_1 + \Gamma_2 = \Gamma_2 + \Gamma_1$. Moreover, the rules also ensure associativity, i.e., $\Gamma_1 + (\Gamma_2 + \Gamma_3) = (\Gamma_1 + \Gamma_2) + \Gamma_3$ for all environments Γ_1, Γ_2, and Γ_3. Note that for every typing rule of the linear type system with more than one subgoal it is required that the type environments of the subgoals are combined by $\Gamma + \Gamma'$ to obtain the type environment of the goal. This ensures that no linearly typed capability is used more than once.

Besides the additional requirement on the type environments, T-MAT$_\mathrm{L}$ differs from T-MAT$_\mathrm{M}$ by the side condition $T \in \mathbb{V}_\mathrm{L}$. Remember that the match prefix occurs only in $[\![\cdot]\!]^\mathrm{m}_\mathrm{a}$ and there only translated source term names typed by a value type are compared in a match. The side condition $T \in \mathbb{V}_\mathrm{L}$ forbids the use of links—names typed by a link type—in the match prefix of well-typed terms in the linear type system. Since we assign multiplicities and polarities only to link types, this ensures that the typing rule for match does not require special treatment in the linear type system. Hence, the use of values in a match prefix does not influence the counting on multiplicities of links.

The remaining rules for output, input, and replicated input check for the correct polarities of the respective links. For this, they use the abbreviations:

$$\begin{aligned} \sharp^{+}(T) &\triangleq \sharp(T) \vee \uparrow^{n}(T) \\ \sharp_{+}(T) &\triangleq \sharp(T) \vee \downarrow_{n}(T) \\ \sharp_{*}(T) &\triangleq \sharp(T) \vee \downarrow_{*}(T) \end{aligned}$$

where $n \in \{\, 1, +, \omega \,\}$. For example $\sharp^{+}(T)$ replaces in T-OUT-B$_\mathrm{L}$ the requirement $\sharp(T)$ in the T-OUT-B$_\mathrm{M}$ that ensures that each name used as link is indeed typed by a link type. $\sharp^{+}(T)$ allows for outputs on names typed by the conventional link type as well as for the new link types as long as they have the right polarity.

The typing rules of the linear type system are presented in Figure 6.12.

Definition 6.2.43 (Linear Type System)**.** The *linear type system* is given by the multiplicity types in Definition 6.2.41 and the typing rules in Figure 6.12.

In the following we show that also the linear type system satisfies a form of subject reduction and, thus, is consistent with the reduction semantics of the target languages of the encoding functions. For this, we consider again the preservation of the remaining properties. First note that Lemma 6.2.9 holds also in the case of T-NAME$_\mathrm{L}$, because the additional side condition $\mathsf{lin}(\Gamma)$ restricts only the set of not used type assignments. However, because of this side condition, weakening holds only in the case of not linear types. This shows that within the linear type system we have to be more careful with the surplus type assignments.

Lemma 6.2.44. *If* $\Gamma; \Delta; \Psi \vdash P$ *then* Γ *does not contain unnecessary linear type information, i.e.,* $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Proof. We perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{\, \mathcal{P}_\mathrm{a}, \mathcal{P}_\mathrm{p}, \mathcal{P}^{=}_\mathrm{a} \,\}$ be the set of processes of the target language of the considered encoding.

$$\text{T-Name}_\text{L}\ \frac{}{\Gamma, x{:}T \vdash x{:}T}\quad \mathsf{lin}(\Gamma) = \emptyset$$

$$\text{T-Nil}_\text{L}\ \frac{}{\Gamma; \emptyset; \emptyset \vdash \mathsf{0}}\quad \mathsf{lin}(\Gamma) = \emptyset \qquad \text{T-Succ}_\text{L}\ \frac{}{\Gamma; \emptyset; \emptyset \vdash \checkmark}\quad \mathsf{lin}(\Gamma) = \emptyset$$

$$\text{T-Res-B}_\text{L}\ \frac{\Gamma'; \Delta; \Psi \vdash P}{\Gamma; \Delta; \Psi \vdash (\nu x{:}T)\, P}\quad \begin{array}{c} T \neq \circ \triangleright T', \\ \Gamma' = \Gamma, x{:}T \vee (T \in \mathbb{T}_{\text{lin}} \wedge \Gamma' = \Gamma) \end{array}$$

$$\text{T-Res-M}_\text{L}\ \frac{\Gamma, x{:}T \triangleright T'; \Delta, \Delta'; \Psi \vdash P}{\Gamma; \Delta; \Psi \vdash (\nu x{:}T \triangleright T')\, P}\quad \Delta' = \begin{cases} \emptyset, & \text{if } T' = \bullet \\ x{:}T', & \text{else} \end{cases}$$

$$\text{T-Par}_\text{L}\ \frac{\Gamma_P; \Delta_P; \Psi_P \vdash P \quad \Gamma_Q; \Delta_Q; \Psi_Q \vdash Q}{\Gamma_P + \Gamma_Q; \Delta_P, \Delta_Q; \Psi_P \cdot \Psi_Q \vdash P \mid Q} \qquad \text{T-Tau}_\text{L}\ \frac{\Gamma; \emptyset; \emptyset \vdash P}{\Gamma; \emptyset; \emptyset \vdash \tau.P}$$

$$\text{T-Mat}_\text{L}\ \frac{\Gamma_1 \vdash a{:}T \quad \Gamma_2 \vdash b{:}T \quad \Gamma_3; \Delta; \Psi \vdash P}{\Gamma_1 + \Gamma_2 + \Gamma_3; \Delta; \Psi \vdash [a = b]\, P}\quad T \in \mathbb{V}_\text{L}$$

$$\text{T-Out-B}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\sharp^+(T) \quad \Gamma_2 \vdash z{:}T}{\Gamma_1 + \Gamma_2; \emptyset; \emptyset \vdash \overline{y}\langle z \rangle} \qquad \text{T-Out-M}_\text{L}\ \frac{\Gamma \vdash z{:}T}{\Gamma; \emptyset; y{:}\sharp(T) \vdash \overline{y}\langle z \rangle}$$

$$\text{T-OutPS}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma_2 \vdash o{:}\sharp^+(\circ \triangleright T) \quad \Gamma_3 \vdash z{:}\circ \triangleright T}{\Gamma_1 + \Gamma_2 + \Gamma_3; \emptyset; \emptyset \vdash \overline{y \cdot o}\langle z \rangle}$$

$$\text{T-In-B}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\sharp_+(T) \quad \Gamma_2, x{:}T; \emptyset; \Psi \vdash P}{\Gamma_1 + \Gamma_2; \emptyset; \Psi \vdash y(x)\,.P}\quad T \neq \circ \triangleright T'$$

$$\text{T-In-M1}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\sharp_+(\circ \triangleright T) \quad \Gamma_2, x{:}\circ \triangleright T; \emptyset; x{:}T \vdash P}{\Gamma_1 + \Gamma_2; \emptyset; \emptyset \vdash y(x)\,.P}$$

$$\text{T-In-M2}_\text{L}\ \frac{\Gamma, x{:}T; \Delta; \emptyset \vdash P}{\Gamma; y{:}\sharp(T) \triangleright T'; \emptyset \vdash y(x)\,.P}\quad \Delta = \begin{cases} \emptyset, & \text{if } T' = \bullet \\ y{:}T', & \text{else} \end{cases}$$

$$\text{T-InPS}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\mathfrak{v}_\mathfrak{n} \quad \Gamma_2 \vdash o{:}\sharp_+(\circ \triangleright T) \quad \Gamma_3, x{:}\circ \triangleright T; \emptyset; x{:}T \vdash P}{\Gamma_1 + \Gamma_2 + \Gamma_3; \emptyset; \emptyset \vdash y \cdot o(x)\,.P}$$

$$\text{T-Rep-B}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\sharp_*(T) \quad \Gamma_2, x{:}T; \emptyset; \emptyset \vdash P}{\Gamma_1 + \Gamma_2; \emptyset; \emptyset \vdash y^\star(x)\,.P}\quad T \neq \circ \triangleright T', \mathsf{lin}^+(\Gamma_2) = \emptyset$$

$$\text{T-Rep-M}_\text{L}\ \frac{\Gamma_1 \vdash y{:}\sharp_*(\circ \triangleright T) \quad \Gamma_2, x{:}\circ \triangleright T; \emptyset; x{:}T \vdash P}{\Gamma_1 + \Gamma_2; \emptyset; \emptyset \vdash y^\star(x)\,.P}\quad \mathsf{lin}^+(\Gamma_2) = \emptyset$$

Figure 6.12.: Typing Rules of the Linear Type System.

Base Case: If $\Gamma;\Delta;\Psi \vdash P$ can be derived from one of the axioms then either $P = 0$ or $P = \checkmark$ and $\mathsf{lin}(\Gamma) = \emptyset$.

Induction Hypothesis: $\Gamma;\Delta;\Psi \vdash P$ implies $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Induction Step: We perform a case split on the inference rules in Figure 6.12. Note that, by Definition 6.2.42, $\Gamma = \Gamma_1 + \Gamma_2$ implies $\mathsf{n}(\mathsf{lin}(\Gamma)) = \mathsf{n}(\mathsf{lin}(\Gamma_1)) \cup \mathsf{n}(\mathsf{lin}(\Gamma_2))$.

Case of T-Res-B$_\mathrm{L}$ **or** T-Res-M$_\mathrm{L}$ **:** Here, $P = (\nu x{:}T)\,P'$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_\mathrm{L}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, and $\Gamma';\Delta';\Psi \vdash P'$, where either $\Gamma' = \Gamma, x : T$ or $\Gamma' = \Gamma$ and $T \in \mathbb{T}_\mathrm{lin}$. Thus, by the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma')) \subseteq \mathsf{fn}(P')$. By Definition 6.2.26, $T \in \mathbb{T}_\mathrm{lin}$ and $\Gamma' = \Gamma, x{:}T$ imply $x \notin \mathsf{n}(\Gamma)$. Since $\mathsf{fn}(P) = \mathsf{fn}(P') \setminus \{\, x \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-Par$_\mathrm{L}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, $\Gamma = \Gamma_1 + \Gamma_2$, $\Gamma_1;\Delta_1;\Psi_1 \vdash P_1$, and $\Gamma_2;\Delta_2;\Psi_2 \vdash P_2$. Thus, by the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) \subseteq \mathsf{fn}(P_1)$ and $\mathsf{n}(\mathsf{lin}(\Gamma_2)) \subseteq \mathsf{fn}(P_2)$. Since $\mathsf{fn}(P) = \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-Tau$_\mathrm{L}$ **:** In this case, $P = \tau.P'$ for some $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$ and $\Gamma;\Delta;\Psi \vdash P'$. By the induction hypothesis, we have $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P')$. Since $\mathsf{fn}(P) = \mathsf{fn}(P')$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-Mat$_\mathrm{L}$ **:** In this case, $P = [a = b]\,P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash a : T$, $\Gamma_2 \vdash b : T$, $\Gamma_3;\Delta;\Psi \vdash P'$, and $\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3$. Because of the side condition of T-Name$_\mathrm{L}$, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) \subseteq \{\, a \,\}$ and $\mathsf{n}(\mathsf{lin}(\Gamma_2)) \subseteq \{\, b \,\}$. By the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma_3)) \subseteq \mathsf{fn}(P')$. Since $\mathsf{fn}(P) = \mathsf{fn}(P') \cup \{\, a, b \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-Out-B$_\mathrm{L}$ **:** In this case, $P = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$, $\Gamma_1 \vdash y{:}T_1$, $\Gamma_2 \vdash z{:}T_2$, and $\Gamma = \Gamma_1 + \Gamma_2$. By T-Name$_\mathrm{L}$, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) \subseteq \{\, y \,\}$ and $\mathsf{n}(\mathsf{lin}(\Gamma_2)) \subseteq \{\, z \,\}$. Since $\mathsf{fn}(P) = \{\, y, z \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-Out-M$_\mathrm{L}$ **:** Here, $P = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$ and $\Gamma \vdash z{:}T$. Again, by T-Name$_\mathrm{L}$, $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \{\, z \,\}$. Since $\mathsf{fn}(P) = \{\, y, z \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-OutPS$_\mathrm{L}$ **:** In this case, $P = \overline{y \cdot o}\langle z\rangle$ for some $o, y, z \in \mathcal{N}$, $\Gamma_1 \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma_2 \vdash y{:}T_1$, $\Gamma_3 \vdash z{:}T_2$, and $\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3$. By T-Name$_\mathrm{L}$, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) = \emptyset$, $\mathsf{n}(\mathsf{lin}(\Gamma_2)) \subseteq \{\, y \,\}$ and $\mathsf{n}(\mathsf{lin}(\Gamma_3)) \subseteq \{\, z \,\}$. Since $\mathsf{fn}(P) = \{\, o, y, z \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-In-B$_\mathrm{L}$, T-In-M1$_\mathrm{L}$, T-Rep-B$_\mathrm{L}$, **or** T-Rep-M$_\mathrm{L}$ **:** Here, $P = y(x)\,.P'$ or $P = y^\star(x)\,.P'$ for some $x, y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash y{:}T_1$, $\Gamma_2, x{:}T_2;\Delta;\Psi' \vdash P'$, and $\Gamma = \Gamma_1 + \Gamma_2$. By T-Name$_\mathrm{L}$, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) \subseteq \{\, y \,\}$. By the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma_2, x{:}T_2)) \subseteq \mathsf{fn}(P')$. By Definition 6.2.26, $\Gamma_2, x : T$ implies $x \notin \mathsf{n}(\mathsf{lin}(\Gamma_2))$. Since $\mathsf{fn}(P) = (\mathsf{fn}(P') \setminus \{\, x \,\}) \cup \{\, y \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-In-M2$_\mathrm{M}$ **:** In this case, $P = y(x)\,.P'$ for some $x, y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, and $\Gamma, x : T;\Delta';\Psi \vdash P'$. By the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma, x{:}T)) \subseteq \mathsf{fn}(P')$. By Definition 6.2.26, $\Gamma, x : T$ implies $x \notin \mathsf{n}(\mathsf{lin}(\Gamma))$. Since $\mathsf{fn}(P) = (\mathsf{fn}(P') \setminus \{\, x \,\}) \cup \{\, y \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

Case of T-InPS$_{\mathrm{M}}$ **:** In this case, $P = y{\cdot}o(x)\,.P'$ for some $o, x, y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{L}})$, $\Gamma_1 \vdash y : \mathfrak{b}_{\mathsf{n}}$, $\Gamma_2 \vdash o : T_1$, $\Gamma_3, x : T_2; \Delta; \Psi' \vdash P'$, and $\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3$. By T-Name$_{\mathrm{L}}$, $\mathsf{n}(\mathsf{lin}(\Gamma_1)) = \emptyset$ and $\mathsf{n}(\mathsf{lin}(\Gamma_2)) \subseteq \{\, o \,\}$. By the induction hypothesis, $\mathsf{n}(\mathsf{lin}(\Gamma_3, x{:}T_2)) \subseteq \mathsf{fn}(P')$. By Definition 6.2.26, $\Gamma_3, x{:}T$ implies $x \notin \mathsf{n}(\mathsf{lin}(\Gamma_3))$. Since $\mathsf{fn}(P) = (\mathsf{fn}(P') \setminus \{\, x \,\}) \cup \{\, o, y \,\}$, then $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$.

□

Because of this, we cannot remove type assignments with linear types from environments, because such assignments are never superfluous. By contrast, assignments with not linear types can still be superfluous and, if they are, can be removed.

Lemma 6.2.45 (Strengthening)**.** *If* $\Gamma, x{:}T; \Delta; \Psi \vdash P$ *and* $x \notin \mathsf{fn}(P)$ *then* $\Gamma; \Delta; \Psi \vdash P$.

Proof. By Lemma 6.2.44, $x \notin \mathsf{fn}(P)$ implies $x \notin \mathsf{n}(\mathsf{lin}(\Gamma, x{:}T))$. Hence, by Definition 6.2.42, if $\Gamma, x : T = \Gamma_1 + \Gamma_2$ then $x : T \in \Gamma_1$ and $x : T \in \Gamma_2$. Similarly, $\Gamma, x : T = \Gamma_1 + \Gamma_2 + \Gamma_3$ implies $x : T \in \Gamma_1$, $x : T \in \Gamma_2$, and $x : T \in \Gamma_3$. Because of this, the proof is similar to the proof of Lemma 6.2.29. □

Because of the side condition $\mathsf{lin}(\Gamma) = \emptyset$, weakening does not hold in the case of names typed by a linear type. However, for all remaining names, weakening is still valid.

Lemma 6.2.46 (Weakening)**.** *If* $\Gamma; \Delta; \Psi \vdash P$ *then* $\Gamma, x : T; \Delta; \Psi \vdash P$ *for any* $T \in \mathbb{T}_{\mathrm{L}}$ *and any name* x *such that* $\Gamma(x)$ *is not defined or equal to* T *and* $\mathsf{lin}(x{:}T) = \emptyset$.

Proof. By Lemma 6.2.44, $\Gamma; \Delta; \Psi \vdash P$ implies that $\mathsf{n}(\mathsf{lin}(\Gamma)) \subseteq \mathsf{fn}(P)$. Since $\mathsf{lin}(x{:}T) = \emptyset$, we have $\mathsf{lin}(\Gamma) = \mathsf{lin}(\Gamma, x{:}T)$. Hence, by Definition 6.2.42, $\Gamma_1, x : T + \Gamma_2, x : T = (\Gamma_1 + \Gamma_2)\,, x{:}T$ for all type environments Γ_1 and Γ_2 in the linear type system. The rest of the proof is similar to the argumentation in the proof of Lemma 6.2.30. □

Note that the main difference between the typing rules of the monadic type system and the typing rules of the linear type system is the additional requirement on the type environments, which is expressed by the function $\Gamma_1 + \Gamma_2$. Since this function is symmetric and associative, the linear type system inherits consistency with structural congruence from the monadic type system.

Lemma 6.2.47. *If* $\Gamma; \Delta; \Psi \vdash P$, $P \equiv Q$, *and* Γ *is closed for* Q *then* $\Gamma; \Delta; \Psi \vdash Q$.

Proof. By Definition 6.2.42, the operation $+$ on type environments is symmetric and associative such that $\Gamma_1 + \Gamma_2 = \Gamma_2 + \Gamma_1$ and $\Gamma_1 + (\Gamma_2 + \Gamma_3) = (\Gamma_1 + \Gamma_2) + \Gamma_3$. Moreover, note that the side condition of T-Res-B$_{\mathrm{L}}$ allows for the introduction of restrictions on fresh names also if they are typed by a linear type, because it allows to forget the respective type. Apart from this, the argumentation for this proof is similar to the argumentation in the proof of Lemma 6.2.31. □

The auxiliary sets Δ and Ψ and their use in the typing rules is not influenced by the changes to linear types. Hence, Lemma 6.2.32 remains valid. Also Lemma 6.2.33 remains valid.

Lemma 6.2.48. *If* $\Gamma;\Delta;\Psi \vdash P$ *then* Γ *is closed for* P*.*

Proof. For linear types the argumentation is similar to the proof of Lemma 6.2.44; for the remaining types it is similar to the proof of Lemma 6.2.33. □

Because weakening does require an additional side condition, we cannot prove robustness with respect to substitutions as in the other two type systems. As done in [SW01] we use a slightly different formulation instead, that is equivalent to the other formulations if there are no linear types. In contrast with the above substitution lemmas, the following substitution lemma removes the type assignment for the substituted name. After the substitution this assignment is superfluous. Hence, its removal ensures that it causes no problems even if it refers to a linear type.

Lemma 6.2.49. *Assume* $\Gamma, x:T;\Delta;\Psi \vdash P$*,* $\Gamma + \Gamma'$ *is defined, and* $\Gamma' \vdash z:T$*. Then* $\Gamma + \Gamma';\{\, z/x \,\}\Delta;\{\, z/x \,\}\Psi \vdash \{\, z/x \,\}P$*.*

Proof. We perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{\, \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \,\}$ be the set of processes of the target language of the considered encodings. Note that, by Lemma 6.2.44 and because of $\Gamma' \vdash z:T$, we have $\mathsf{lin}(\Gamma') \subseteq \{\, z:T \,\}$. By Definition 6.2.42, $\Gamma + \Gamma'$ implies that there are no clashes between assignments in Γ and Γ'.

Base Case: If $\Gamma, x:T;\Delta;\Psi \vdash P$ can be derived from one of the axioms then either $P = \mathsf{0}$ or $P = \checkmark$, $\Delta = \Psi = \emptyset$, and $\mathsf{lin}(\Gamma) = \emptyset$, i.e., $x \notin \mathsf{fn}(P)$, $\{\, z/x \,\}P = P$, $\{\, z/x \,\}\Delta = \Delta$, and $\{\, z/x \,\}\Psi = \Psi$. By Lemma 6.2.44, then $\mathsf{lin}(\Gamma') = \emptyset$. Thus, $\Gamma + \Gamma';\{\, z/x \,\}\Delta;\{\, z/x \,\}\Psi \vdash \{\, z/x \,\}P$ follows from T-Nil$_{\mathrm{L}}$ or T-Succ$_{\mathrm{L}}$.

Induction Hypothesis: $\Gamma, x:T;\Delta;\Psi \vdash P$, $\Gamma + \Gamma'$ is defined, and $\Gamma' \vdash z:T$ imply $\Gamma + \Gamma';\{\, z/x \,\}\Delta;\{\, z/x \,\}\Psi \vdash \{\, z/x \,\}P$

Induction Step: We perform a case split on the inference rules in Figure 6.12.

Case of T-Res-B$_{\mathrm{L}}$ **:** In this case, $P = (\nu x':T')\,P'$ for some $x' \in \mathcal{N}$, $T' \in \mathbb{T}_{\mathrm{L}}$, $P' \in (\mathcal{P}:\mathbb{T}_{\mathrm{L}})$, and $\Gamma, x:T, \Gamma_{x'};\Delta;\Psi \vdash P'$, where either $\Gamma_{x'} = x':T'$ or $T' \in \mathbb{T}_{\mathrm{lin}}$ and $\Gamma_{x'} = \emptyset$. Without loss of generality let us assume that $x \neq x' \neq z$ and $x' \notin \mathsf{n}(\Gamma) \cup \mathsf{n}(\Gamma')$. By the induction hypothesis, then $\Gamma, \Gamma_{x'} + \Gamma';\{\, z/x \,\}\Delta;\{\, z/x \,\}\Psi \vdash \{\, z/x \,\}P'$. We conclude by T-Res-B$_{\mathrm{L}}$.

Case of T-Res-M$_{\mathrm{L}}$ **:** This case is similar to the case before. Note that the type assignment $x:T'$ potentially added by T-Res-M$_{\mathrm{L}}$ to Δ is not influenced by the substitution, because $x' \neq x \neq z$.

Case of T-Par$_{\mathrm{L}}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}:\mathbb{T}_{\mathrm{L}})$, $\Gamma, x:T = \Gamma_1 + \Gamma_2$, $\Delta = \Delta_1, \Delta_2$, $\Psi = \Psi_1 \cdot \Psi_2$, $\Gamma_1;\Delta_1;\Psi_1 \vdash P_1$, and $\Gamma_2;\Delta_2;\Psi_2 \vdash P_2$. By Definition 6.2.42, $\Gamma_1 + \Gamma'$ and $\Gamma_2 + \Gamma'$ are defined if $\Gamma, x:T + \Gamma'$ is defined. Moreover, $\{\, z/x \,\}\Delta = \{\, z/x \,\}\Delta_1, \{\, z/x \,\}\Delta_2$ and $\{\, z/x \,\}\Psi = \{\, z/x \,\}\Psi_1 \cdot \{\, z/x \,\}\Psi_2$. By Definition 6.2.42, $\Gamma, x:T = \Gamma_1 + \Gamma_2$ implies that $x \in \mathsf{n}(\Gamma_1)$ or $x \in \mathsf{n}(\Gamma_2)$ or both. If $x \in \mathsf{n}(\Gamma_1)$ then, by the induction hypothesis, $\Gamma_1' + \Gamma';\{\, z/x \,\}\Delta_1;\{\, z/x \,\}\Psi_1 \vdash \{\, z/x \,\}P_1$, where $\Gamma_1' = \{\, y:T' \mid y \neq x \wedge y:T' \in \Gamma_1 \,\}$.

Else, by Lemma 6.2.48 and Lemma 6.2.32, $x \notin \mathsf{fn}(P_1) \cup \mathsf{n}(\Delta_1) \cup \mathsf{n}(\Psi_1)$. Similarly, if $x \in \mathsf{n}(\Gamma_2)$ then $\Gamma_2' + \Gamma'; \{ z/x \}\Delta_2; \{ z/x \}\Psi_2 \vdash \{ z/x \}P_2$, where $\Gamma_2' = \{ y{:}T' \mid y \neq x \wedge y{:}T' \in \Gamma_2 \}$, else $x \notin \mathsf{fn}(P_2) \cup \mathsf{n}(\Delta_2) \cup \mathsf{n}(\Psi_2)$. We conclude by T-PAR$_\mathrm{L}$.

Case of T-TAU$_\mathrm{L}$ **:** In this case, $P = \tau.P'$ for some $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$ and $\Gamma, x{:}T; \Delta; \Psi \vdash P'$. By the induction hypothesis, $\Gamma + \Gamma'; \{ z/x \}\Delta; \{ z/x \}\Psi \vdash \{ z/x \}P'$. We conclude by T-TAU$_\mathrm{L}$.

Case of T-MAT$_\mathrm{L}$ **:** In this case, $P = [a = b]\,P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash a{:}T'$, $\Gamma_2 \vdash b{:}T'$, $T' \in \mathbb{V}_\mathrm{L}$, $\Gamma_3; \Delta; \Psi \vdash P'$, and $\Gamma, x{:}T = \Gamma_1 + \Gamma_2 + \Gamma_3$. If $\mathsf{lin}(x{:}T) = \emptyset$ then $x{:}T \in \Gamma_1 \cap \Gamma_2 \cap \Gamma_3$. Else, by Lemma 6.2.44 and $T' \in \mathbb{V}_\mathrm{L}$, $x{:}T \in \Gamma_3$. In both cases there exists some Γ_3' such that $\Gamma_3', x{:}T = \Gamma_3$. By Definition 6.2.42, $\Gamma_3' + \Gamma'$ is defined if $\Gamma + \Gamma'$ is defined. By the induction hypothesis, then $\Gamma_3' + \Gamma'; \{ z/x \}\Delta; \{ z/x \}\Psi \vdash \{ z/x \}P'$. We conclude again by T-MAT$_\mathrm{L}$.

Case of T-OUT-B$_\mathrm{L}$ **:** In this case, $P = \overline{y}\langle z' \rangle$ for some $y, z' \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma_1 \vdash y : \sharp^+(T')$, $\Gamma_2 \vdash z' : T'$, and $\Gamma, x : T = \Gamma_1 + \Gamma_2$. By Definition 6.2.26, $x \notin \mathsf{n}(\Gamma)$. By Definition 6.2.42, $\Gamma_1' + \Gamma'$ and $\Gamma_2' + \Gamma'$ are defined if $\Gamma + \Gamma'$ is defined, where $\Gamma_i' = \{ x_1{:}T_1 \mid x_1 \neq x \wedge x_1{:}T_1 \in \Gamma_i \}$ for $i \in \{ 1, 2 \}$. Since $\sharp^+(T) \neq T$, either $y \notin \{ x, z \}$ or $z' \notin \{ x, z \}$. Moreover, by T-NAME$_\mathrm{L}$, $\Gamma_1 \vdash y{:}\sharp^+(T')$ and $\Gamma_2 \vdash z'{:}T'$ imply $\mathsf{lin}(\Gamma_1) \cap \mathsf{lin}(\Gamma_2) = \emptyset$.

If T is not a linear type then $x{:}T \in \Gamma_1 \cap \Gamma_2$. Hence, $\Gamma_1' + \Gamma' \vdash (\{ z/x \}\, y){:}\sharp^+(T')$ and $\Gamma_2' + \Gamma' \vdash (\{ z/x \}\, z'){:}T'$. We conclude by T-OUT-B$_\mathrm{L}$.

Else, by Lemma 6.2.44 and Definition 6.2.42, $\Gamma, x{:}T; \Delta; \Psi \vdash P$ implies that either $y = x \neq z'$, $x{:}T \in \Gamma_1 \setminus \Gamma_2$, and $\Gamma_1' + \Gamma' \vdash z : \sharp^+(T')$ or $y \neq x = z'$, $x{:}T \in \Gamma_2 \setminus \Gamma_1$, and $\Gamma_2' + \Gamma' \vdash z{:}T'$. In both cases we conclude by T-OUT-B$_\mathrm{L}$.

Case of T-OUT-M$_\mathrm{L}$ **:** In this case, $P = \overline{y}\langle z' \rangle$ for some $y, z' \in \mathcal{N}$, $\Delta = \emptyset$, $y \in \mathsf{n}(\Psi)$, and $\Gamma, x{:}T \vdash z'{:}T'$. If $x \neq z'$ then, by Lemma 6.2.44, T is not a linear type and $(\Gamma \setminus x{:}T) + \Gamma' \vdash z'{:}T'$. Else, if $x = z'$ then $T' = T$ and $(\Gamma \setminus x{:}T) + \Gamma' \vdash z{:}T$. In both cases we conclude again by T-OUT-M$_\mathrm{L}$.

Case of T-OUTPS$_\mathrm{L}$ **:** In this case, $P = \overline{y \cdot o}\langle z' \rangle$ for some $o, y, z' \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma_1 \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma_2 \vdash o : \sharp^+(T')$, $\Gamma_3 \vdash z' : T'$, and $\Gamma, x : T = \Gamma_1 + \Gamma_2 + \Gamma_3$. By Definition 6.2.26, $x \notin \mathsf{n}(\Gamma)$. By Definition 6.2.42, $\Gamma_1' + \Gamma'$, $\Gamma_2' + \Gamma'$, and $\Gamma_3' + \Gamma'$ are defined if $\Gamma + \Gamma'$ is defined, where $\Gamma_i' = \{ x_1{:}T_1 \mid x_1 \neq x \wedge x_1{:}T_1 \in \Gamma_i \}$ for $i \in \{ 1, 2, 3 \}$. Note that neither $\mathfrak{v}_\mathfrak{n}$ nor T' are linear types. Hence, by T-NAME$_\mathrm{L}$, $\mathsf{lin}(\Gamma_1) = \mathsf{lin}(\Gamma_3) = \emptyset$.

If T is not a linear type then $x{:}T \in \Gamma_1 \cap \Gamma_2 \cap \Gamma_3$. Hence, $\Gamma_1' + \Gamma' \vdash (\{ z/x \}\, y) : \mathfrak{v}_\mathfrak{n}$, $\Gamma_2' + \Gamma' \vdash (\{ z/x \}\, o) : \sharp^+(T')$, and $\Gamma_3' + \Gamma' \vdash (\{ z/x \}\, z') : T'$. We conclude by T-OUTPS$_\mathrm{L}$.

Else, by Lemma 6.2.44 and Definition 6.2.42, $x = o$, $x \notin \{ y, z' \}$, $x : T \in \Gamma_2 \setminus (\Gamma_1 \cup \Gamma_3)$, and $\Gamma_2' + \Gamma' \vdash z{:}\sharp^+(T')$. We conclude by T-OUTPS$_\mathrm{L}$.

Case of T-In-B$_\mathrm{L}$ **:** In this case, $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}:\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash y : \sharp_+(T')$, $\Gamma_2, x' : T'; \emptyset; \Psi \vdash P'$, $\Delta = \emptyset$, and $\Gamma, x : T = \Gamma_1 + \Gamma_2$. By Definition 6.2.42, $\Gamma'_1 + \Gamma'$ and $\Gamma'_2, x' : T' + \Gamma'$ are defined if $\Gamma + \Gamma'$ is defined, where $\Gamma'_i = \{\, x_1 : T_1 \mid x_1 \neq x \wedge x_1 : T_1 \in \Gamma_i \,\}$ for $i \in \{\, 1, 2 \,\}$. Without loss of generality let us assume $x \neq x'$.

If $x \in \mathsf{n}(\Gamma_2)$ then, by the induction hypothesis, $\Gamma'_2, x' : T' + \Gamma'; \emptyset; \{\, z/x \,\}\, \Psi \vdash \{\, z/x \,\}\, P'$. Else, by Lemma 6.2.48 and Lemma 6.2.32, $x \notin \mathsf{fn}(P') \cup \mathsf{n}(\Psi)$.

If T is not a linear type then $x : T \in \Gamma_1 \cap \Gamma_2$. Hence, $\Gamma'_1 + \Gamma' \vdash (\{\, z/x \,\}\, y) : \sharp^+(T')$. Else, by Lemma 6.2.44 and Definition 6.2.42, then either $y = x$ and again $\Gamma'_1 + \Gamma' \vdash (\{\, z/x \,\}\, y) : \sharp^+(T')$ or $y \neq x$ and $x \notin \Gamma_1$. In all cases we conclude by T-In-B$_\mathrm{L}$.

Case of T-In-M1$_\mathrm{L}$ **:** In this case, $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}:\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash y : \sharp_+(T')$, $\Gamma_2, x' : T'; \emptyset; x' : T'' \vdash P'$, $\Delta = \Psi = \emptyset$, and $\Gamma, x : T = \Gamma_1 + \Gamma_2$. By Definition 6.2.42, $\Gamma'_1 + \Gamma'$ and $\Gamma'_2, x' : T' + \Gamma'$ are defined if $\Gamma + \Gamma'$ is defined, where $\Gamma'_i = \{\, x_1 : T_1 \mid x_1 \neq x \wedge x_1 : T_1 \in \Gamma_i \,\}$ for $i \in \{\, 1, 2 \,\}$. Without loss of generality let us assume $x \neq x'$.

If $x \in \mathsf{n}(\Gamma_2)$ then, by the induction hypothesis, $\Gamma'_2, x' : T' + \Gamma'; \emptyset; x' : T'' \vdash \{\, z/x \,\}\, P'$. Else, by Lemma 6.2.48 and Lemma 6.2.32, $x \notin \mathsf{fn}(P')$.

If T is not a linear type then $x : T \in \Gamma_1 \cap \Gamma_2$. Hence, $\Gamma'_1 + \Gamma' \vdash (\{\, z/x \,\}\, y) : \sharp^+(T')$. Else, by Lemma 6.2.44 and Definition 6.2.42, then either $y = x$ and again $\Gamma'_1 + \Gamma' \vdash (\{\, z/x \,\}\, y) : \sharp^+(T')$ or $y \neq x$ and $x \notin \Gamma_1$. In all cases we conclude by T-In-M1$_\mathrm{L}$.

Case of T-In-M2$_\mathrm{L}$ **:** In this case, $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}:\mathbb{T}_\mathrm{L})$, $\Delta = y : T_1 \triangleright T_2$, $\Psi = \emptyset$, and $\Gamma, x : T, x' : T'; \Delta'; \emptyset \vdash P'$. Without loss of generality let us assume $x \neq x'$. Then $x \notin \mathsf{n}(\Delta')$. Since $y \in \mathsf{n}(\Delta)$ and by Definition 6.2.26, $T' \neq T$ and $x \neq y$. By the induction hypothesis, $\Gamma, x' : T' + \Gamma'; \Delta'; \emptyset \vdash \{\, z/x \,\}\, P'$. We conclude by T-In-M2$_\mathrm{L}$.

Case of T-InPS$_\mathrm{L}$ **:** In this case, $P = y \cdot o(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}:\mathbb{T}_\mathrm{L})$, $\Gamma_1 \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma_2 \vdash o : \sharp_+(T')$, $\Gamma_3, x' : T'; \emptyset; x' : T'' \vdash P'$, $\Delta = \Psi = \emptyset$, $\Gamma, x : T = \Gamma_1 + \Gamma_2 + \Gamma_3$, and T' is not a linear type. By Definition 6.2.42, $\Gamma'_1 + \Gamma'$, $\Gamma'_2 + \Gamma'$, and $\Gamma'_3, x' : T' + \Gamma'$ are defined if $\Gamma + \Gamma'$ is defined, where $\Gamma'_i = \{\, x_1 : T_1 \mid x_1 \neq x \wedge x_1 : T_1 \in \Gamma_i \,\}$ for $i \in \{\, 1, 2, 3 \,\}$.

If $x \in \mathsf{n}(\Gamma_3)$ then, by the induction hypothesis, $\Gamma'_3, x' : T' + \Gamma'; \emptyset; x' : T'' \vdash \{\, z/x \,\}\, P'$. Else, by Lemma 6.2.48 and Lemma 6.2.32, $x \notin \mathsf{fn}(P')$.

If T is not a linear type then $x : T \in \Gamma_1 \cap \Gamma_2 \cap \Gamma_3$. Hence, $\Gamma'_1 + \Gamma' \vdash (\{\, z/x \,\}\, y) : \mathfrak{v}_\mathfrak{n}$ and $\Gamma'_2 + \Gamma' \vdash (\{\, z/x \,\}\, o) : \sharp^+(T')$. Else, by Lemma 6.2.44 and Definition 6.2.42, then $y \neq x$, $x \notin \Gamma_1$, and either $x = o$ and again $\Gamma'_2 + \Gamma' \vdash (\{\, z/x \,\}\, o) : \sharp^+(T')$ or $y \neq x$ and $x \notin \Gamma_2$. In all cases we conclude by T-InPS$_\mathrm{L}$.

Case of T-Rep-B$_\mathrm{L}$ **:** Similar to the Case of T-In-B$_\mathrm{L}$.

Case of T-Rep-M$_\mathrm{L}$ **:** Similar to the Case of T-In-M1$_\mathrm{L}$.

□

To prove subject reduction we explicitly allow to remove superfluous type assignments in the type environment of the derivative. This is necessary to avoid problems with linear types.

Lemma 6.2.50 (Subject Reduction)**.** *If* $\Gamma;\Delta;\Psi \vdash P$, $P \longmapsto P'$, Γ *is closed for* P', *and* Δ *and* Ψ *are consistent then there exists* Γ' *and* Γ'' *such that* $\Gamma' + \Gamma'' = \Gamma$, $\Gamma';\Delta';\Psi' \vdash P'$ *and* Δ' *and* Ψ' *are consistent derivatives of* Γ, Δ, *and* Ψ.

Proof. We perform an induction on the depth of the derivation of $P \longmapsto P'$. Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \}$ be the set of processes of the target language of the considered encoding. Without loss of generality let us assume that there are no name clashes in P or P'.

Base Case: The reduction semantics of π_{a}, π_{p}, and $\pi_{\mathrm{a}}^{=}$ in Figure 2.3 contains the Axioms Pi-Tau$_{\mathrm{a,p}}$, Pi-Com$_{\mathrm{a,p}}$, Pi-ComPS$_{\mathrm{p}}$, and Pi-Rep$_{\mathrm{a,p}}$. The first rule requires that $P = \tau.Q$ and $P' = Q$ for some $Q \in (\mathcal{P} : \mathbb{T}_{\mathrm{L}})$. Hence, $\mathsf{fn}(P) = \mathsf{fn}(P')$ and $\Gamma;\Delta;\Psi \vdash P'$ follows from $\Gamma;\Delta;\Psi \vdash P$ and T-Tau$_{\mathrm{L}}$.

Rule Pi-Com$_{\mathrm{a,p}}$ requires that $P = y(x).Q \mid \overline{y}\langle z\rangle$ and $P' = \{ {}^{z}/_{x} \} Q$. In this case, the derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with

$$\frac{D_1 \quad D_2}{\Gamma;\Delta;\Psi \vdash y(x).Q \mid \overline{y}\langle z\rangle}\,\text{T-Par}_{\mathrm{L}}$$

where $\Gamma_1;\Delta_1;\Psi_1 \vdash y(x).Q$ is the goal of D_1 and $\Gamma_2;\Delta_2;\Psi_2 \vdash \overline{y}\langle z\rangle$ is the goal of D_2 such that $\Gamma = \Gamma_1 + \Gamma_2$, $\Delta = \Delta_1, \Delta_2$, and $\Psi = \Psi_1 \cdot \Psi_2$. Since there are no name clashes, $x \notin \mathsf{n}(\Gamma) \cup \mathsf{n}(\Delta) \cup \mathsf{n}(\Psi)$. On the left hand side we have to apply next one of the input Rules T-In-B$_{\mathrm{L}}$, T-In-M1$_{\mathrm{L}}$, or T-In-M2$_{\mathrm{L}}$ and on the right hand side one of the Rules T-Out-B$_{\mathrm{L}}$ or T-Out-M$_{\mathrm{L}}$.

Case of T-In-B$_{\mathrm{L}}$ **:** If D_1 is shown by T-In-B$_{\mathrm{L}}$, we have

$$D_1 = \frac{\dfrac{}{\Gamma_{1,1} \vdash y : \sharp_{+}(T)}\,\text{T-Name}_{\mathrm{L}} \quad \dfrac{\cdots}{\Gamma_{1,2}, x:T;\Delta_1;\Psi_1 \vdash Q}\,\cdots}{\Gamma_1;\Delta_1;\Psi_1 \vdash y(x).Q}\,\text{T-In-B}_{\mathrm{L}}$$

for some $T \in \mathbb{T}_{\mathrm{L}}$ that does not contain $\triangleright$ and $\Gamma_1 = \Gamma_{1,1} + \Gamma_{1,2}$. By Lemma 6.2.9 and Definition 6.2.42, $\Gamma_{1,1} \vdash y : \sharp_{+}(T)$ implies $\Gamma(y) \neq T_1 \triangleright T_2$. Because of that and by Definition 6.2.26, y does not occur in Δ or Ψ. Since $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$ and by Definition 6.2.27, this implies that y does also not occur in Δ_1, Δ_2, Ψ_1, or Ψ_2. Thus, we cannot apply T-Out-M$_{\mathrm{L}}$ for D_2. Hence,

$$D_2 = \frac{\dfrac{}{\Gamma_{2,1} \vdash y : \sharp^{+}(T')}\,\text{T-Name}_{\mathrm{L}} \quad \dfrac{}{\Gamma_{2,2} \vdash z : T'}\,\text{T-Name}_{\mathrm{L}}}{\Gamma_2;\Delta_2;\Psi_2 \vdash \overline{y}\langle z\rangle}\,\text{T-Out-B}_{\mathrm{L}}$$

where $\Delta_2 = \Psi_2 = \emptyset$ and $\Gamma_2 = \Gamma_{1,2} + \Gamma_{2,2}$. By Definition 6.2.27, then $\Delta = \Delta_1$ and $\Psi = \Psi_1$. By Lemma 6.2.9 and Definition 6.2.42, $\Gamma_{1,1}(y) = \sharp_{+}(T)$, $\Gamma_{2,1}(y) = \sharp^{+}(T')$, and $T = T'$. By Definition 6.2.42, $\Gamma_{1,2} + \Gamma_{2,2}$ is defined,

because $(\Gamma_{1,1} + \Gamma_{1,2}) + (\Gamma_{2,1} + \Gamma_{2,2})$ is defined. Because of Lemma 6.2.49, then $\Gamma_{1,2}, x{:}T; \Delta_1; \Psi_1 \vdash Q$ and $\Gamma_{2,2} \vdash z{:}T'$ imply $\Gamma_{1,2} + \Gamma_{2,2}; \Delta; \Psi \vdash P'$. Moreover, by Definition 6.2.42, Definition 6.2.41, and T-NAME$_\text{L}$, if $\mathsf{lin}(y{:}\sharp_+(T)) \neq \emptyset$ then $\Gamma_1 = \Gamma_{1,2} + y{:}\sharp_+(T)$ and $y{:}\sharp_+(T) \notin \Gamma_{1,2}$, and else $\Gamma_1 = \Gamma_{1,2}$. Similarly, if $\mathsf{lin}^+(y{:}\sharp^+(T)) \neq \emptyset$ then $\Gamma_2 = \Gamma_{2,2} + y{:}\sharp^+(T)$ and $y{:}\sharp^+(T) \notin \Gamma_{2,2}$, and else $\Gamma_2 = \Gamma_{2,2}$. In both cases we can choose $\Gamma' = \Gamma_{1,2} + \Gamma_{2,2}$.

Case of T-IN-M1$_\text{L}$ **:** If D_1 is shown by T-IN-M1$_\text{L}$, we have $\Delta_1 = \Psi_1 = \emptyset$ and

$$D_1 = \dfrac{\dfrac{}{\Gamma_{1,1} \vdash y{:}\sharp_+(\circ \triangleright T)}\text{T-NAME}_\text{L} \quad \dfrac{\cdots}{\Gamma_{1,2}, x{:}\circ \triangleright T; \emptyset; x{:}T \vdash Q}\cdots}{\Gamma_1; \emptyset; \emptyset \vdash y(x).Q}\text{T-IN-M1}_\text{L}$$

where $\Gamma_1 = \Gamma_{1,1} + \Gamma_{1,2}$. By Lemma 6.2.9 and Definition 6.2.42, $\Gamma_{1,1} \vdash y{:}\sharp(\circ \triangleright T)$ implies $\Gamma(y) = \sharp(\circ \triangleright T)$. Because of that and by Definition 6.2.26, y does not occur in $\Delta = \Delta_2$ or $\Psi = \Psi_2$. Thus, we cannot apply T-OUT-M$_\text{L}$ for D_2. Hence,

$$D_2 = \dfrac{\dfrac{}{\Gamma_{2,1} \vdash y{:}\sharp^+(T')}\text{T-NAME}_\text{L} \quad \dfrac{}{\Gamma_{2,2} \vdash z{:}T'}\text{T-NAME}_\text{L}}{\Gamma_2; \Delta_2; \Psi_2 \vdash \overline{y}\langle z\rangle}\text{T-OUT-B}_\text{M}$$

where $\Gamma_2 = \Gamma_{2,1} + \Gamma_{2,2}$ and $\Delta_2 = \Psi_2 = \emptyset$. By Lemma 6.2.9 and Definition 6.2.42, $\Gamma_{1,1}(y) = \sharp_+(\circ \triangleright T)$, $\Gamma_{2,1}(y) = \sharp^+(T')$, and $T' = \circ \triangleright T$. By Definition 6.2.42, $\Gamma_{1,2} + \Gamma_{2,2}$ is defined, because $(\Gamma_{1,1} + \Gamma_{1,2}) + (\Gamma_{2,1} + \Gamma_{2,2})$ is defined. Because of Lemma 6.2.49, then $\Gamma_{1,2}, x{:}T'; \emptyset; x{:}T \vdash Q$ and $\Gamma_{2,2} \vdash z{:}T'$ imply $\Gamma_{1,2} + \Gamma_{2,2}; \emptyset; z{:}T \vdash P'$. Moreover, by Definition 6.2.42, Definition 6.2.41, and T-NAME$_\text{L}$, if $\mathsf{lin}(y{:}\sharp_+(T')) \neq \emptyset$ then $\Gamma_1 = \Gamma_{1,2} + y{:}\sharp_+(T')$ and $y{:}\sharp_+(T') \notin \Gamma_{1,2}$, and else $\Gamma_1 = \Gamma_{1,2}$. Similarly, if $\mathsf{lin}^+(y{:}\sharp^+(T')) \neq \emptyset$ then $\Gamma_2 = \Gamma_{2,2} + y{:}\sharp^+(T')$ and $y : \sharp^+(T') \notin \Gamma_{2,2}$, and else $\Gamma_2 = \Gamma_{2,2}$. In both cases we can choose $\Gamma' = \Gamma_{1,2} + \Gamma_{2,2}$. Note that $\emptyset$ and $z{:}T$ are consistent and are derivatives of Γ, $\emptyset$, and $\emptyset$.

Case of T-IN-M2$_\text{L}$ **:** If D_1 is shown by T-IN-M2$_\text{L}$, we have $\Delta_1 = y : \sharp(T) \triangleright T'$, $\Psi_1 = \emptyset$, and

$$D_1 = \dfrac{\dfrac{\cdots}{\Gamma_1, x{:}T; \Delta_1'; \emptyset \vdash Q}\cdots}{\Gamma_1; y{:}\sharp(T) \triangleright T'; \emptyset \vdash y(x).Q}\text{T-IN-M2}_\text{L}$$

where Δ_1' is $\emptyset$ if $T' = \bullet$ and else $\Delta_1' = y : T'$. By Definition 6.2.26 and Definition 6.2.42, $y \in \mathsf{n}(\Delta_1)$ implies $\Gamma_1(y) = \circ \triangleright T_y = \Gamma(y)$ for some T_y. Thus, by Lemma 6.2.9, we cannot apply T-OUT-B$_\text{L}$ on D_2. Hence,

$$D_2 = \dfrac{\dfrac{}{\Gamma_2 \vdash z{:}T''}\text{T-NAME}_\text{L}}{\Gamma_2; \Delta_2; \Psi_2 \vdash \overline{y}\langle z\rangle}\text{T-OUT-M}_\text{L}$$

where $\Delta_2 = \emptyset$ and $\Psi_2 = y : \sharp(T'')$. Thus, $\Delta = \Delta_1$ and $\Psi = \Psi_2$. Since Δ and Ψ are consistent and by Definition 6.2.35, $T'' = T \neq T'$. Because of Lemma 6.2.49, then $\Gamma_1, x{:}T; \Delta_1'; \emptyset \vdash Q$ and $\Gamma_2 \vdash z{:}T''$ imply $\Gamma; \Delta_1'; \emptyset \vdash P'$. Note that Δ_1' and $\emptyset$ are consistent and are derivatives of Γ, Δ_1, and Ψ_2.

Rule Pi-ComPS$_{\mathrm{p}}$ requires that $P = y \cdot o(x).Q \mid \overline{y \cdot o}\langle z\rangle$ and $P' = \{\, {}^{z}/_{x} \,\} Q$. In this case, the derivation of $\Gamma; \Delta; \Psi \vdash P$ starts again with

$$\frac{D_1 \quad D_2}{\Gamma; \Delta; \Psi \vdash y \cdot o(x).Q \mid \overline{y \cdot o}\langle z\rangle}\,\text{T-Par}_{\mathrm{L}}$$

where $\Gamma_1; \Delta_1; \Psi_1 \vdash y \cdot o(x).Q$ is the goal of D_1 and $\Gamma_2; \Delta_2; \Psi_2 \vdash \overline{y \cdot o}\langle z\rangle$ is the goal of D_2 such that $\Gamma = \Gamma_1 + \Gamma_2$, $\Delta = \Delta_1, \Delta_2$, and $\Psi = \Psi_1 \cdot \Psi_2$. On the left hand side we have to apply next T-InPS$_{\mathrm{L}}$ and on the right hand side T-OutPS$_{\mathrm{L}}$. Hence, we have $\Delta_1 = \Psi_1 = \emptyset$ and

$$D_1 = \frac{\dfrac{}{\Gamma_{1,1} \vdash y : \mathfrak{b}_{\mathfrak{n}}}N \quad \dfrac{}{\Gamma_{1,2} \vdash o : \sharp_{+}(\circ \triangleright T)}N \quad \dfrac{\dfrac{\cdots}{\Gamma_{1,3}, x : \circ \triangleright T; \emptyset; x : T \vdash Q}\cdots}{\Gamma_1; \emptyset; \emptyset \vdash y \cdot o(x).Q}I$$

where N = T-Name$_{\mathrm{L}}$, I = T-InPS$_{\mathrm{L}}$, and $\Gamma_1 = \Gamma_{1,1} + \Gamma_{1,2} + \Gamma_{1,3}$. Moreover, $\Delta_2 = \Psi_2 = \emptyset$ and

$$D_2 = \frac{\dfrac{}{\Gamma_{2,1} \vdash y : \mathfrak{b}_{\mathfrak{n}}}N \quad \dfrac{}{\Gamma_{2,2} \vdash o : \sharp^{+}(T')}N \quad \dfrac{}{\Gamma_{2,3} \vdash z : T'}N}{\Gamma_2; \emptyset; \emptyset \vdash \overline{y}\langle z\rangle}\,\text{T-OutPS}_{\mathrm{L}}$$

where N = T-Name$_{\mathrm{L}}$ and $\Gamma_2 = \Gamma_{2,1} + \Gamma_{2,2} + \Gamma_{2,3}$. By Definition 6.2.27, then $\Delta = \Psi = \emptyset$. By Lemma 6.2.9 and Definition 6.2.42, $\Gamma_{1,2}(o) = \sharp_{+}(\circ \triangleright T)$, $\Gamma_{2,2}(o) = \sharp^{+}(T')$, and $T' = \circ \triangleright T$. By Definition 6.2.42, $\Gamma_{1,3} + \Gamma_{2,3}$ is defined, because $(\Gamma_{1,1} + \Gamma_{1,2} + \Gamma_{1,3}) + (\Gamma_{2,1} + \Gamma_{2,2} + \Gamma_{2,3})$ is defined. Because of Lemma 6.2.49, then $\Gamma_{1,3}, x : T'; \emptyset; x : T \vdash Q$ and $\Gamma_{2,3} \vdash z : T'$ imply $\Gamma_{1,3} + \Gamma_{2,3}; \emptyset; z : T \vdash P'$. Moreover, by Definition 6.2.42, Definition 6.2.41, and T-Name$_{\mathrm{L}}$, if $\mathsf{lin}(o : \sharp_{+}(T')) \neq \emptyset$ then $\Gamma_1 = \Gamma_{1,3} + o : \sharp_{+}(T')$ and $o : \sharp_{+}(T') \notin \Gamma_{1,3}$, and else $\Gamma_1 = \Gamma_{1,3}$. Similarly, if $\mathsf{lin}^{+}(o : \sharp^{+}(T')) \neq \emptyset$ then $\Gamma_2 = \Gamma_{2,3} + o : \sharp^{+}(T')$ and $o : \sharp^{+}(T') \notin \Gamma_{2,3}$, and else $\Gamma_2 = \Gamma_{2,3}$. In both cases we can choose $\Gamma' = \Gamma_{1,3} + \Gamma_{2,3}$. Again, $\emptyset$ and $z : T$ are consistent and are derivatives of Γ, $\emptyset$, and $\emptyset$.

Rule Pi-Rep$_{\mathrm{a,p}}$ requires that $P = y^{\star}(x).Q \mid \overline{y}\langle z\rangle$ and $P' = \{\, {}^{z}/_{x} \,\} Q \mid y^{\star}(x).Q$. The derivation of $\Gamma; \Delta; \Psi \vdash P$ starts with

$$\frac{D_1 \quad D_2}{\Gamma; \Delta; \Psi \vdash y^{\star}(x).Q \mid \overline{y}\langle z\rangle}\,\text{T-Par}_{\mathrm{L}}$$

where $\Gamma_1; \Delta_1; \Psi_1 \vdash y^{\star}(x).Q$ is the goal of D_1 and $\Gamma_2; \Delta_2; \Psi_2 \vdash \overline{y}\langle z\rangle$ is the goal of D_2 such that $\Gamma = \Gamma_1 + \Gamma_2$, $\Delta = \Delta_1, \Delta_2$, and $\Psi = \Psi_1 \cdot \Psi_2$. On the left hand side we have to apply next T-Rep-B$_{\mathrm{L}}$ or T-Rep-M$_{\mathrm{L}}$. In the first case the rest of the proof is similar to the Case of T-In-B$_{\mathrm{L}}$ for Pi-Com$_{\mathrm{a,p}}$ and in the other case the rest of the proof is similar to the Case of T-In-M1$_{\mathrm{L}}$ for Pi-Com$_{\mathrm{a,p}}$.

Induction Hypothesis: $\Gamma; \Delta; \Psi \vdash P$, $P \longmapsto P'$, Γ is closed for P', and Δ and Ψ are consistent imply that there exists Γ' and Γ'' such that $\Gamma' + \Gamma'' = \Gamma$, $\Gamma'; \Delta'; \Psi' \vdash P'$, and Δ' and Ψ' are consistent derivatives of Γ, Δ, and Ψ.

Induction Step: There are three cases.

Case of Pi-Par$_{\mathrm{m,s,a,p}}$ **:** In this case, $P = P_1 \mid P_2$, $P_1 \longmapsto P_1'$, and $P' = P_1' \mid P_2$ for some $P_1, P_1', P_2 \in (\mathcal{P}:\mathbb{T}_{\mathrm{L}})$. The derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with

$$\dfrac{\dfrac{\cdots}{\Gamma_1;\Delta_1;\Psi_1 \vdash P_1}\cdots \quad \dfrac{\cdots}{\Gamma_2;\Delta_2;\Psi_2 \vdash P_2}\cdots}{\Gamma;\Delta;\Psi \vdash P_1 \mid P_2}\text{T-Par}_{\mathrm{L}}$$

where $\Gamma = \Gamma_1 + \Gamma_2$, $\Delta = \Delta_1, \Delta_2$, and $\Psi = \Psi_1 \cdot \Psi_2$. By the induction hypothesis, we have $\Gamma_1';\Delta_1';\Psi_1' \vdash P_1'$ such that Δ_1' and Ψ_1' are consistent derivatives of Γ_1, Δ_1, and Ψ_1, and $\Gamma_1' + \Gamma_1'' = \Gamma_1$. We can choose $\Gamma' = \Gamma_1' + \Gamma_2$. By Definition 6.2.35, Definition 6.2.36, and Definition 6.2.42, $\Delta' = \Delta_1', \Delta_2$ and $\Psi' = \Psi_1' \cdot \Psi_2$ are consistent derivatives of Γ, Δ, and Ψ. Hence,

$$\dfrac{\dfrac{\cdots}{\Gamma_1';\Delta_1';\Psi_1' \vdash P_1'}IH \quad \dfrac{}{\Gamma_2;\Delta_2;\Psi_2 \vdash P_2}A}{\Gamma';\Delta';\Psi' \vdash P_1' \mid P_2}\text{T-Par}_{\mathrm{L}}$$

where IH is the induction hypothesis and A means by assumption, i.e., by the derivation above.

Case of Pi-Res$_{\mathrm{m,s,a,p}}$ **:** In this case, $P = (\nu x{:}T)\,P_1$, $P_1 \longmapsto P_2$, and $P' = (\nu x{:}T)\,P_2$ for some $x \in \mathcal{N}$, $T \in \mathbb{T}_{\mathrm{L}}$, and $P_1, P_2 \in (\mathcal{P}:\mathbb{T}_{\mathrm{L}})$. The derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with

$$\dfrac{\dfrac{\cdots}{\Gamma,\Gamma_x;\Delta_1;\Psi \vdash P_1}\cdots}{\Gamma;\Delta;\Psi \vdash (\nu x{:}T)\,P_1}R$$

where either $R = \text{T-Res-B}_{\mathrm{L}}$, $\Gamma_x = x{:}T$ or $T \in \mathbb{T}_{\mathrm{lin}}, \Gamma_x = \emptyset$, and $\Delta_1 = \Delta$ or $R = \text{T-Res-M}_{\mathrm{L}}$, $T = T_1 \triangleright T_2$, and $\Delta_1 = \Delta, x{:}T_2$ or again $\Delta_1 = \Delta$. Note that if T is a linear type then Lemma 6.2.44 implies that $x \in \mathsf{fn}(P_1)$. Hence, by the induction hypothesis, there exists $\Gamma_2 + \Gamma_2' = \Gamma$ such that $\Gamma_2, \Gamma_x;\Delta_2;\Psi_2 \vdash P_2$ and Δ_2, Ψ_2 are consistent derivatives of Γ, Γ_x, Δ_1, and Ψ. Choose $\Gamma' = \Gamma_2$. Then

$$\dfrac{\dfrac{}{\Gamma',\Gamma_x;\Delta_2;\Psi_2 \vdash P_2}\text{by the induction hypothesis}}{\Gamma';\Delta_2';\Psi_2 \vdash (\nu x{:}T)\,P_2}R$$

holds, if we find an appropriate Δ_2'. If $R = \text{T-Res-B}_{\mathrm{M}}$ then $\Delta_1 = \Delta$ and, so, $\Delta_2' = \Delta_2$. Else, if $R = \text{T-Res-M}_{\mathrm{M}}$, we have $T = T_1 \triangleright T_2$ and $\Delta_1 = \Delta, x{:}T_2$ or $\Delta_1 = \Delta$. By Definition 6.2.36, $\Delta_1 = \Delta$ implies $x \notin \mathsf{n}(\Delta_2)$. In this case $\Delta_2' = \Delta_2$ again. Else, by Definition 6.2.36, $\Delta_1 = \Delta, x{:}T_2$ implies $\Delta_2 = \Delta_1$ or $\Delta_2 = \Delta, x{:}T_2'$, where $T_2 = T_2'' \triangleright T_2'$. In both cases, we can choose $\Delta_2' = \Delta$.

Case of Pi-Cong$_{\mathrm{m,s,a,p}}$ **:** In the this case, $P \equiv Q$, $Q \longmapsto Q'$, and $Q' \equiv P'$ for some $Q, Q' \in (\mathcal{P}:\mathbb{T}_{\mathrm{M}})$. Without loss of generality let us assume that this is the only application of Pi-Cong$_{\mathrm{m,s,a,p}}$ in $P \longmapsto P'$. Let R, R' be such that $Q \equiv R$,

$Q' \equiv R'$, and neither R nor R' contains unguarded subterms guarded by a match prefix $[a = a]$. Then, by Pi-Cong$_{\mathrm{m,s,a,p}}$, also $P \equiv R$, $R \longmapsto R'$, and $R' \equiv P'$. This time, R and R' do not have a match that is not already in P or P', respectively. Moreover, by Lemma 6.2.48, $\Gamma; \Delta; \Psi \vdash P$ implies that Γ is closed for P, i.e., provides a type for each free name of P. By assumption, Γ is also closed for P'. Since the only rule of structural congruence that allows to introduce free names is the rule that introduces matches, Γ is also closed for R and R'. By Lemma 6.2.47, then $\Gamma; \Delta; \Psi \vdash P$ and $P \equiv R$ imply $\Gamma; \Delta; \Psi \vdash R$. By the induction hypothesis $\Gamma; \Delta; \Psi \vdash R$ and $R \longmapsto R'$ imply $\Gamma; \Delta'; \Psi' \vdash R'$. Finally, by Lemma 6.2.47, $\Gamma; \Delta'; \Psi' \vdash R'$ and $R' \equiv P'$ imply $\Gamma; \Delta'; \Psi' \vdash P'$.

□

It remains to show that the typed encodings $\mathcal{T}_{\mathrm{L}}^{1}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, $\mathcal{T}_{\mathrm{L}}^{2}[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$, and $\mathcal{T}_{\mathrm{L}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ are well-typed with respect to some appropriate type environments. Since $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ does not introduces free names, we can again show well-typedness with respect to well-typed source terms S and $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}} = \{\, x{:}\sharp(\mathfrak{n}_{\circ,T_S}) \mid x \in \mathsf{fn}(S) \wedge x{:}T_S \in \mathcal{T}_S \,\}$, where $\mathfrak{n}_{\circ,T_S} = \circ \triangleright \sharp\big(\uparrow^{1}(\mathfrak{l})\big) \triangleright \sharp\big(\uparrow^{1}(\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}))\big) \triangleright \sharp\big(\uparrow^{1}(T_S)\big) \triangleright \bullet$.

Lemma 6.2.51. *For all source terms $S \in \mathcal{P}_{\mathrm{s}}$ that are well-structured with respect to $\mathbb{T}_{\mathrm{s}}$ and $\mathcal{T}_S$, the encoding $\mathcal{T}_{\mathrm{L}}^{1}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ is well-typed with respect to $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}}$.*

The other two encoding functions $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ and $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ introduce free outputs on request channels. Since $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ and $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ use translated source term names only as values and introduce all other names under restriction except for the outermost occurrences of the request channels p_o and p_i, for all source terms $S \in \mathcal{P}_{\mathrm{m}}$, the encodings $\mathcal{T}_{\mathrm{L}}^{2}[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ and $\mathcal{T}_{\mathrm{L}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ are well-typed with respect to

$$\Gamma_{[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}} = \left\{\, p_o{:}\uparrow^{\omega}(\mathfrak{o}'_{\circ})\,, p_i{:}\uparrow^{\omega}(\mathfrak{i}'_{\circ}) \mid p_o, p_i \in \mathsf{fn}\Big([\![\, S \,]\!]_{\mathrm{p}}^{\mathrm{m}}\Big) \,\right\} \cup \{\, x{:}\mathfrak{v}_{\mathfrak{n}} \mid x \in \mathsf{fn}(S) \,\} \text{ and}$$

$$\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}} = \{\, p_o{:}\uparrow^{\omega}(\mathfrak{o}_{\circ})\,, p_i{:}\uparrow^{\omega}(\mathfrak{i}_{\circ}) \mid p_o, p_i \in \mathsf{fn}([\![\, S \,]\!]_{\mathrm{a}}^{\mathrm{m}}) \,\} \cup \{\, x{:}\mathfrak{v}_{\mathfrak{n}} \mid x \in \mathsf{fn}(S) \,\}\,,$$

respectively. Note that the set $\{\, p_o{:}\uparrow^{\omega}(\mathfrak{o}_{\circ})\,, p_i{:}\uparrow^{\omega}(\mathfrak{i}_{\circ}) \mid p_o, p_i \in \mathsf{fn}([\![\, S \,]\!]_{\mathrm{a}}^{\mathrm{m}}) \,\}$ is either empty (e.g. in the case of $[\![\, 0 \,]\!]_{\mathrm{a}}^{\mathrm{m}}$) or contains exactly two assignments.

Lemma 6.2.52. *The encoding $\mathcal{T}_{\mathrm{L}}^{2}[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ is well-typed with respect to $\Gamma_{[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}}$.*

Lemma 6.2.53. *The encoding $\mathcal{T}_{\mathrm{L}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ is well-typed with respect to $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}}$.*

The proofs of these three lemmata can be obtained by adapting the respective proofs in the basic type system. We present the proof for $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$, i.e., Lemma 6.2.53, in the appendix in Section A.1.3. With the subject reduction lemma we conclude that all target terms are well-typed.

Theorem 6.2.54. *All target terms of $\mathcal{T}_{\mathrm{L}}^{1}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, $\mathcal{T}_{\mathrm{L}}^{2}[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$, and $\mathcal{T}_{\mathrm{L}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$, i.e., all terms $P_1 \in (\mathcal{P}_{\mathrm{a}}{:}\mathbb{T}_{\mathrm{L}})\restriction_{\mathcal{T}_{\mathrm{L}}^{1}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}}$, $P_2 \in (\mathcal{P}_{\mathrm{p}}{:}\mathbb{T}_{\mathrm{L}})\restriction_{\mathcal{T}_{\mathrm{L}}^{2}[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}}$, and $P_3 \in (\mathcal{P}_{\mathrm{a}}^{=}{:}\mathbb{T}_{\mathrm{L}})\restriction_{\mathcal{T}_{\mathrm{L}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}}$, are modulo structural congruence well-typed with respect to $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}}$ (and well-structured source terms), $\Gamma_{[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}}$, and $\Gamma_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}}$, respectively, and $\Delta = \Psi = \emptyset$.*

Proof. By Lemmata 6.2.51, 6.2.52, and 6.2.53, all encoded source terms are well-typed. The target languages of the first two encodings does not contain a match prefix. Hence, by Lemma 6.2.50, all target terms $P_1 \in (\mathcal{P}_\mathrm{a} : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^1 [\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$ and $P_2 \in (\mathcal{P}_\mathrm{p} : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^2 [\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$ are well-typed with respect to $\Gamma_{[\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$ (and well-structured source terms) and $\Gamma_{[\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$, respectively.

We observe that in $[\![\cdot]\!]_\mathrm{a}^\mathrm{m}$ match is used only in the translation of the parallel operator on (bound) occurrences of translated source term names of type $\mathfrak{v}_\mathfrak{n}$. The side condition "modulo structural congruence" allows us to abstract from unnecessary matches, i.e., from matches that do not result from the encoding function but are introduced by the rule $[a = a]\, P \equiv P$ of structural congruence. Thus, for all target terms $P_3 \in (\mathcal{P}_\mathrm{a}^= : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^3 [\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$ without unnecessary matches the type environment $\Gamma_{[\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$ is closed. By Lemma 6.2.50, we conclude that for all P_3 there is some $P_3' \equiv P_3$ that is well-typed with respect to $\Gamma_{[\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$. □

We observe that we are able to assign types with special multiplicities to many of the introduced links. Fortunately, the special types we consider induce some nice properties. We use these properties in the following section to abbreviate some of the proofs on the correctness of the encoding functions. Since we do not want to drag along all the information necessary to identify the typed version of a target term, we explain how to obtain the type of names in untyped terms.

Definition 6.2.55 (Type of Names in Target Terms)**.** Let $P_1 \in \mathcal{P}_\mathrm{a} \restriction_{[\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$, $P_2 \in \mathcal{P}_\mathrm{p} \restriction_{[\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$, $P_3 \in \mathcal{P}_\mathrm{a}^= \restriction_{[\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$, $P_1' \in (\mathcal{P}_\mathrm{a} : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^1 [\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$, $P_2' \in (\mathcal{P}_\mathrm{p} : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^2 [\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$ and $P_3' \in (\mathcal{P}_\mathrm{a}^= : \mathbb{T}_\mathrm{L}) \restriction_{\mathcal{T}_\mathrm{L}^3 [\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$ such that, for all $i \in \{ 1, 2, 3 \}$, P_i and P_i' are free of name clashes, $P_i' = \mathcal{T}(P_i)$ for some set of type assignments $\mathcal{T}$ of names to linear types, and $\Gamma_i; \emptyset; \emptyset \vdash P_i'$, where $\Gamma_1 = \Gamma_{[\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$, $\Gamma_2 = \Gamma_{[\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$, and $\Gamma_3 = \Gamma_{[\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$.

Then, for all $i \in \{ 1, 2, 3 \}$, the *type of a name x in P_i* is $T \in \mathbb{T}_\mathrm{L}$ if $x \in \mathsf{n}(P_i)$ and either

- $x \in \mathsf{fn}(P_i)$ and $x{:}T \in \Gamma_i$, or
- $x \in \mathsf{bn}(P_i)$ and there exists some subterm P'' in P_i' such that either the typed restriction $(\nu x{:}T)\, P''$ or an input $y(x)\,.P''$ is a subterm of P_i' and in the latter case the type of y in P_i is some kind of link type carrying a value of type T.

The multiplicities 1 and + are introduced to capture unique use of links. As it turns out we need in particular uniqueness of inputs to ease the proofs of the following section.

Lemma 6.2.56 (Unique Input)**.** *Let $P \in \mathcal{P}_\mathrm{a} \restriction_{[\![\cdot]\!]_\mathrm{a}^\mathrm{s}}$, $P \in \mathcal{P}_\mathrm{p} \restriction_{[\![\cdot]\!]_\mathrm{p}^\mathrm{m}}$, or $P \in \mathcal{P}_\mathrm{a}^= \restriction_{[\![\cdot]\!]_\mathrm{a}^\mathrm{m}}$, and let $y \in \mathsf{n}(P)$, where the type of y in P is $T \in \mathbb{T}_\mathrm{L}$ and*

$$T \in \left\{ \updownarrow_1^1(T') , \updownarrow_1^+(T') , \updownarrow_+^\omega(T') \mid T' \in \mathbb{T}_\mathrm{L} \right\}.$$

Then there is at most one input on y in P.

Proof. By contradiction and an induction over the derivation of well-typedness of a term P that contains more than one input on y. The contradiction results from Definition 6.2.42 and the requirements on type environments in the typing rules of Figure 6.12. □

Similarly, $*$ is introduced to capture unique replicated inputs. $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ frequently use such links. The multiplicity ensures that, once the replicated input is unguarded, outputs are always eventually processed and are always handled in the same way.

Lemma 6.2.57 (Unique Replicated Input)**.** *Let $P \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $P \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $P \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, and let $y \in \mathsf{n}(P)$, where the type of y in P is $\updownarrow^{\omega}_{*}(T) \in \mathbb{T}_{\mathrm{L}}$. Then either y occurs only within restrictions or there is exactly one replicated input on y in P.*

Proof. By contradiction and an induction over the derivation of well-typedness of a term P that contains more than one replicated input on y or none and at least one occurrence of y in a match-prefix, or as subject of input or output, or object of output, input or replicated input. If y occurs in a match-prefix the contradiction results from the side condition of T-$\textsc{Mat}_{\mathrm{L}}$. Else, the contradiction results from Definition 6.2.42 and the requirements on type environments in the typing rules of Figure 6.12. □

Finally we prove that steps on linear links or links of type $\updownarrow^{+}_{1}(T')$, $\updownarrow^{\omega}_{*}(T')$, or $T_1 \triangleright T_2$ for some $T', T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}}$ can never be in conflict with some alternative step. Note that partial confluence allows us to define some equivalences in Section 6.3.4 that abstract from most of the target term steps. Since we prove correctness of the encodings modulo this equivalences, this significantly shortens the proofs.

Lemma 6.2.58 (Partial Confluence)**.** *Let $P \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $P \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $P \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, and let $y \in \mathsf{n}(P)$, where the type of y in P is $T \in \mathbb{T}_{\mathrm{L}}$ and*

$$T \in \left\{ \updownarrow^{1}_{1}(T') , \updownarrow^{+}_{1}(T') , \updownarrow^{\omega}_{*}(T') , T_1 \triangleright T_2 \mid T', T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}} \right\}.$$

Then each step of P on the link y is not in conflict to any other step, i.e., for all $P \longmapsto P_1$ and $P \longmapsto P_2$ such that the latter step is a step on y, there exists P' such that $P_1 \longmapsto P'$ and $P_2 \longmapsto P'$.

Proof. Note that within the considered asynchronous variants of the pi-calculus all conflicts result from alternative steps on the same subject. If the type of y is $\updownarrow^{\omega}_{*}(T')$ then the statement follows directly from Lemma 6.2.57. If the type of y is $\updownarrow^{1}_{1}(T')$ or $\updownarrow^{+}_{1}(T')$, show—similarly to the proof of Lemma 6.2.56—that there is at most one output on y in P. Then conclude by Lemma 6.2.56. Else, the lemma follows by contradiction and an induction over the derivation of well-typedness of a term P that has a conflict on an auxiliary link (Definition 5.4.1) of type $T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}}$. The contradiction results from the fact that no target term has a free name of type $T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}}$ and the requirements on the auxiliary sets Δ and Ψ in the typing rules of Figure 6.12 (and also of Figure 6.8). □

6.3. Semantic Properties

In Section 6.1 we showed that the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ are compositional and satisfy name invariance. The main goal of this section is to show the remaining criteria: operational correspondence, divergence reflection, and success sensitiveness of the general

framework as described in Section 3.3. Moreover, we want to discuss some general properties of encoding functions and how they influence the possibility of satisfying the quality criteria.

Within this section we make strongly use of the type information gained in the last section. Note that nearly all links that are introduced by the encoding functions appear under restriction. To unambiguously identify the links, i.e., to fix a particular name or a class of such names, we make use of the type information. Of course we are only interested in names that occur in the considered target term. The following definition allows us to abstract away from names that are never used as links.

Definition 6.3.1 (Used Names)**.** Let $P \in \mathcal{P}_{\mathrm{a}}$, $P \in \mathcal{P}_{\mathrm{p}}$, or $P \in \mathcal{P}_{\mathrm{a}}^{=}$ and let $x \in \mathsf{n}(P)$. Then P *uses the name* x if x does not only occur as parameter of the restriction operator within P. Moreover, let $\mathsf{un}(P)$ denote the set of names that are used in P.

Note that in the last chapter we intuitively explain what we mean with e.g. sum locks. The type information allows us to formalise this notion. Furthermore we formalise instantiations of sum locks.

Definition 6.3.2 (Sum Lock)**.** Let $T \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{s}}}$, $T \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{p}}^{\mathrm{m}}}$, or $T \in \mathcal{P}_{\mathrm{a}}^{=}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ and let $l \in \mathsf{un}(T)$ such that the type of l in T is $\mathfrak{l}$. Then l is a *sum lock* of T. Furthermore, T has a *positive instantiation* of l if T has an unguarded subterm

$$\overline{l}\langle\top\rangle \overset{\text{Def. 5.1.1}}{=} l(t,f)\,.\,\overline{t} \overset{\text{Def. 5.4.1}}{=} l(u_{\sim,l})\,.\,\big((\nu u_t)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big((\nu u_f)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,(\nu v_t)\,\overline{t}\langle v_t\rangle\big)\big)\big)\big)$$

and T has a *negative instantiation* of l if T has an unguarded subterm

$$\overline{l}\langle\bot\rangle \overset{\text{Def. 5.1.1}}{=} l(t,f)\,.\,\overline{f} \overset{\text{Def. 5.4.1}}{=} l(u_{\sim,l})\,.\,\big((\nu u_t)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big((\nu u_f)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,(\nu v_f)\,\overline{f}\langle v_f\rangle\big)\big)\big)\big)\,.$$

In these cases, we also say that l is instantiated positively or negatively in T, respectively. Note that positive and negative instantiations are clearly distinguished by the type of the link of the terminal output.

To consume an instantiation of a sum lock it suffices to perform a step on the sum lock, i.e., on a channel typed with the sum lock type, because the continuation of the input on the sum lock $l(u_{\sim,l})$ does not have the form of an instantiation on a sum lock. However, we sometimes say that an instantiation of a sum lock is *completely consumed* or *completely reduced* to denote the fact that the whole term represented by $\overline{l}\langle\top\rangle$ or $\overline{l}\langle\bot\rangle$ is reduced to $\mathsf{0}$. In these cases, we also say that the protocol associated with an instantiation of a sum lock is completed. Of course to do so also the protocol associated to the counterpart of an instantiation on a sum lock, i.e., to a test-construct $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$, has to be completed.

Section 6.3.1 analyses the different kinds of steps a target term may perform to emulate a source term step. As we discuss and show in the following sections a distinction of these steps can help to show the different criteria. Furthermore, we analyse the possibility of

intermediate states. In Section 6.3.2 we capture the intention behind the links introduced by the encoding functions within invariants. For this we formalise the purpose of the introduced links as it is done above for sum locks. Then we show how the respective class of links is used within the encodings in form of an invariant. In Section 6.3.3 we analyse how the encoding functions treat source term observables and formalise the relationship between source term observables and their correspondence in target terms within so called *translated observables*. Such considerations are very important to reason about encoding functions, because (1) they show how source and target terms are related with respect to standard source term observables and hence are very important to understand the way the encoding handles source terms and (2) it allows us to define equivalences to compare target terms with respect to their corresponding source terms. We define such equivalences and also congruences in Section 6.3.4. Of course equivalences defined with respect to translated observables are specific to the encoding function and are usually not standard equivalences. However, if no standard equivalence turns out to be suitable to prove correctness of an encoding, translated observables provide the possibility to nonetheless obtain a suitable notion of equivalence. Moreover, the connection between target terms covered in equivalences on translated observables may be of great assistance as it is the case for the presented encodings. Section 6.3.5 discusses junk. Garbage or junk is something nearly every not trivial encoding has to deal with. We discuss different forms of junk that can result from emulation attempts in encodings and how they influence the possibility of satisfying quality criteria. Note that junk in particular influences which equivalences can be used to show the quality of an encoding. Finally, in Section 6.3.6 we conclude with the proof of operational correspondence, divergence reflection, and success sensitiveness.

6.3.1. Steps and States of Target Terms

Within this section we analyse and classify the steps performed by target terms. As we can observe in the following sections, such a classification can help to prove semantic properties. In the simplest case an encoding translates each source term step into exactly one target term step. Such a one-to-one correspondence obviously significantly eases and also guides the proof of quality criteria. However, encodings that translate each source term step into exactly one target term step are extremely rare. None of the encoding functions presented or discussed in the last chapter satisfies this strict requirement. Otherwise, a single source term step is translated into a sequence of target term steps. Theoretically, an encoding function—in some settings even a good one—can translate a source term such that some of its steps are translated into an empty sequence of steps in the target term. But this case is also very rare. Thus, not surprisingly, all three encoding functions $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ translate each source term step into a not empty sequence of target term steps, as it is the usual case.

Note that such sequences of target term steps modelling a single source term step are denoted as emulation within this thesis. Difficulties arise from interleaving of emulations. Emulations are very seldom fixed sequences of consecutive steps that are always performed in the same order and even in this case there are usually interleaving between

different emulations. Note that this is not a sign of a bad design of an encoding function. Instead, parallel processes and concurrent behaviour of different parallel components belong to the fundamental concepts of all (not trivial) process calculi. An encoding that is not able to mimic the variability of behaviour that comes from concurrency can hardly be considered as good. Note that, because of compositionality, it is usually not possible that encoding functions shift all this variability into the first step of an emulation. Hence, the designers of encodings have to deal with the interleaving of target term steps. This leads to extremely large state spaces even in the case of source terms without infinite executions. A classification of target term steps can help to manage this state explosion.

An emulation usually consists of some preprocessing steps, followed by one or more steps that mimic the behaviour of the source term step, and finally there are often some postprocessing steps. Of course there is no clear border line between the steps considered as pre- or postprocessing steps and the steps doing the actual work of mimicking the source term behaviour. Their distinction rather depends on the point of view. Nonetheless, such a distinction, if carefully performed, supports the proof of semantic properties. We mainly distinguish between two classes of steps: *administrative steps* and *core steps*. Core steps of target terms with respect to an encoding function are the steps that contribute to the abstract behaviour of the target terms. Obviously, these include all steps that disable possible ways to reach success. Success sensitiveness requires that every source term and its encoding answer the test on the reachability of success in the same way. Moreover, because $\asymp$ has to respect success, operational correspondence constraints reachability of success also for derivatives of encoded source terms with respect to the source term behaviour. Apart from the reachability of success, source and target terms are often compared with respect to (at least some) standard observables of the source language. Literally, the criteria of the general framework of Gorla do not require an additional consideration of observables apart from success. Sometimes it is indeed hard to establish a correspondence between observables of fundamentally different process calculi, which is the main reason to base the consideration of the preservation and reflection of the source term behaviour on the reachability of success. However, an additional consideration of standard source term observables and their correspondence in target terms often reveals a lot of insights and intuition on the encoding function. It helps to explain connections between source terms and their encodings as well as connections between derivatives of encoded terms. Hence, such considerations can often be used to guide the proof of semantic properties. In this case, a core step is any step that changes the set of considered observables.

Note that a core step is not necessarily the step that finally unguards a term representing an observable, but can also be a preceding step that removes all ways that do not lead to this observable. Hence, core steps are steps that decide about the further behaviour of a term by removing some alternatives. Consider for example a source term $S = y.\checkmark + y.\mathsf{0} \mid \overline{y}$. It can perform two conflicting steps, one leading to success and the other one leading to $\mathsf{0}$. An encoding of this source term has to mimic this variability within the emulations of target terms. By operational completeness, both alternative steps of S have to be emulated in the encoding of S. Moreover, because $\asymp$ respects

success and by operational soundness, the two emulations of the target term are again in conflict. Hence, there is one step in each emulation that decides which of the possible source term steps is currently emulated. This is the core step. It can be seen as the point of no return illuminating the identity of the source term step emulated currently by ruling out the conflicting emulations. The S above could be a source term of all three considered encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Remember that the latter two encodings inherit the main idea of encoding choices with the help of sum locks from the first encoding. Hence, any emulation of a source term step on a τ-prefix or a communication with a replicated input consumes exactly one positive instantiation of a sum lock and any emulation of a source term communication not reducing a replicated input consumes exactly two positive instantiations of sum locks within a nested test-construct. In all three encodings, as we will show in the following sections, this consumption of positive instantiations of sum locks marks the point of no return, i.e., the core steps.

Definition 6.3.3 (Core Step)**.** Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$. A step $T_1 \longmapsto T_2$ is a *core step*, denoted by $T_1 \overset{\mapsto}{\longmapsto} T_2$, if this step reduces a positive instantiation of a sum lock within a single test-construct or within the second test of a nested test-construct.

Accordingly, administrative steps are pre- or postprocessing steps of an encoding that do not influence the set of considered observables which contain at least success. In this sense, they do not contribute to the abstract behaviour of target terms, but of course they prepare terms to perform steps that do so. In principle, for all three encodings, any step that does not reduce a positive instantiation of a sum lock is an administrative step. Moreover, we distinguish between two kinds of administrative steps within our three encoding functions. The protocol that is used to unfold polyadic communications is very strict and predictable. It does not rule out the existence of interleaving steps, but the encapsulation of the protocol ensures that its steps are not in conflict with other steps of the target term as it is shown by Lemma 6.2.58. The same holds for all steps on names that are typed by a type in $\left\{ \updownarrow^{1}_{1}(T'), \updownarrow^{+}_{1}(T'), \updownarrow^{\omega}_{*}(T'), T_1 \triangleright T_2 \mid T', T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}} \right\}$. Because of that, we introduce a special kind of administrative steps, denoted as *strict administrative steps*, capturing these kinds of steps. This allows us to ignore these steps in many situations and eases e.g. the definition of translated observables later in this chapter.

Definition 6.3.4 (Administrative Step)**.** Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$. A step $T_1 \longmapsto T_2$ is an *administrative step*, denoted by $T_1 \overset{\cdot}{\longmapsto} T_2$, if it is not a step on a translated source term name and does not reduce a positive instantiation of a sum lock.

Moreover, an administrative step $T_1 \overset{\cdot}{\longmapsto} T_2$ is a *strict administrative step*, denoted by $T_1 \overset{\cdot\cdot}{\longmapsto} T_2$, if it reduces a link $y \in \mathsf{n}(T_1)$ such that the type of y in T_1 belongs to $\left\{ \updownarrow^{1}_{1}(T'), \updownarrow^{+}_{1}(T'), \updownarrow^{\omega}_{*}(T'), T_1 \triangleright T_2 \mid T', T_1 \triangleright T_2 \in \mathbb{T}_{\mathrm{L}} \right\}$.

Let $\overset{\cdot}{\Longmapsto}$ denote the transitive and reflexive closure of $\overset{\cdot}{\longmapsto}$ and $\overset{\cdot\cdot}{\Longmapsto}$ denote the transitive and reflexive closure of $\overset{\cdot\cdot}{\longmapsto}$, respectively.

Because they do not influence the considered observables of a term, semantic criteria as success sensitiveness or even operational correspondence are often easier to prove in the case of administrative steps. The proof of operational correspondence depends on the choice of $\asymp$. In Section 6.3.4 we discuss the choice of this equivalence and show how a clear separation between administrative and core steps can guide this choice. An optimal choice leads to an equivalence that is insensitive to administrative steps, which significantly eases the proof of operational correspondence. On the other hand, the existence of core steps can be linked to the existence of source term steps which helps to prove properties like divergence reflection. Unfortunately, it is not always possible to separate that clearly between administrative and core steps. The intuition behind the distinction in administrative and core steps describes a semantic property. A core step decides on which source term step is emulated and an administrative step does not. But our formalisation of this distinction in Definitions 6.3.3 and 6.3.4 is rather structural, because it distinguishes steps by the type of the reduced link and, in the case of positive instantiations of sum locks, the type of the continuation of the reduced input. This leads to a clear separation if the syntactical difference in all situations induces the intended semantic consequence. For the three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ this holds in the case of single test-constructs, i.e., the consumption of a positive instantiation of a sum lock within a single test-construct is clearly a core step. But in the case of nested test-constructs the situation is different. Sometimes the consumption of the first positive instantiation marks the point of no return, sometimes the second does, and sometimes even both consumptions remove conflicting emulations. Such steps that are somehow in between, i.e., sometimes influence the set of considered observables of a term and sometimes do not, need special consideration for all three semantic criteria. We denote such steps as *impure steps*. An attentive reader may have observed that the consumption of a positive instantiation of a sum lock within the second test of a nested test-construct is by definition a core steps, but the consumption of a positive instantiation of a sum lock within the first test of a nested test-construct is not captured by the distinction in administrative and core steps so far. The same holds for steps on translated source term names in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. This is because, if a nested test-construct consumes first a positive and then a negative instantiation of a sum lock, then the respective emulation attempt is aborted, i.e., the corresponding reduction does not really belong to any emulation at all but can rather be considered as junk. By contrast, the consumption of a positive sum lock within the second test of a nested test-construct always leads to a successful emulation of a source term step. Accordingly, we consider the reduction of a positive instantiation of a sum lock within the second test of a nested test-construct as an impure core step and such a reduction within a single test-construct as a pure core step. By the way, this allows us to prove later that for each emulation of a source term step, there is exactly one core step. But the consumption of a positive instantiation of a sum lock within the first test of a nested test-construct is neither administrative nor core but only impure.

Note that in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ to check for a potential pair of translated communication partners a communication on the translated channel name is performed. In $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ polyadic synchronisation channels on translated source term names are used for the same purpose.

Similarly to the consumption of positive instantiations of sum locks, this might rule out alternative emulations. Because of that, we consider steps on translated source term names as impure steps of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$. Also note that in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ translated source term names are used as values only; so there are no steps on translated source term names.

Definition 6.3.5 (Impure Step)**.** Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$. A step $T_1 \longmapsto T_2$ is an *impure step*, denoted by $T_1 \overset{\cdot/\mapsto}{\longmapsto} T_2$, if it either reduces a positive instantiation of a sum lock within the first test of a nested test-construct, or is a step on a translated source term name.

Obviously, each emulation of a source term step contains at least one core step or, if a clear separation is not possible, at least one core or impure step. We show later that, for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, each successful emulation of a single source term step contains exactly one core step. But apart from that, such a successful emulation can also contain impure steps. Both core and impure steps may rule out alternative emulation attempts. Hence, a sequence of steps containing both an impure and a core step may successively rule out different alternative emulations in consecutive steps such that both steps mark a point of no return. This is sometimes necessary if the source term removes and/or adds several observables simultaneously within a single step—as it is the case for reductions of sums—and the target language or at least the target terms of the encoding are not able to mimic all these changes within a single step. We denote this situation as *partial commitment*. Gradual or partial commitments were already described e.g. in [PS92, PS94] and lead to the definition of coupled simulation (see Definition 2.2.3). Moreover, already [Nes96, Nes00, NP00] show that partial commitments are a side effect of encoding choice.

Example 6.3.6 (Partial Commitment)**.** Let us consider the source term

$$S = (a.S_1 + a.S_2) \mid \overline{a} \mid a.S_3$$

for some $S_1, S_2, S_3 \in \mathcal{P}_{\mathrm{s}}$. So $S \in \mathcal{P}_{\mathrm{s}}$ as well as $S \in \mathcal{P}_{\mathrm{m}}$. The encoding of S, regardless whether it is encoded by $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, or $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, generates three sum locks, one for each sum. Let us assume the sum lock generated for $a.S_1 + a.S_2$ is l_1, the sum lock for $\overline{a}$ is l_2, and l_3 is generated for $a.S_3$. S can perform three conflicting steps leading to S_1, S_2, or S_3, respectively. Each of these steps can be emulated by the encodings. To emulate the step to S_1 the positive instantiation of the sum lock l_1 is consumed first and to complete the emulation a positive instantiation of the sum lock l_2 has to be consumed. However, it is possible that the encoded term instead performs another nested test-construct and consumes both positive instantiations of l_2 and l_3 to emulate the step to S_3 instead. Because of that, the step on l_1 is not a core step. Even, if at this point the sum lock that is tested next by this test-construct is still instantiated positive, it can become negative by a concurrent test-construct.

So after the consumption of the positive instantiation of l_1 in order to emulate the source term step to S_1, there is still the possibility to complete instead the emulation of

the source term step to S_3. However, there is no possibility to complete the emulation of the source term step to S_2. Note that, to emulate the step to S_2, a positive instantiation of each of the locks l_1 and l_2 is necessary. But the instantiation of l_1 is already consumed. The only possibility to restore it is to consume a negative instantiation of l_2 (compare to the nested test-constructs in the encodings of input guarded terms in Figure 5.1, Figure 5.4, and Figure 5.8). As we prove later, there is no possibility to change that negative instantiation back into a positive one. So, as soon as l_1 is consumed, one of the three possible emulations, namely either the one leading to S_1 or to S_2, is ruled out while there are still two possible emulations left.

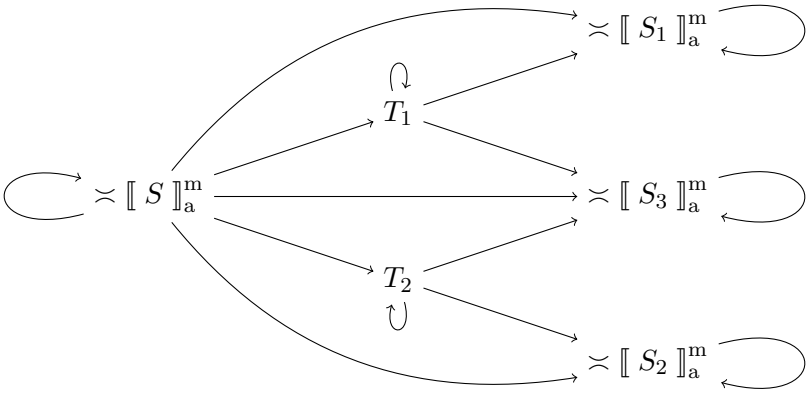


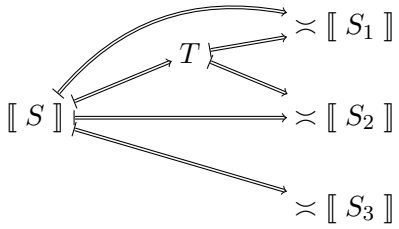
Figure 6.13.: Partially Committed States.

We visualise this phenomenon in Figure 6.13 (for the case of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$). Here, the nodes represent states of target terms, i.e., equivalence classes of target terms modulo the instantiation of $\asymp$ which is chosen later, and the edges represent target term steps. Note that administrative steps do not change the state of a target term modulo the considered equivalence, i.e., are represented by the cycles. The impure steps lead to T_1 or T_2. The other steps are core steps. Between the encoding of the source term S and the encodings of its derivatives S_1, S_2, and S_3 there are two *partially committed states* depicted by T_1 and T_2, i.e., two states that differ from each encoded source term within that picture.

Definition 6.3.7 (Partially Committed State)**.** Let the process calculi $\mathcal{L}_{\mathrm{S}} = \langle\, \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle\, \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be the source and target language of an encoding $[\![\cdot]\!]$ that is good with respect to some equivalence $\asymp \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$. Then a term $T \in \mathcal{P}_{\mathrm{T}}{\restriction_{[\![\cdot]\!]}}$ is a *partially committed state*, if there exist some $S, S_1, S_2, S_3 \in \mathcal{P}_{\mathrm{S}}$ such that

1. $S \Longmapsto_{\mathrm{S}} S_1$, $S \Longmapsto_{\mathrm{S}} S_2$, and $S \Longmapsto_{\mathrm{S}} S_3$ but there exists no $S' \in \mathcal{P}_{\mathrm{S}}$ such that $S \Longmapsto_{\mathrm{S}} S'$, $S' \Longmapsto_{\mathrm{S}} S_1$, and $S' \Longmapsto_{\mathrm{S}} S_2$; and

2. $[\![S]\!] \Longmapsto_{\mathrm{T}} T$, $T \Longmapsto_{\mathrm{T}} \asymp [\![S_1]\!]$, and $T \Longmapsto_{\mathrm{T}} \asymp [\![S_2]\!]$, but there exists no $T' \in \mathcal{P}_{\mathrm{T}}$ such that $T \Longmapsto_{\mathrm{T}} T'$ and $T' \asymp [\![S_3]\!]$.



Note that the observability of a partially committed state depends on the chosen equivalence $\asymp$ on target terms. If we for example consider the equivalence classes of target terms modulo weak reduction bisimulation then there are no partially committed states for all three encodings.

In Example 6.3.6 impure steps lead to the partially committed states. Indeed, for all three encoding functions $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, any partially committed state results from an impure step, but not every impure step leads to a partially committed state.

Remarkably, the existence of partially committed states in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ is independent of structural congruence—if two source terms are structurally congruent, then their encodings have the same partially committed states—while this is not true for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ or $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Here, the locks are always tested according to a total ordering created along the structure induced by the nesting of parallel operators of the source term. Because of that, this ordering of sum locks differ for source terms that are structurally congruent but differ in the order of their subprocesses, i.e., differ by rule $P \mid Q \equiv Q \mid P$. So, structurally congruent source terms can differ in the number and nature of reachable partially committed states; e.g. $S' = \overline{a} \mid (a.S_1 + a.S_2) \mid a.S_3$, which is structurally congruent to S from Example 6.3.6, does not reach any of the above partially committed states. Instead, in S' the first consumption of a positive instantiation of a sum lock, which is an impure step, completely determines which emulation can be completed. Note that in this case the two following core steps are semantically rather administrative steps. Interestingly, also in the alternative of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ presented in Section 5.3 the presence of partially committed states is influenced by structural congruence of source terms. Instead of commutativity, here associativity of the parallel operator, i.e., $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, is the problem, because it changes the identity of the closest common parent node of a pair of communication partners in the parallel structure used in the target term and, thus, the instance of the coordinator lock that has to be consumed in order to enable their emulation. However, separating these kinds of steps by structural properties in Definitions 6.3.3, 6.3.4, and 6.3.5 eases the argumentation in the following sections.

6.3.2. Invariants

In Chapter 5 we explain that each name that is introduced by one of the encoding functions is introduced for a particular purpose. In this section we give more intuition for the respective purposes, because we formulate and prove conditions on the use of the respective links. More precisely, we formalise the properties of different links in form of *invariants*. Invariants define some kind of properties of terms that remain valid under

reduction steps. Usually they are shown by an induction on the reductions of target terms. Note that already the type information in Section 6.2 formulates some invariants of the links, as for example that some links are never used as input channels. Moreover, Lemma 6.2.20 shows that in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ (translations of) source term names are never used as links.

Observation 6.3.8. No translated source term name is the name of a link in a target term of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$.

Thus, some of the following invariants are simple consequences of the type information from Section 6.2. Note that we can show all of the following invariants (without the type information) by an exhaustive induction on the structure of source terms and the number of steps that are necessary to reach a particular target term from an encoded source term. However, because also the type information and in particular Theorem 6.2.54 results from such inductions, we can use the type information to abbreviate the following proofs.

Already in [Nes00] invariants are used to describe the properties of sum locks:

> "On each [sum] lock, at most one message may ever be available at any time. This guarantee implements locking, which enables mutual exclusion."
>
> "Each reader of a [sum] lock eventually writes back to the lock. This obligation enables the correct abortion of non-chosen branches."

We do alike for all three encodings that use sum locks in basically the same way.

Note that for all three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, or $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ the encoded subterms of restrictions, parallel composition, and choice appear unguarded in the respective encodings of these operators. But the encoded subterms of τ, output, or (replicated) input prefixes appear guarded. Thus, the encoding of a source term part is guarded iff the source term part is guarded.

Observation 6.3.9. Let $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ be one of the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, or $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, and let $S, S' \in \mathcal{P}_{\mathrm{S}}$ be such that S' is subterm of S. Then $[\![S']\!]$ is guarded in $[\![S]\!]$ iff S' is guarded in S.

The most difficult property of sum locks is that for each consumed instantiation eventually a new instantiation of this lock is unguarded, i.e., that there are no deadlocks caused by the processing of sum locks. We postpone this condition to the end of this section, because its proof makes use of some other invariants. Now we show that the encoding of a source term sum leads to exactly one sum lock that is initially instantiated positively. We prove that each sum lock stems from the encoding of a source term sum. Moreover, all outputs on sum locks belong to a test-construct and all inputs on sum locks are possibly guarded instantiations of this lock. We show that after a step on a sum lock the reduction of the respective instantiation and test-construct can be interleaved but not interfered, i.e., is eventually completed. Furthermore, there is at most one instantiation of each sum lock and a negatively instantiated sum lock can never become positive again.

Lemma 6.3.10 (Sum Lock)**.** *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *satisfy the following invariants on sum locks for all their target terms:*

1. *In each encoded source term there is exactly one sum lock for each source term sum (and in* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *also for each replicated source term input) and one input on this lock that is unguarded iff the encoded sum (replicated input) is unguarded. If the input is unguarded it is a positive instantiation. All other inputs on sum locks are guarded. Moreover, there is exactly one output on a sum lock for each* τ*-guard or replicated input in the source term and exactly two such outputs for each source term input. Outputs that result from the translation of a* τ*-guard are unguarded iff the translation of the* τ*-guard is unguarded. All other outputs are guarded.*

2. *Each sum lock originates from the encoding of exactly one source term sum (or, in the case of the encoding* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$*, exactly one replicated source term input).*

3. *All (replicated) inputs on sum locks* l *are of the form* $\overline{l}\langle\top\rangle$ *or* $\overline{l}\langle\bot\rangle$ *and all outputs on sum locks* l *belong to a* test*-construct* test l then P else Q *for some* P, Q*.*

4. *For each step on a sum lock* l *eventually exactly one instantiation of* l *is completely consumed and exactly one* test*-construct* test l then P else Q *on* l *is reduced to its* then*-case* $P \mid (\nu f)\,(f.Q)$ *if the consumed instantiation was positive and else to its* else*-case* $Q \mid (\nu t)\,(t.P)$*.*

5. *There is at most one instantiation of each sum lock.*

6. *A negative instantiation of a sum lock can never become positive.*

Proof. By Theorem 6.2.54, all sum locks of all target terms are restricted. The first condition holds by Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53. By Lemma 6.2.47, the condition holds also for terms that are structurally congruent to encoded source terms. For the remaining conditions we perform an induction on the number of steps from an encoded source term.

Base Case: Note that all three encodings translates choice in the same way. The Conditions 2, 3, and 4 follow from Lemma 6.2.51, Lemma 6.2.52, Lemma 6.2.53, and Lemma 6.2.47, for all encoded source terms and all target terms that are structurally congruent to an encoded source term. Note that outputs on sum locks are introduced by the translations of terms guarded by τ, input, or replicated input and inputs on sum locks are introduced by the translations of sums and terms guarded by τ, input, or replicated input. The other conditions hold trivially, because there were no steps on sum locks yet.

Induction Hypothesis: All target terms of the encodings satisfy the above conditions.

Induction Step: Consider the target terms T, T' for one of the three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ such that $T \longmapsto T'$. By the induction hypothesis, the five properties hold for T.

The only possibility to introduce new restrictions and hence new sum locks is to reduce a replicated input. However, since all sum locks in T originate from the translation of exactly one source term sum (or replicated source term input) which also includes sum locks whose restriction is guarded by a replicated input, the new sum lock also originates from the translation of exactly one source term sum (or replicated source term input).

Since Condition 3 considers all inputs and outputs on an arbitrary name of type $\mathfrak{l}$ and also guarded inputs and outputs, the condition is (at least for the inputs) invariant under steps. For outputs remember that $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q \overset{\text{Def. 5.1.1}}{=} (\nu t, f)\left(\overline{l}\langle t, f\rangle \mid t.P \mid f.Q\right)$, i.e., the first step that reduces a test-construct has to be a step on a sum lock. Thus, also the property for outputs in Condition 3 is invariant under steps.

For the fourth condition we have to analyse all possibilities for steps on sum locks. By Definition 6.3.2, instantiations on sum locks are unguarded inputs on sum locks. By Condition 3, all outputs on sum locks correspond to test-constructs and all inputs on sum locks are of the form $\overline{l}\langle\top\rangle$ or $\overline{l}\langle\bot\rangle$. Remember that:

$$\begin{aligned}\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q &\overset{\text{Def. 5.1.1}}{=} (\nu t, f)\left(\overline{l}\langle t, f\rangle \mid t.P \mid f.Q\right)\\ &\overset{\text{Def. 5.4.1}}{=} (\nu t, f)\Big(\ (\nu u_{\sim,l})\left(\overline{l}\langle u_{\sim,l}\rangle \mid u_{\sim,l}(u_t)\,.\,(\overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f)\,.\overline{u_f}\langle f\rangle)\right)\\ &\qquad\qquad \mid t(v_t)\,.P \mid f(v_f)\,.Q\Big)\end{aligned}$$

Hence, for every output and input on a sum lock there is some sequence of steps in which either a positive instantiation $\overline{l}\langle\top\rangle$ is reduced to $\mathsf{0}$ and a test-construct $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is reduced to $P \mid (\nu f)\,(f.Q)$, or a negative instantiation $\overline{l}\langle\bot\rangle$ is reduced to $\mathsf{0}$ and a test-construct $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is reduced to $Q \mid (\nu t)\,(t.P)$. By Theorem 6.2.54, all steps in the respective reductions are on links that have the type $t_1 \triangleright t_2$ (for $u_{\sim,l}$), $\updownarrow_1^1(t)$ (for u_t and u_f), or $\updownarrow_1^+(t)$ (for t and f) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_\mathrm{L}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T' or its derivatives. We conclude that if $T \longmapsto T'$ is a step on a sum lock, then eventually exactly one instantiation of l is completely consumed and exactly one test-construct $\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q$ on l is reduced to its then-case $P \mid (\nu f)\,(f.Q)$ if the consumed instantiation was positive or to its else-case $Q \mid (\nu t)\,(t.P)$, otherwise.

By the induction hypotheses there is at most one instantiation of each sum lock in T and each of its predecessors. By the argumentation above, the step $T \longmapsto T'$ can only lead to new instantiations on sum locks if either the completion of the protocol on a test-construct unguards some or if the encoding of a source term sum is unguarded by the step. In the second case, by Condition 1, there is exactly one positive instantiation of a fresh sum lock. By the argumentation in the case before, to complete a test-construct on a sum lock l it is necessary to consume an instantiation of l. We observe that in all three encoding functions in Figure 5.1, Figure 5.4, and Figure 5.8, the completion of a (nested) test-construct on l (and

l') unguards exactly one instantiation of l (and l'). Since there is at most one instantiation of each sum lock in T and each of its predecessors, we conclude that there is again at most one instantiation of each sum lock in T', namely exactly one for each sum lock that is not currently processed by the protocol of a test-construct on the respective sum lock.

By the argumentation above, new instantiations of an existing sum lock are the result of resolving a (nested) test-construct. More precisely, such a (nested) test-construct on l (and l') unguards exactly one instantiation of l (and l'). In all three encoding functions in Figure 5.1, Figure 5.4, and Figure 5.8, positive instantiations can be changed into negative instantiations in this way, but negative instantiations always lead to the else-case that simply restores the consumed instantiations of the respective sum locks without changing their value.

□

Note that the remainders $(\nu f)\,(f.Q)$ and $(\nu t)\,(t.P)$ represent unobservable and inactive junk as explained in Section 6.3.5, i.e., they do no harm.

Also sender locks—second sender locks in the case of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$—are similar in all three encodings.

Definition 6.3.11 (Sender Lock)**.** Let $T \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, $T \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, or $T \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ and let $s \in \mathsf{un}(T)$ such that the type of s in T is $\mathfrak{s}$. Then s is a *sender lock* of T. Furthermore, T has an *instantiation* of s if T has an unguarded subterm

$$\overline{s} \overset{\text{Def. 5.4.1}}{=} (\nu v_s)\,\overline{s}\langle v_s\rangle\,.$$

In this case, we also say that s is instantiated in T.

Sender locks guard the encoded continuation of source term senders. They are unguarded by an instantiation of a sender lock that is provided after the successful emulation of a source term step, illuminated by the reduction of a nested test-construct to its first case or the reduction of a single test-construct to its then-case in the translation of a replicated input. Therefore sender locks are transmitted as part of output requests and outputs on receiver locks. Accordingly, if the concept of the encodings is implemented correctly within the functions $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, there is at most one step on each sender lock. Accordingly, the invariant is very simple.

Lemma 6.3.12 (Sender Lock)**.** *The encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ satisfy the following invariants on sender locks for all their target terms:*

1. *In each encoded source term there is exactly one sender lock for each source term output and exactly one input on this lock that guards the encoded continuation of the source term output. This is unguarded iff the encoded output is unguarded.*
2. *No encoded source term contains an instantiation of a sender lock.*
3. *Each sender lock originates from the encoding of exactly one source term output.*

4. *All (replicated) inputs on a sender lock are of the form* $s(v_s).P$ *for some* P *such that* $v_s \notin \mathsf{fn}(P)$ *and* P *is an encoded source term.*

5. *There is at most one input on each sender lock and no replicated input.*

Proof. By Theorem 6.2.54, all sender locks of all target terms are restricted. Condition 5 follows from Definition 6.3.11 and Lemma 6.2.56. By Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53, the first two conditions are satisfied and Condition 3 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source term. Condition 3 is invariant under steps, because it considers only the origin of restrictions and restriction can only be copied by the reduction of a replicated input but there is no possibility to introduce new restrictions.

Consider an arbitrary target term T. By Lemma 6.2.51, Lemma 6.2.52, Lemma 6.2.53, and Lemma 6.2.47, all (replicated) inputs on sender locks are of the form $s(v_s).P$, where P is the encoding of the continuation of a source term output (Condition 1) for all encoded source terms. By Figure 6.12 and because the type of s in T is $\mathfrak{s} = \updownarrow^{\omega}_{+}(\mathfrak{v}_{\mathfrak{s}})$, the type of v_s in T is $\mathfrak{v}_{\mathfrak{s}}$. The respective renaming policy ensures that v_s is different from each translated source term name. By Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53, P is well-typed with respect to a type environment that does not contain an assignment for v_s. By Lemma 6.2.48 and Definition 6.2.4, then $v_s \notin \mathsf{fn}(P)$. Again, Condition 4 is invariant under reduction steps of the target, because it considers all—guarded or unguarded—(replicated) inputs on all links—free or bound—that are, due to their type, sender locks. □

Note that the Condition 4 ensures that the auxiliary value introduced by the unfolding of polyadic communication does not introduce any further behaviour, i.e., does no harm.

Receiver locks are used differently by the three encodings, but we also find some similarities. However, we observe that, in contrast with $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ introduces two kinds of sender and receiver locks. As defined above, we denote the second kind of sender locks as sender locks, because they correspond to the sender locks used in the other encodings. Remember that we assign the same type to the first sender and receiver locks.

Definition 6.3.13 (Auxiliary Sender or Receiver Lock)**.** Let $T \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$ and let $s_1 \in \mathsf{un}(T)$ such that the type of s_1 in T is $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$. Then s_1 is a *auxiliary sender or receiver lock* of T. Furthermore, T has an *instantiation* of s_1 if T has an unguarded subterm

$$\overline{s_1} \overset{\text{Def. 5.4.1}}{=} (\nu\, v_s)\, \overline{s_1}\langle v_s \rangle\,.$$

In this case, we also say that s_1 is instantiated in T.

Auxiliary sender and receiver locks are used to retransmit output and input requests to compensate for consumed requests in aborted emulation attempts. To ensure a first emulation attempt they have to be initially instantiated.

Lemma 6.3.14 (Auxiliary Sender or Receiver Lock)**.** *The encoding* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ *satisfies the following invariants on auxiliary sender or receiver locks for all of its target terms:*

1. *In each encoded source term there is exactly one auxiliary sender or receiver lock for each source term output or input and at least one output and exactly one replicated input (that guards an output or an input request) on this lock. This output and this replicated input are guarded iff the encoded out- or input is guarded. No other output on an auxiliary sender or receiver lock is unguarded.*

2. *Each auxiliary sender or receiver lock originates from the encoding of exactly one source term output or input.*

3. *All (replicated) inputs on an auxiliary sender or receiver lock are of the form* $s_1^\star(v_{s,r}).P$ *for some* P *such that* $v_{s,r} \notin \mathsf{fn}(P)$ *and* P *is either an output or an input request.*

4. *There is exactly one replicated input on each auxiliary sender or receiver lock and no input.*

Proof. By Theorem 6.2.54, all auxiliary sender and receiver locks of all target terms are restricted. Condition 4 follows from Definition 6.3.13 and Lemma 6.2.57. By Lemma 6.2.52, the first condition is satisfied and Condition 2 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source term. Condition 2 is invariant under steps, because it considers only the origin of restrictions.

Consider an arbitrary target term T. By Lemma 6.2.52 and Lemma 6.2.47, all (replicated) inputs on an auxiliary sender or receiver lock are of the form $s_1^\star(v_{s,r}).P$, where P is either an output or an input request (Condition 1) for all encoded source terms. By Figure 6.12 and because the type of s_1 in T is $\updownarrow_*^\omega(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$, the type of $v_{s,r}$ in T is $\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. By Lemma 6.2.52, no free name of requests has this type. Hence, $v_{s,r} \notin \mathsf{fn}(P)$. □

The second receiver locks of $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ are introduced for the same purpose as the receiver locks in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. For the definition of receiver locks and their instantiations, we use again the information gained by the linear type system.

Definition 6.3.15 (Receiver Lock)**.** Let $T \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}}$ and let $r \in \mathsf{un}(T)$ such that the type of r in T is $\updownarrow_*^\omega(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$. Then r is a *receiver lock* of T. Furthermore, T has an *instantiation* of r if T has an unguarded subterm

$$\overline{r} \overset{\text{Def. 5.4.1}}{=} (\nu v_{s,r})\, \overline{r}\langle v_{s,r}\rangle\,.$$

In this case, we also say that r is instantiated in T.

Let $T \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}}$ and let $r_2 \in \mathsf{un}(T)$ such that the type of r_2 in T is $\mathfrak{r}'$. Then r_2 is a *receiver lock* of T. Furthermore, T has an *instantiation* of r_2 if T has an unguarded subterm

$$\begin{aligned}\overline{r_2}\langle l_1, l_2, l_s, s_2, z, v, w\rangle \overset{\text{Def. 5.4.1}}{=} (\nu u_{\sim,r})\,\big(&\overline{r_2}\langle u_{\sim,r}\rangle \mid u_{\sim,r}(u_l).\big(\overline{u_l}\langle l_1\rangle \mid u_{\sim,r}(u_l).\big(\overline{u_l}\langle l_2\rangle\\ &\mid u_{\sim,r}(u_l).\big(\overline{u_l}\langle l_s\rangle \mid u_{\sim,r}(u_s).\big(\overline{u_s}\langle s_2\rangle\\ &\mid u_{\sim,r}(u_n).\big(\overline{u_n}\langle z\rangle \mid u_{\sim,r}(u_{s,r}).\big(\overline{u_{s,r}}\langle v\rangle\\ &\mid u_{\sim,r}(u_{s,r}).\overline{u_{s,r}}\langle w\rangle\,\big)\big)\big)\big)\big)\big)\big)\end{aligned}$$

for some $l_1, l_2, l_s, s_2, z, v, w \in \mathcal{N}$ such that l_1, l_2, l' are sum locks of T, s_2 is a sender lock of T, z is typed as a translated source term name by $\mathfrak{v}_\mathfrak{n}$, and v, w are auxiliary sender or receiver locks. In this case, we also say that r_2 is instantiated in T.

Let $T \in \mathcal{P}_\mathrm{a}^{=}\restriction_{[\![\,\cdot\,]\!]_\mathrm{a}^\mathrm{m}}$ and let $r \in \mathsf{un}(T)$ such that the type of r in T is $\mathfrak{r}$. Then r is a *receiver lock* of T. Furthermore, T has an *instantiation* of r if T has an unguarded subterm

$$\overline{r}\langle l_1, l_2, l', s, z\rangle \overset{\text{Def. 5.4.1}}{=} (\nu\, u_{\sim,r})\,\big(\overline{r}\langle u_{\sim,r}\rangle \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\langle l_1\rangle \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\langle l_2\rangle \\ \mid u_{\sim,r}(u_l)\,.\big(\overline{u_l}\big\langle l'\big\rangle \mid u_{\sim,r}(u_s)\,.\,(\overline{u_s}\langle s\rangle \mid u_{\sim,r}(u_n)\,.\,\overline{u_n}\langle z\rangle)\,\big)\big)\big)\big)$$

for some $l_1, l_2, l', s, z \in \mathcal{N}$ such that l_1, l_2, l' are sum locks of T, s is a sender lock of T, and z is typed as a translated source term name by $\mathfrak{v}_\mathfrak{n}$. In this case, we also say that r is instantiated in T.

The different ways to instantiate a receiver lock underpin the difference of the three encodings in the use of receiver locks. In $[\![\,\cdot\,]\!]_\mathrm{a}^\mathrm{s}$ receiver locks are introduced to guard inputs on translated source term names in the encoding of source term inputs. Initially there is exactly one instantiation of the receiver lock and new instantiations are unguarded if an emulation attempt is aborted because of the test of a negative instantiation of a sum lock associated to a sender.

Lemma 6.3.16 (Receiver Locks in $[\![\,\cdot\,]\!]_\mathrm{a}^\mathrm{s}$)**.** *The encoding $[\![\,\cdot\,]\!]_\mathrm{a}^\mathrm{s}$ satisfies the following invariants on receiver locks for all of its target terms:*

1. *In each encoded source term there is exactly one receiver lock r for each source term input $y(x)\,.P$ and at least one output and exactly one replicated input (that guards the term*

$$\varphi_\mathrm{a}^\mathrm{s}(y)\big(l', s, \varphi_\mathrm{a}^\mathrm{s}(x)\big)\,.\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\ P\ ]\!]_\mathrm{a}^\mathrm{s} \\ \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\ \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\varphi_\mathrm{a}^\mathrm{s}(y)}\big\langle l', s, \varphi_\mathrm{a}^\mathrm{s}(x)\big\rangle\,\big)$$

   for two sum locks l, l' and a sender lock s) on this lock. This output and this replicated input are guarded iff the encoded input is guarded.

2. *Each receiver lock originates from the encoding of exactly one source term input.*

3. *All (replicated) inputs on receiver locks are of the form $r^\star(v_{s,r})\,.P$ for some P such that $v_{s,r} \notin \mathsf{fn}(P)$ and P is as described by Condition 1.*

4. *There is exactly one replicated input on each receiver lock and no input.*

Proof. By Theorem 6.2.54, all receiver locks of all target terms are restricted. Condition 4 follows from Definition 6.3.15 and Lemma 6.2.57. By Lemma 6.2.51, the first condition is satisfied and Condition 2 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source

term. Condition 2 is invariant under steps, because it considers only the origin of restrictions.

Consider an arbitrary target term T. By Lemma 6.2.51 and Lemma 6.2.47, all (replicated) inputs on receiver locks are of the form $r^\star(v_{s,r})\,.T_r$ such that

$$T_r = \varphi^{\mathrm{s}}_{\mathrm{a}}(y)\big(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big)\,.\mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\ \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\ \mathsf{else}\ \overline{l}\langle\bot\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\big\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big\rangle\,\big)$$

as described in Condition 1 for all encoded source terms. By Figure 6.12 and because the type of r in T is $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$, the type of $v_{s,r}$ in T is $\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. $\varphi^{\mathrm{s}}_{\mathrm{a}}$ ensures that $v_{s,r}$ is different from each translated source term name. By Lemma 6.2.51, $[\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is well-typed with respect to a type environment that does not contain an assignment for $v_{s,r}$. By Lemma 6.2.48 and Definition 6.2.4, then $v_{s,r} \notin \mathsf{fn}([\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}})$. By Lemma 6.2.51, then no free name of T_r has the type $\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. Hence, $v_{s,r} \notin \mathsf{fn}(T_r)$. □

In the encodings $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ translated source term names are used only as values and receiver locks directly guard a (nested) test-construct. Remember that $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is an encoding from π_{m} without replicated inputs into π_{p}.

Lemma 6.3.17 (Receiver Locks in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$)**.** *The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ satisfies the following invariants on receiver locks for all of its target terms:*

1. *In each encoded source term there is exactly one receiver lock r_2 for each source term input $y(x)\,.P$ and exactly one replicated input and this term is of the form*

$$r_2^\star\big(l_1, l_2, -, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(x)\,, v, w\big)\,.\mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s_2} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{p}} \\ \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{v} \\ \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{w}\big).$$

   This replicated input is guarded iff the encoded input is guarded. No output on a receiver lock is unguarded.

2. *Each receiver lock originates from the encoding of exactly one source term input.*

3. *For each step on a receiver lock r_2 eventually exactly one instantiation on r_2 of the form $\overline{r_2}\langle l_1, l_2, l_s, s_2, z, v, w\rangle$ is completely consumed and exactly one replicated input $r_2^\star(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\,.P$ is reduced to $\{\, {}^{l_1}/_{x_1}, {}^{l_2}/_{x_2}, {}^{l_s}/_{x_3}, {}^{s_2}/_{x_4}, {}^{z}/_{x_5}, {}^{v}/_{x_6}, {}^{w}/_{x_7} \,\}\, P$*

4. *There is exactly one replicated input on each receiver lock and no input.*

Proof. By Theorem 6.2.54, all receiver locks of all target terms are restricted. Condition 4 follows from Definition 6.3.15 and Lemma 6.2.57. By Lemma 6.2.52, the first condition is satisfied and Condition 2 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source

term. Condition 2 is invariant under steps, because it considers only the origin of restrictions.

Consider an arbitrary target term T. By Condition 1, Lemma 6.2.20, and Theorem 6.2.54, all outputs on receiver locks r_2 are of the form $\overline{r_2}\langle l_1, l_2, l_s, s_2, z, v, w\rangle$ and all (replicated) inputs on r_2 are replicated inputs of the form $r_2^\star(x_1, x_2, x_3, x_4, x_5, x_6, x_7)\,.P$. Hence, for every output and replicated input on a receiver lock there is some sequence of steps in which the instantiation is completely consumed and the replicated input is reduced to $\{\, {}^{l_1}/_{x_1}, {}^{l_2}/_{x_2}, {}^{l_s}/_{x_3}, {}^{s_2}/_{x_4}, {}^{z}/_{x_5}, {}^{v}/_{x_6}, {}^{w}/_{x_7} \,\} P$. By Theorem 6.2.54, all steps in the respective reduction are on links that have type $t_1 \triangleright t_2$ (for $u_{\sim,r}$) or $\updownarrow_1^1(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_{\mathrm{L}}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly one instantiation of r_2 is completely consumed and exactly one replicated input is reduced as required. □

In contrast to the other two encodings, $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ introduces receiver locks also for replicated inputs. Apart from this, the use of receiver locks in $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ and $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ differs by the number of parameters.

Lemma 6.3.18 (Receiver Locks in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$)**.** *The encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ satisfies the following invariants on receiver locks for all of its target terms:*

1. *In each encoded source term there is exactly one receiver lock r for each source term input $y(x)\,.P$ and exactly one replicated input and this term is of the form*

$$r^\star(l_1, l_2, -, s, \varphi_{\mathrm{a}}^{\mathrm{m}}(x))\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]_{\mathrm{a}}^{\mathrm{m}} \ \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \ \mathsf{else}\ \overline{l_1}\langle\bot\rangle\,).$$

   This replicated input is guarded iff the encoded input is guarded. No output on a receiver lock is unguarded.

2. *In each encoded source term there is exactly one receiver lock r for each replicated source term input $y^\star(x)\,.P$ and exactly one replicated input*

$$r^\star(-, -, l_s, s, z)\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle z\rangle\ \mathsf{else}\ \overline{l_s}\langle\bot\rangle\,,$$

   where c_{r1} is a chain lock of type $\updownarrow_*^\omega(\mathfrak{v}_{\mathfrak{n}})$ (Definition 6.3.23). This replicated input is guarded iff the encoded replicated input is guarded.

3. *Each receiver lock originates from the encoding of exactly one (replicated) source term input.*

4. *For each step on a receiver lock r eventually exactly one instantiation on r of the form $\overline{r}\langle l_1, l_2, l', s, z\rangle$ is completely consumed and exactly one replicated input $r^\star(x_1, x_2, x_3, x_4, x_5)\,.P$ is reduced to $\{\, {}^{l_1}/_{x_1}, {}^{l_2}/_{x_2}, {}^{l'}/_{x_3}, {}^{s}/_{x_4}, {}^{z}/_{x_5} \,\} P$.*

5. *There is exactly one replicated input on each receiver lock and no input.*

Proof. By Theorem 6.2.54, all receiver locks of all target terms are restricted. Condition 5 follows from Definition 6.3.15 and Lemma 6.2.57. By Lemma 6.2.53, the first two conditions are satisfied and Condition 3 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source term. Condition 3 is invariant under steps, because it considers only the origin of restrictions.

Consider an arbitrary target term T. By Condition 1, Condition 2, Lemma 6.2.20, and Theorem 6.2.54, all outputs on receiver locks r are of the form $\overline{r}\langle l_1, l_2, l', s, z\rangle$ and all (replicated) inputs on r are of the form $r^\star(x_1, x_2, x_3, x_4, x_5)\,.P$. Hence, for every output and replicated input on a receiver lock there is some sequence of steps in which the instantiation is completely consumed and the replicated input is reduced to $\{\, {}^{l_1}/_{x_1}, {}^{l_2}/_{x_2}, {}^{l'}/_{x_3}, {}^{s}/_{x_4}, {}^{z}/_{x_5} \,\} P$. By Theorem 6.2.54, all steps in the respective reduction are on links that have type $t_1 \triangleright t_2$ (for $u_{\sim,r}$) or $\updownarrow^1_1(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_{\mathrm{L}}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly one instantiation of r is completely consumed and exactly one replicated input is reduced as required. □

The encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ translates outputs and inputs on source term names into outputs and inputs on translated source term names augmented with additional parameters for sum, sender, and receiver locks.

Lemma 6.3.19. *The encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ satisfies the following invariants on out- and inputs on translated source term names for all of its target terms:*

1. *In each encoded source term for each source term output $\overline{y}\langle z\rangle$ there is exactly one output on a translated source term name, i.e., a name of type $\sharp(\mathfrak{n}_{\circ,T_S})$, and this term is of the form $\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle$ for some sum lock l and sender lock s that is introduced by the same translation as the source term output. This term is guarded iff the respective source term output is guarded.*

2. *In each encoded source term for each source term input $y(x)\,.P$ there is exactly one input and this term is as described in Condition 1 of Lemma 6.3.16. This input is guarded by a replicated input on the receiver lock that is introduced by this translation of a source term input.*

3. *In each encoded source term for each replicated source term input $y^\star(x)\,.P$ there is exactly one replicated input*

$$\varphi^{\mathrm{s}}_{\mathrm{a}}(y)^\star(l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{s}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle\,.$$

   This replicated input is guarded iff the respective replicated source term input is guarded.

4. *In each encoded source term $[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ and for all translated source term names $\varphi^{\mathrm{s}}_{\mathrm{a}}(y) \in \mathsf{n}([\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}})$, $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ is free in $[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ iff y is free in S.*

5. *Each output, input, and replicated input on a translated source term name, i.e., on a name of type* $\sharp(\mathfrak{n}_{\circ,T_S})$*, originates from the encoding of exactly one source term output, source term input, or replicated source term input, respectively.*

6. *For each step on a translated source term name* y*, i.e.,* y *is of type* $\sharp(\mathfrak{n}_{\circ,T_S})$*, eventually exactly one output* $\overline{y}\langle l, s, z\rangle$ *is reduced to* $\mathsf{0}$ *and exactly one (replicated) input* $y(x_1, x_2, x_3).P$ *or* $y^\star(x_1, x_2, x_3).P$ *is reduced to* $\{\, {}^{l}/_{x_1}, {}^{s}/_{x_2}, {}^{z}/_{x_3} \,\} P$*.*

Proof. By Lemma 6.2.53 and an induction on the structure of source terms, the first four conditions are satisfied and Condition 5 holds for all encoded source terms. By Lemma 6.2.47, they also hold for all target terms that are structurally congruent to an encoded source term. Condition 5 is invariant under steps, because it considers all outputs and (replicated) inputs on all links that are, due to their type, translated source term names.

By the Conditions 1, 2, 3, Lemma 6.2.20, and Theorem 6.2.54, all outputs on translated source term names y belong to a term of the form $\overline{y}\langle l, s, z\rangle$ and all (replicated) inputs are of the form $y(x_1, x_2, x_3).P$ or $y^\star(x_1, x_2, x_3).P$. Hence, for every output and (replicated) input on a translated source term name there is some sequence of steps in which the term $\overline{y}\langle l, s, z\rangle$ is reduced to $\mathsf{0}$ and the (replicated) input $y(x_1, x_2, x_3).P$ or $y^\star(x_1, x_2, x_3).P$ is reduced to $\{\, {}^{l}/_{x_1}, {}^{s}/_{x_2}, {}^{z}/_{x_3} \,\} P$. By Theorem 6.2.54, all steps in the respective reduction are on links that have type $t_1 \triangleright t_2$ (for $u_{\sim,T_S}$) or $\updownarrow^1_1(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_\mathrm{L}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly one term $\overline{y}\langle l, s, z\rangle$ is reduced to $\mathsf{0}$ and exactly one (replicated) input is reduced as required. □

Instead of out- and inputs on translated source term names, the encodings $[\![\cdot]\!]^\mathrm{m}_\mathrm{p}$ and $[\![\cdot]\!]^\mathrm{m}_\mathrm{a}$ introduce requests. The translations of source term names are used as values only. Requests in $[\![\cdot]\!]^\mathrm{m}_\mathrm{p}$ and $[\![\cdot]\!]^\mathrm{m}_\mathrm{a}$ differ in their number of arguments.

Definition 6.3.20 (Request)**.** Let $T \in \mathcal{P}_\mathrm{p}\restriction_{[\![\cdot]\!]^\mathrm{m}_\mathrm{p}}$ and let $p_o \in \mathsf{un}(T)$ such that the type of p_o in T is $\mathfrak{o}'$. Then p_o is an *output request channel* of T. Similarly, let $T \in \mathcal{P}^{=}_\mathrm{a}\restriction_{[\![\cdot]\!]^\mathrm{m}_\mathrm{a}}$ and let $p_i \in \mathsf{un}(T)$ such that the type of p_i in T is $\mathfrak{i}$. Then p_i is an *input request channel* of T. Furthermore, $T \in \mathcal{P}_\mathrm{p}\restriction_{[\![\cdot]\!]^\mathrm{m}_\mathrm{p}}$ has an *output request* if there exist $y, l, s_1, s_2, z \in \mathcal{N}$ such that y, z are translated source term names of type $\mathfrak{v}_\mathfrak{n}$, l is a sum lock of T, s_1 is an auxiliary sender or receiver lock of T, s_2 is a sender lock of T, and T has an unguarded subterm

$$\begin{aligned}\overline{p_o}\langle y, l, s_1, s_2, z\rangle \overset{\text{Def. } 5.4.1}{=} \big(\nu u_{\sim,o'}\big)\, \big(\overline{p_o}\langle u_{\sim,o'}\rangle \mid u_{\sim,o'}(u_n).\big(\overline{u_n}\langle y\rangle \mid u_{\sim,o'}(u_l).\big(\overline{u_l}\langle l\rangle \\ \mid u_{\sim,o'}(u_{s,r}).\big(\overline{u_{s,r}}\langle s_1\rangle \mid u_{\sim,o'}(u_s).\big(\overline{u_s}\langle s_2\rangle \\ \mid u_{\sim,o'}(u_n).\overline{u_n}\langle z\rangle\big)\big)\big)\big)\big)\end{aligned}$$

for some output request channel p_o of T. And $T \in \mathcal{P}_\mathrm{p}\restriction_{[\![\cdot]\!]^\mathrm{m}_\mathrm{p}}$ has an *input request* if there exist $y, l, r_1, r_2 \in \mathcal{N}$ such that y is of type $\mathfrak{v}_\mathfrak{n}$, l is a sum lock of T, r_1 is an auxiliary

sender or receiver lock of T, r_2 is a receiver lock of T, and T has an unguarded subterm

$$\overline{p_i}\langle y, l, r_1, r_2\rangle \overset{\text{Def. } 5.4.1}{=} \big(\nu u_{\sim,i'}\big)\big(\overline{p_i}\langle u_{\sim,i'}\rangle \mid u_{\sim,i'}(u_n).\big(\overline{u_n}\langle y\rangle \mid u_{\sim,i'}(u_l).\big(\overline{u_l}\langle l\rangle \mid u_{\sim,i'}(u_{s,r}).\big(\overline{u_{s,r}}\langle r_1\rangle \mid u_{\sim,i'}(u_{r'}).\overline{u_{r'}}\langle r_2\rangle\big)\big)\big)\big)$$

for some input request channel p_i of T. Two requests on different request channels but with the same parameters are considered as the same request.

Let $T \in \mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ and let $p_o \in \mathsf{un}(T)$ such that the type of p_o in T is $\mathfrak{o}$ or $\mathfrak{o}_*$. Then p_o is an *output request channel* of T. Similarly, let $T \in \mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ and let $p_i \in \mathsf{un}(T)$ such that the type of p_i in T is $\mathfrak{i}$ or $\mathfrak{i}_*$. Then p_i is an *input request channel* of T. Furthermore, $T \in \mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ has an *output request* if there exist $y, l, s, z \in \mathcal{N}$ such that y, z are typed as translated source term names of type $\mathfrak{v}_{\mathfrak{n}}$, l is a sum lock of T, s is a sender lock of T, and T has an unguarded subterm

$$\overline{p_o}\langle y, l, s, z\rangle \overset{\text{Def. } 5.4.1}{=} \big(\nu u_{\sim,o}\big)\big(\overline{p_o}\langle u_{\sim,o}\rangle \mid u_{\sim,o}(u_n).\big(\overline{u_n}\langle y\rangle \mid u_{\sim,o}(u_l).\big(\overline{u_l}\langle l\rangle \mid u_{\sim,o}(u_s).\big(\overline{u_s}\langle s\rangle \mid u_{\sim,o}(u_n).\overline{u_n}\langle z\rangle\big)\big)\big)\big)$$

for some output request channel p_o of T. And $T \in \mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ has an *input request* if there exist $y, l, r \in \mathcal{N}$ such that y is of type $\mathfrak{v}_{\mathfrak{n}}$, l is a sum lock of T, r is a receiver lock of T, and T has an unguarded subterm

$$\overline{p_i}\langle y, l, r\rangle \overset{\text{Def. } 5.4.1}{=} (\nu u_{\sim,i})\,(\overline{p_i}\langle u_{\sim,i}\rangle \mid u_{\sim,i}(u_n).\,(\overline{u_n}\langle y\rangle \mid u_{\sim,i}(u_l).\,(\overline{u_l}\langle l\rangle \mid u_{\sim,i}(u_r).\overline{u_r}\langle r\rangle)))$$

for some input request channel p_i of T. Two requests on different request channels but with the same parameters are considered as the same request.

Requests are introduced by the translation of guarded source terms. They fix the relationship between sender and receiver locks and their corresponding sum locks. Note that we consider here two requests that only differ by their link name but not their values as the same request. Remarkably, requests are preserved (modulo $\overset{\cdot\cdot}{\longmapsto}$-steps) by the encoding function, i.e., each derivative of a target term has all the requests of its predecessor. In $[\![\,\cdot\,]\!]_{\mathrm{p}}^{\mathrm{m}}$ requests are initially guarded by auxiliary sender or receiver locks.

Lemma 6.3.21 (Requests in $[\![\,\cdot\,]\!]_{\mathrm{p}}^{\mathrm{m}}$)**.** *Then encoding $[\![\,\cdot\,]\!]_{\mathrm{p}}^{\mathrm{m}}$ satisfies the following invariants on requests for all of its target terms:*

1. *In each encoded source term for each source term output $\overline{y}\langle z\rangle$ there is one output on an output request channel and this is of the form $\overline{p_o}\langle \varphi_{\mathrm{p}}^{\mathrm{m}}(y), l, s_1, s_2, \varphi_{\mathrm{p}}^{\mathrm{m}}(z)\rangle$, where s_1, s_2 are introduced by the same translation as the source term output. This term is guarded by a replicated input on s_1. No output on an output request channel is unguarded.*

2. *In each encoded source term for each source term input $y(x).P$ there is one output on an input request channel and this term is of the form $\overline{p_i}\langle \varphi_{\mathrm{p}}^{\mathrm{m}}(y), l, r_1, r_2\rangle$, where r_1, r_2 are introduced by the same translation as the source term input. This term is guarded by a replicated input on r_1. No output on an output request channel is unguarded.*

3. *In each encoded source term* $[\![S]\!]^{\mathrm{m}}_{\mathrm{p}}$ *and for all translated source term names* $\varphi^{\mathrm{m}}_{\mathrm{p}}(y) \in \mathsf{n}\left([\![S]\!]^{\mathrm{m}}_{\mathrm{p}}\right)$, $\varphi^{\mathrm{m}}_{\mathrm{p}}(y)$ *is free in* $[\![S]\!]^{\mathrm{m}}_{\mathrm{p}}$ *iff* y *is free in* S.

4. *Each output request and each input request originates from the encoding of exactly one source term output and exactly one source term input, respectively.*

5. *For each step on an output request channel* p_o *eventually exactly one output request of the form* $\overline{p_o}\langle y, l, s_1, s_2, z\rangle$ *is completely consumed and exactly one replicated input* ${p_o}^{\star}(x_1, x_2, x_3, x_4, x_5).P$ *reduces to* $P' = \{\, {}^{y}/_{x_1}, {}^{l}/_{x_2}, {}^{s_1}/_{x_3}, {}^{s_2}/_{x_4}, {}^{z}/_{x_5} \,\} P$ *such that* P' *contains the consumed output request on some output request channel* ${p_o}' \in \mathsf{fn}(P')$.

6. *For each step on an input request channel* p_i *eventually exactly one input request of the form* $p_i(y, l, r_1, r_2)$ *is completely consumed and exactly one replicated input* ${p_i}^{\star}(x_1, x_2, x_3, x_4).P$ *reduces to* $P' = \{\, {}^{y}/_{x_1}, {}^{l}/_{x_2}, {}^{r_1}/_{x_3}, {}^{r_2}/_{x_4} \,\} P$ *such that* P' *contains the consumed input request on some input request channel* ${p_i}' \in \mathsf{fn}(P')$.

7. *For all* $T \longmapsto T'$ *there exists some* T'' *such that* $T' \stackrel{\cdot\cdot}{\Longmapsto} T''$ *and* T'' *contains all requests of* T.

8. *There is exactly one replicated input on each output or input request channel that is not free in the respective target term. There are no inputs on request channels.*

9. *Requests are unambiguously identified by their sender/receiver locks, i.e., all requests that share the same third or fourth parameter are equal.*

Proof. By Definition 6.3.20, Theorem 6.2.54, and Lemma 6.2.57, Condition 8 is valid. By Lemma 6.2.52 and an induction on the structure of source terms, the first three conditions are satisfied, all (replicated) inputs on an output request channel p_o are modulo structural congruence equal to

$$P_o = p_o^{\star}(x_1, x_2, x_3, x_4, x_5) \,.\, \left(\overline{p'_o}\langle x_1, x_2, x_3, x_4, x_5\rangle \mid P'_o\right)$$

for some P'_o that does not contain requests and some output request $\overline{p'_o}\langle x_1, x_2, x_3, x_4, x_5\rangle$, and all (replicated) inputs on an input request channel p_i are modulo structural congruence equal to

$$P_i = p_i^{\star}(x_1, x_2, x_3, x_4) \,.\, \left(\overline{p'_i}\langle x_1, x_2, x_3, x_4\rangle \mid P'_i\right)$$

for some P'_i that does not contain requests and some input request $\overline{p'_i}\langle x_1, x_2, x_3, x_4\rangle$, for all encoded source terms. By Lemma 6.2.47, the same holds for all target terms that are structurally congruent to an encoded source term. Because the conditions on (replicated) inputs on request channel above hold for all (replicated) inputs on request channels—and not only for unguarded—the conditions are invariant under reduction steps.

Consider an arbitrary target term T. By the above argumentation, Lemma 6.2.20, and Theorem 6.2.54, all unguarded outputs on an output/input request channel are

output/input requests, all (replicated) inputs on an output request channel are as P_o, and all (replicated) inputs on an input request channel are as P_i above. Hence, for every output and (replicated) input on a request channel there is some sequence of steps that is as described by Condition 5 and Condition 6. By Theorem 6.2.54, all steps in the respective reduction are on links that have type $t_1 \triangleright t_2$ (for $u_{\sim,o'}$ and $u_{\sim,i'}$) or $\updownarrow_1^1(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_{\mathrm{L}}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly output/input request is completely reduced and exactly one (replicated) input is reduced as required by Condition 5 and Condition 6.

Condition 7 follows from Condition 5, Condition 6, Definition 5.4.1, and Definition 6.3.4.

By Lemma 6.2.52 and Lemma 6.2.47, for all encoded source terms and all target terms that are structurally congruent to an encoded source term, all requests, i.e., all unguarded outputs on request channels, originate from the encoding of exactly one source term output or input. Moreover, all guarded outputs on request channels are guarded by a replicated input on a request channel as described by P_o and P_i above. Thus, Condition 4 follows from Condition 5 and Condition 6.

By Conditions 1, 2, and 4, each request is either the copy of a former request or results from a step on an auxiliary sender or receiver lock. Hence, Condition 9 follows from Lemma 6.3.14, because there is exactly one replicated input on each auxiliary sender or receiver lock. □

In $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ input requests are also introduced by replicated inputs. Moreover, because of the processing of requests on the right hand side of the parallel operator encoding, we cannot easily prove that there is not more one a single unguarded (replicated) input for each request channel as we did in Lemma 6.3.21. Instead we have to take chain locks into account (Lemma 6.3.24). Note that the distinction on the kind of input capabilities in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ provides a clear distinction between left and right requests. Right requests are requests that interact with inputs on request channels, whereas left requests interact with replicated inputs on request channels and are the only kind of requests that are duplicated within the encoding of a parallel operator.

Lemma 6.3.22 (Requests in $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$)**.** *Then encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ satisfies the following invariants on requests for all of its target terms:*

1. *In each encoded source term for each source term output $\overline{y}\langle z\rangle$ there is one output on an output request channel and this term is of the form $\overline{p_o}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, s, \varphi_{\mathrm{a}}^{\mathrm{m}}(z)\rangle$, where s is introduced by the same translation as the source term output. This term is guarded iff the encoded source term output is guarded. No other output on an output request channel is unguarded.*

2. *In each encoded source term for each source term input $y(x).P$ there is one output on an input request channel and this term is of the form $\overline{p_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle$, where r is introduced by the same translation as the source term input. This term is guarded iff the encoded source term input is guarded.*

3. *In each encoded source term for each replicated source term input* $y^\star(x).P$ *there are two outputs on input request channels and this terms are of the form* $\overline{p_i}\langle\varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle$ *(with different subjects), where* r *is introduced by the same translation as the replicated source term input. These terms are guarded iff the encoded replicated source term input is guarded. No other output on an input request channel is unguarded.*

4. *In each encoded source term* $[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}$ *and for all translated source term names* $\varphi_{\mathrm{a}}^{\mathrm{m}}(y) \in \mathsf{n}([\![S]\!]_{\mathrm{a}}^{\mathrm{m}})$, $\varphi_{\mathrm{a}}^{\mathrm{m}}(y)$ *is free in* $[\![S]\!]_{\mathrm{a}}^{\mathrm{m}}$ *iff* y *is free in* S.

5. *Each output request and each input request originates from the encoding of exactly one source term output and exactly one (replicated) source term input, respectively.*

6. *For each step on an output request channel* p_o *eventually exactly one output request of the form* $\overline{p_o}\langle y, l, s, z\rangle$ *is completely consumed and exactly one (replicated) input* $p_o(x_1, x_2, x_3, x_4).P$ *or* $p_o{}^\star(x_1, x_2, x_3, x_4).P$ *reduces to* $P' = \{ {}^{y}/_{x_1}, {}^{l}/_{x_2}, {}^{s}/_{x_3}, {}^{z}/_{x_4} \} P$ *such that* P' *contains the consumed output request on some output request channel* ${p_o}' \in \mathsf{fn}(P')$.

7. *For each step on an input request channel* p_i *eventually exactly one input request of the form* $p_i(y, l, r)$ *is completely consumed and exactly one (replicated) input* $p_i(x_1, x_2, x_3).P$ *or* $p_i{}^\star(x_1, x_2, x_3).P$ *reduces to* $P' = \{ {}^{y}/_{x_1}, {}^{l}/_{x_2}, {}^{r}/_{x_3} \} P$ *such that* P' *contains the consumed input request on some input request channel* ${p_i}' \in \mathsf{fn}(P')$.

8. *For all* $T \longmapsto T'$ *there exists some* T'' *such that* $T' \stackrel{\cdot\cdot}{\Longmapsto} T''$ *and* T'' *contains all requests of* T.

9. *Requests are unambiguously identified by their sender/receiver locks, i.e., all requests that share the same third parameter are equal.*

Proof. By Lemma 6.2.52 and an induction on the structure of source terms, the first four conditions are satisfied and all guarded outputs on a request channel are of the form $\overline{y_1}\langle z_1, z_2, z_3, z_4\rangle$ (for output request channels) and $\overline{y_2}\langle z_1, z_2, z_3\rangle$ (for input request channels) and these terms are guarded by a (replicated) input on a request channel $y_3(x_1, x_2, x_3, x_4).P$ or $y_3^\star(x_1, x_2, x_3, x_4).P$ (for output request channels) and (for output request channels) $y_4(x_1, x_2, x_3).P$ or $y_4^\star(x_1, x_2, x_3).P$ such that the respective request is unguarded in P. Moreover, all (replicated) inputs on a request channel guard at least one request on the same parameters.

Consider an arbitrary target term T. By the above argumentation, Lemma 6.2.20, and Theorem 6.2.54, all unguarded outputs on an output/input request channel are an output/input request and all (replicated) inputs on request channels are as described above. Hence, for every output and (replicated) input on a request channel there is some sequence of steps that is as described by Condition 6 and Condition 7. By Theorem 6.2.54, all steps in the respective reduction are on links that have type $t_1 \triangleright t_2$ (for $u_{\sim,o'}$ and $u_{\sim,i'}$) or $\updownarrow_1^1(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_{\mathrm{L}}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly output/input request is completely

reduced and exactly one (replicated) input is reduced as required by Condition 6 and Condition 7.

Condition 8 follows from Condition 6, Condition 7, Definition 5.4.1, and Definition 6.3.4.

By Lemma 6.2.53 and Lemma 6.2.47, for all encoded source terms and all target terms that are structurally congruent to an encoded source term, all requests, i.e., all unguarded outputs on request channels, originate from the encoding of exactly one source term output or (replicated) input. Moreover, all guarded outputs on request channels are guarded by a replicated input on a request channel as described above. Thus, Condition 5 follows from Condition 6 and Condition 7.

By Conditions 1, 2, 3, and 5, each request is either the copy of a former request or results from the encoding of a source term output or (replicated) source term input. Hence, Condition 9 is satisfied. □

$[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ translates source term observables into requests, which are then combined to search for potential communication partners. In order to avoid divergence, requests cannot be copied arbitrarily often. To ensure that indeed each left request is combined with each possible matching right request, the right requests—in the encoding of a parallel operator as well as in the encoding of a replicated input—are linked within some kind of chain or list, along which the left requests are forwarded. Again to avoid divergence these chains cannot be infinitely long, so the links c_o and c_i are introduced by the encoding function to extend these chain or list by a new right request as soon as its last place is occupied. We denote these links as *chain locks*. Similarly, the chain lock c_{r1} in the encoding of a replicated input is used to establish some kind of chain on encoded source terms—the encoded continuations of that replicated input—instead of right requests.

Definition 6.3.23 (Chain Lock)**.** Let $T \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ and let $c \in \mathsf{un}(T)$ such that the type of c in T is $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$, or $\sharp(\mathfrak{c}_{\circ})$. Then, c is a *chain lock* of T. Furthermore, if the type of c in T is $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, or $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$, then T has an *instantiation* on a chain lock c if T has an unguard output on c. Else, if the type of c in T is $\sharp(\mathfrak{c}_{\circ})$, T has an *instantiation* on a chain lock c if T has an unguard subterm

$$\overline{c}\langle r_o, r_i\rangle \overset{\text{Def. 5.4.1}}{=} (\nu u_{\sim,c})\,(\overline{c}\langle u_{\sim,c}\rangle \mid u_{\sim,c}(u_o)\,.\,(\overline{u_o}\langle r_o\rangle \mid u_{\sim,c}(u_i)\,.\overline{u_i}\langle r_i\rangle))$$

for some output request channel r_o and some input request channel r_i of T. If T has a chain lock c such that the type of c in T is t we say that T has a chain lock c of type t.

To reason about steps on chain locks we fix within the following invariant their use in the encoding function. A very important property is the last condition. Since chain locks are the only links that—by Theorem 6.2.54—are allowed to transmit request channels, the last conditions is necessary to ensure that the encoding can maintain the parallel structure of the source term. Intuitively, this parallel structure is reflected by the restrictions on request channels. To preserve this structure chain locks transmit only fresh request channels.

Lemma 6.3.24 (Chain Lock)**.** *The encoding* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *satisfies the following invariants on chain locks for all of its target terms:*

1. *In each encoded source term there is exactly one chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *and exactly one chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ *for each source term parallel operator or replicated source term input, and exactly two outputs and exactly one replicated input of these locks that guard exactly one input on a request channel. One output and the replicated input of each of these locks that result from the translation of a parallel operator encoding are guarded iff the respective encoding of the parallel operator is guarded. The respective other output is guarded by an input on a request channel. In the other case there is an additional guard on an input of a chain lock of type* $\sharp(\mathfrak{o},\mathfrak{i})$ *for the respective outputs and the replicated input.*

2. *In each encoded source term each input on an output request channel and each input on an input request channel is guarded by a replicated chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *or* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$*, respectively.*

3. *In each encoded source term there is exactly one chain lock* c_{r1} *of type* $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ *and exactly one chain lock* c_{r_2} *of type* $\sharp(\mathfrak{o},\mathfrak{i})$ *for each replicated source term input. Moreover, there is exactly one output on* c_{r1} *that is guarded as described in Condition 2 of Lemma 6.3.18, exactly two outputs on* c_{r2} *that are of the form* $\overline{c_{r2}}\langle r_o, r_i\rangle$*, exactly one replicated input on* c_{r1}*, and exactly one input on* c_{r2} *that is guarded by the replicated input on* c_{r1}*. One output on* c_{r2} *and the replicated input on* c_{r1} *are guarded iff the encoded replicated source term input is guarded. The other output on* c_{r2} *is guarded by the input on this lock.*

4. *Each chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *or* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ *originates from the encoding of exactly one source term parallel operator or exactly one replicated source term. Each other chain lock originates from the encoding of exactly one replicated source term.*

5. *For each step on a chain lock of type* $\sharp(\mathfrak{v}_{\mathfrak{n}})$ *exactly one pair of chain locks of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *and* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ *as described in Condition 1 is introduced. Moreover, eventually exactly one step on a chain lock of type* $\sharp(\mathfrak{o},\mathfrak{i})$ *is performed as described in Condition 6.*

6. *For each step on a chain lock* c_{r2} *of type* $\sharp(\mathfrak{o},\mathfrak{i})$ *eventually exactly one instantiation on* c_{r2} *of the form* $\overline{c_{r2}}\langle r_o, r_i\rangle$ *is completely consumed and exactly one input* $c_{r2}(x_1, x_2).P$ *is reduced to* $\{\, {}^{r_o}/_{x_1}, {}^{r_i}/_{x_2} \,\}\, P$ *such that* P *contains exactly one instantiation on* c_{r2} *and unguards exactly one output and exactly one replicated input of a chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *and exactly one output and exactly one replicated input of a chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ *that are introduced by the step on a chain lock of type* $\sharp(\mathfrak{v}_{\mathfrak{n}})$ *that unguards the reduced input on* c_{r2}*.*

7. *For each step that reduces an input on an output request channel eventually exactly one instantiation on a chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ *is unguarded. For each step that*

reduces an input on an input request channel eventually exactly one instantiation on a chain lock of type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ *is unguarded.*

8. *There is exactly one replicated input on each chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, *and* $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ *and no other input.*

9. *There is at most one instantiation of each chain lock of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, *or* $\sharp(\mathfrak{c}_{\circ})$.

10. *Within each execution of a target term no two instantiations on chain locks of type* $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, *or* $\sharp(\mathfrak{c}_{\circ})$ *carry the same request channel.*

Proof. By Theorem 6.2.54 all chain locks are restricted. Condition 8 follows from Lemma 6.2.57. By Lemma 6.2.53, the first three conditions are satisfied and Condition 4 holds for all encoded source terms. By Lemma 6.2.47, these conditions hold also for all target terms that are structurally congruent to an encoded source term. Condition 4 is invariant under steps, because it considers only the origin of restrictions. Moreover, we observe in the case of replicated source terms the resulting restrictions on chain locks of types $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ and $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ are guarded by a replicated input on the chain lock of type $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ followed by an input on a chain lock of type $\sharp(\mathfrak{o},\mathfrak{i})$ that is introduced by this translation.

Consider an arbitrary target term T. By the above argumentation, Conditions 1 and 3, Lemma 6.2.17, and Theorem 6.2.53, all outputs on chain locks c_{r2} of type $\sharp(\mathfrak{o},\mathfrak{i})$ are of the form $\overline{c_{r2}}\langle r_o, r_i\rangle$ and all (replicated) inputs on this lock are structurally equivalent to $c_{r2}(x_1,x_2).((\nu c_o, c_i, \tilde{z})\,(P_1 \mid (\nu r_o, r_i)\,(P_2 \mid \overline{c_{r2}}\langle r_o, r_i\rangle)))$ such that c_o is of type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$, c_i is of type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$, $c_o, c_i, c_{r2} \notin \mathsf{n}(P_2)$, and P_1 contains exactly two outputs of which exactly one is unguarded and exactly one unguarded replicated input of c_o and c_i, respectively, and there are no other chain locks or out/inputs on chain locks in P_1. Hence, for every step on a chain lock of type $\sharp(\mathfrak{o},\mathfrak{i})$ there is a sequence of steps that is as required in Condition 6. By Theorem 6.2.54, all steps in the respective reduction are on links that have the type $t_1 \triangleright t_2$ (for $u_{\sim,c}$) or $\updownarrow^{1}_{1}(t)$ (for the remaining auxiliary links) in T, for some $t_1 \triangleright t_2, t \in \mathbb{T}_{\mathrm{L}}$. So, by Lemma 6.2.58, these steps cannot be in conflict with any other step of T or its derivatives. We conclude that eventually exactly one instantiation on c_{r2} is completely consumed and exactly one input on this lock is reduced as required by Condition 6.

Condition 5 then follows from Conditions 3 and 6.

Condition 7 follows from Lemma 6.3.21 and Condition 1.

Condition 9 follows from Conditions 1, 3, 8, Lemma 6.2.47, and an induction on the number of steps of a target term from an encoded source term, because to unguard a new instantiation a former one has to be consumed.

Condition 10 follows from Lemma 6.2.53, because all—guarded or unguarded—outputs on the respective locks appear unguarded under a restriction of its parameter(s) and no other output on the same lock appears unguarded under the same restriction(s). □

Note that if the consumption of a chain lock of type $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ or $\sharp(\mathfrak{o},\mathfrak{i})$ is lazy, a fast emulation of a later step on the respective encoded replicated input can overtake a

former one such that there are two instantiations of the former lock or two inputs on the latter lock. By Lemma 6.2.58, the former case causes no problem. In the latter case, the corresponding branches are not necessarily ordered within the chain induced by encodedContinuations in the order matching to the respective emulations. However, since eventually all branches are added to the chain and within the chain all requests are transmitted eventually to all branches, this causes no problem.

Finally we compose some of the above introduced concepts and show how they interact in emulations. The first two conditions of the following invariant show that for each emulation there is a core step and that each core step belongs to the emulation of a source term step. Conditions 3, 4, and 5 show how requests are combined to unguard test-constructs. Condition 7 allows us to abstract from administrative steps in most of the following proofs. Condition 8 and 10 prove that the encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ do not introduce deadlock.

Lemma 6.3.25. *The encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ satisfy the following invariants for all of their target terms:*

1. *Each execution that unguards a term that originates from the encoding of a subterm of a source term contains at least one core step.*
2. *Each execution $T \Longmapsto \overset{\mapsto}{\longmapsto} \Longmapsto T'$ unguards at least one term that originates from the encoding of a subterm of a source term.*
3. *In $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ the parallel structure of source terms is maintained by the restrictions on request channels.*
4. *Each instantiation of a receiver lock r in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is of the form $\overline{r_2}\langle l_1, l_2, l_3, s_2, z, v, w\rangle$ such that modulo $\overset{\cdot\cdot}{\longmapsto}$-steps either $l_1 = l_3$ and there is a left output request of the form $\overline{p_o}\langle y, l_1, s_2, v, z\rangle$ and a right input request $\overline{p_i}\langle y, l_2, w, r_2\rangle$, or $l_2 = l_3$ and there is a left input request $\overline{p_i}\langle y, l_1, v, r_2\rangle$ and a right output request $\overline{p_o}\langle y, l_2, s_2, w, z\rangle$.*
5. *Each instantiation of a receiver lock r in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is of the form $\overline{r}\langle l_1, l_2, l_3, s_2, z\rangle$ such that modulo $\overset{\cdot\cdot}{\longmapsto}$-steps either $l_1 = l_3$ and there is a left output request $\overline{p_o}\langle y, l_1, s_2, z\rangle$ and a right input request $\overline{p_i}\langle y, l_2, r_2\rangle$, or $l_2 = l_3$ and there is a left input request $\overline{p_i}\langle y, l_1, r_2\rangle$ and a right output request $\overline{p_o}\langle y, l_2, s_2, z\rangle$.*
6. *In each execution there is at most one instantiation of each sender lock.*
7. *Let $T \in \mathcal{P}_{\mathrm{a}}{\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}}$ or $T \in \mathcal{P}_{\mathrm{p}}{\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}}$. Then no administrative step of T is in conflict with any alternative step of T. Let $T \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}}$. Then no administrative step of T that does not reduce a right request of some parallel operator encoding or an instantiation of a chain lock of type $\sharp(\mathtt{o}, \mathtt{i})$ is in conflict with any alternative step of T.*
8. *Administrative steps do not introduce deadlock.*
9. *Administrative steps do not influence reachability of success.*

10. For each step on a sum lock l there is eventually an instantiation of l.

Proof. By Figures 5.1, 5.4, and 5.8, all three encodings ensure that a subterm of a source term is guarded in the encoding iff it is guarded in the source term. Moreover, the encodings of subterms that are guarded by τ, input, or replicated input in the source appear within the then-case of a single test-construct or the first case of a nested test-construct and the encodings of subterms that are guarded by an output on the source are guarded in the encoding by an input on a sender lock. By Lemma 6.3.12 initially no sender lock is instantiated. Moreover, by Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53, all inputs on sender locks appear within the then-case of a single test-construct or the first case of a nested test-construct. By Lemma 6.3.10, each execution that reduces test l then P else Q to $P \mid (\nu f)\,(f.Q)$ completely consumes a positive instantiation of l. Hence, Condition 1 and Condition 2 follow from Definition 6.3.3.

By Lemma 6.2.52 and Lemma 6.2.53, for all encoded source terms the parallel structure is maintained by the encoding by the restriction of left or right request channels in the encoding of the parallel operator. In $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ request channels are never transmitted, i.e., the structure induced by restrictions on request channels is not changed during reductions. In $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ request channels are transmitted by reductions on chain locks but as shown in the proof of Lemma 6.3.24 only fresh request channels are transmitted, i.e., new branches that result from emulations of steps on replicated source term inputs are added and new members are added to the chain of left or right requests in procRightOutReq and procRightInReq. This proves Condition 3.

By Lemma 6.2.52, all outputs on receiver locks originate from the encoding of a parallel operator. Remember that

$$
\begin{aligned}
\mathsf{procLeftOutReq} &\triangleq p_o{}^{\star}(y, l, s_1, s_2, z)\,.\,(\overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle)\\
\mathsf{procLeftOutReq} &\triangleq p_i{}^{\star}(y, l, r_1, r_2)\,.\,\big(\overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle\big)\\
\mathsf{procRightOutReq} &\triangleq p_o{}^{\star}(y, l_s, s_1, s_2, z)\,.\\
&\qquad (y \cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle \mid \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle)\\
\mathsf{procRightInReq} &\triangleq p_i{}^{\star}(y, l_r, r_1, r_2)\,.\\
&\qquad (y \cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle \mid \overline{p_{i,up}}\langle y, l_r, r_1, r_2\rangle)
\end{aligned}
$$

and, by Condition 3, procLeftOutReq and procLeftOutReq consume only left requests while procRightOutReq and procRightInReq consume only right requests. Moreover, note that the polyadic links $y \cdot o$ and $y \cdot i$ are restricted for the respective parallel operator encoding. By Definition 6.3.15, an instantiation of a receiver lock is an unguarded output of the form $\overline{r_2}\langle z_1, z_2, z_3, z_4, z_5, z_6, z_7\rangle$. Hence, to unguard such an instantiation an input on $y \cdot i$ or $y \cdot o$ has to be consumed that requires in turn the consumption of the requests that are required by Condition 4. Note that by Lemma 6.2.52 and Lemma 6.2.58, we can ignore the unfoldings of polyadic communications. We conclude by Lemma 6.3.21, because requests are preserved modulo $\overset{\cdot\cdot}{\longmapsto}$-steps.

By Lemma 6.2.53, all outputs on receiver locks originate from procRightOutReq and procRightInReq in the encoding of a parallel operator or replicated input. Remember

that

$$\begin{aligned}
\mathsf{procRightOutReq} \triangleq\ & \overline{c_o}\langle m_i\rangle \mid c_o{}^{\star}(m_i)\,.p_o(y, l_s, s, z)\,.\big(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \\
& \mid (\nu m_{i,up})\,\big(\, m_i{}^{\star}\big(y', l_r, r\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big) \\
& \qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\langle m_i\rangle)\,)\big) \\
\mathsf{procRightInReq} \triangleq\ & \overline{c_i}\langle m_o\rangle \mid c_i{}^{\star}(m_o)\,.p_i(y, l_r, r)\,.\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \\
& \mid (\nu m_{o,up})\,\big(\, m_o{}^{\star}\big(y', l_s, s, z\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_r, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\
& \qquad \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,)\big)
\end{aligned}$$

and, by Condition 3, the inputs on requests channels refer to right requests, where the not restricted m_o and m_i above refer to left requests. By the processing of right output requests a chain is constructed whose elements are all of the form

$$\begin{aligned}
& (\nu m_{i,up})\,\big(\, m_i{}^{\star}\big(y', l_r, r\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big) \\
& \qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \overline{c_o}\langle m_i\rangle)\,\big)
\end{aligned}$$

Different members differ only by the free names $m_{i,up}$, y, l_s, s, z of which all but $m_{i,up}$ result from the consumed right output request. By Lemma 6.3.24, all members of the chain are linked by $m_{i,up}$ over that only left requests are transmitted and the consumption of a right request does not prevent the consumption of another right request but only temporary blocks it. We conclude that the order of right output requests in this chain does not matter for the possibility to unguard a particular instantiation of a receiver lock. The argumentation for right input requests is similar. Hence, to unguard an instantiation $\overline{r}\langle l_1, l_2, l_3, s, z\rangle$ of a receiver lock r a right and a left request have to be consumed and are as required by Condition 5. We conclude by Lemma 6.3.22, because requests are preserved modulo $\overset{\cdot\cdot}{\longmapsto}$-steps.

By Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53, the then-case of a single test-construct or the first case of a nested test-construct instantiated all tested sum locks with $\bot$. By Lemma 6.3.21 and Lemma 6.3.18, all output requests that share the same sender lock also share the same sum lock. In the case of $[\![\,\cdot\,]\!]^{\mathrm{s}}_{\mathrm{a}}$, by Lemma 6.2.51 and Lemma 6.3.19, each test-construct that contains an output on a sender lock is guarded by an input or a replicated input on a translated source term name and to unguard the output on the sender lock the sum lock that is transmitted over this translated source term name has to be instantiated by $\bot$. Note that, by Lemma 6.3.12 and Lemma 6.3.19, all outputs on translated source term names that share the same sender lock as parameter also share the same sum lock. In the case of $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$, by Lemma 6.3.17, Lemma 6.3.18, and Conditions 4 and 5, all (nested) test-constructs that contain an output on a sender lock s are guarded by a receiver lock and, to unguard the output s, a sum lock l has to be instantiated by $\bot$ such that in all output requests that contain s also l is contained. Condition 3 then follows from Lemma 6.3.10, because no negative instantiation of a sum lock can become positive again.

By Lemma 6.3.12 and Condition 6, there is at most one input and at most one instantiation on each sender lock in T, i.e., steps on sender locks cannot be in conflict with any alternative step of T. Then Condition 7 follows from Definition 6.3.4, Lemma 6.2.58,

and the types of the remaining links that are not translated source term names, not sender locks, not sum locks, and for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ also right request channels, or chain locks of type $\sharp(\mathfrak{o}, \mathfrak{i})$.

Condition 8 follows for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ from Condition 7. The administrative steps of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ that are not covered by Condition 7 influence only the order in which right requests or new branches of encoded replicated inputs are added to the respective chain. By a similar argumentation as above, we can show that also all unguarded encodings of continuations of encoded replicated source term inputs are eventually added as branches to the latter chain and that the order of branches in this chain does not matter. Hence, no administrative step can influence which impure or core steps can be reached which proves Condition 8.

By Figures 5.1, 5.4, and 5.8, ✓ is translated into ✓ and there are no other occurrences of ✓ in the encoding functions, i.e., an encoded source term contains ✓ iff the source term contains ✓ and for each ✓ in the source term the ✓ in the encoding is guarded iff it is guarded in the source term. Hence, by the above conditions and the above argument, $T_1 \stackrel{\cdot}{\Longrightarrow} T_2$ implies $(T_1 \Downarrow_{\checkmark}$ iff $T_2 \Downarrow_{\checkmark})$ for all target terms T_1, T_2.

By the argumentation for Condition 3 and 4 of Lemma 6.3.10, instantiations on sum locks are consumed only by test-constructs and the completion of a (nested) test-construct on l (and l') unguards exactly one instantiation of l (and l'). Hence, the only possibility that a consumed instantiation of a sum lock is never restored is that there is a deadlock caused by a nested test-construct because of a missing instantiation of a sum lock (see the discussion in Section 5.2.1). For $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ Condition 10 follows from the proof in [Nes00] that there are no such deadlocks in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. Note the slightly changes in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ as presented here and in [Nes00] do not influences reachability of such deadlocks. Moreover, [Nes00] already points out that a total order on sum locks ensures that the same holds also in the case of mixed choice. Hence, Condition 10 follows for the other encoding from the above conditions, because the test of the left most sum lock and Condition 3 lead to such a total ordering. □

6.3.3. Translated Observables

As already stated in Section 6.3.1, the analysis of source term observables (apart from ✓) and their correspondence in target terms can help to describe the relationship between source terms and their encodings as well as between source terms or encoded source terms and derivatives of encoded source terms. By doing so we gain more insights in the operating principles of the encodings and, moreover, ease their proof of correctness. The consideration of source term observables and their correspondence in target terms in particular guides our choice of $\asymp$ for the three encodings in the next section.

In the simplest case a source term observable μ is translated into an observable $\varphi_{[\![]\!]}(\mu)$, where $\varphi_{[\![]\!]}$ is the renaming policy of the considered encoding function. Of course this requires that both the source and the target language have basically the same kind of observables, which is usually the case only between close members of the same family of process calculi. Note that not even π_{s} and π_{a} satisfy this requirement, because usually in the synchronous π_{s} input as well as output observables are considered whereas for its

asynchronous fragment π_{a} usually only output observables are considered. However, if such a strict correspondence is achieved it may be even possible to compare source and target terms directly by some standard equivalence.

A more liberal case is the translation of standard source term observables into standard target term observables. This case usually allows to choose $\asymp$ as a standard equivalence of the target language. Unfortunately, for none of the three encoding functions $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ considered here the situation is that easy. As already explained in [Nes00], the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ translates source term output barbs y into an output on $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ whose first parameter is a sum lock that has to be instantiated positively, i.e., $S \downarrow_{\overline{y}}$ iff $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv (\nu l, s, \tilde{x}) \left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, z\rangle \mid \overline{l}\langle \top \rangle \mid T \right)$ for some $l, s, z \in \mathcal{N}$, $\tilde{x} \in \mathcal{S}(\mathcal{N})$, and $T \in \mathcal{P}_{\mathrm{a}}$ such that y is not a name of $\tilde{x}$. Source term input barbs are translated similarly, but are not considered in [Nes00]. Hence, in comparison to standard observables of π_{s} or π_{a}, we have to add a requirement on the current instantiation of the corresponding sum lock. We capture the translation of source term observables explicitly within so called *translated observables*. The concept of translated observables is in principle equivalent to the Σ-Barbs introduced in [Nes96, Nes00], but in contrast to [Nes00] we define the translation of output as well as input observables. We do so, because our aim is not only to define a suitable equivalence for the target language but also to capture the relationship between source and target terms in order to guide to proof of correctness. For this it seems reasonable to consider all standard observables of the source language.

Note that within $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ inputs on the translated source term names are guarded by a receiver lock. Similarly, after a core step the translated observables of the encoded source term continuations that are unguarded by the corresponding source term step are still guarded by a step on a sender lock. Moreover, because of the unfolding of polyadic communication in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, the requests that define translated observables for these encodings can vanish temporarily, e.g. they are temporarily consumed by the forwarders in $\mathsf{procLeftOutReq}$ or $\mathsf{procLeftInReq}$. Because of that, we define translated observables modulo the administrative steps.

Definition 6.3.26 (Translated Observables in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$)**.** Let $T \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$. Then T has a *translated output observables* y, denoted by $T \downarrow^{1}_{\overline{y}}$, if T has modulo $\stackrel{\cdot}{\longmapsto}$-steps an output on $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ whose first parameter is a sum lock l such that l is instantiated positively in T and $\varphi^{\mathrm{s}}_{\mathrm{a}}(y) \in \mathsf{fn}(T)$.

T has a *translated input observables* y, denoted by $T \downarrow^{1}_{y}$, if T has modulo $\stackrel{\cdot}{\longmapsto}$-steps an input on $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ followed by a test on a sum lock l such that l is instantiated positively in T and $\varphi^{\mathrm{m}}_{\mathrm{p}}(y) \in \mathsf{fn}(T)$, or if T has modulo $\stackrel{\cdot}{\longmapsto}$-steps a replicated input on $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ and $\varphi^{\mathrm{s}}_{\mathrm{a}}(y) \in \mathsf{fn}(T)$.

Moreover, T *reaches* some translated output (input) observable, denoted as $T \Downarrow^{1}_{\mu}$, if there exists some $T' \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ such that $T \Longmapsto T'$ and $T' \downarrow^{1}_{\mu}$.

The condition $y \notin \tilde{x}$ is necessary to rule out translated observables that corresponds to invisible in- or outputs of the source term (see Definition 2.2.6).

The encoding functions $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ translate source term observables into a request again augmented with the information covered by the sum lock.

Definition 6.3.27 (Translated Observables in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$)**.** Let $T \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$. Then T has a *translated output (input) observable* y, denoted by $T \downarrow^2_{\overline{y}}$ ($T \downarrow^2_y$), if T has modulo $\overset{\cdot}{\longmapsto}$-steps an output (input) request whose first parameter is $\varphi^{\mathrm{m}}_{\mathrm{p}}(y)$ and whose second parameter is a sum lock l such that l is instantiated positively in T and $\varphi^{\mathrm{m}}_{\mathrm{p}}(y) \in \mathsf{fn}(T)$.

Moreover, T *reaches* some translated output (input) observable, denoted as $T \Downarrow^2_\mu$, if there exists some $T' \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$ such that $T \Longmapsto T'$ and $T' \downarrow^2_\mu$.

The definition of translated observables for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is similar, but remember that the definition of requests differ in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ in the number of parameters.

Definition 6.3.28 (Translated Observables in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$)**.** Let $T \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$. Then T has a *translated output (input) observable* y, denoted by $T \downarrow^3_{\overline{y}}$ ($T \downarrow^3_y$), if T has modulo $\overset{\cdot}{\longmapsto}$-steps an output (input) request whose first parameter is $\varphi^{\mathrm{m}}_{\mathrm{a}}(y)$ and whose second parameter is a sum lock l such that l is instantiated positively in T and $\varphi^{\mathrm{m}}_{\mathrm{a}}(y) \in \mathsf{fn}(T)$.

Moreover, T *reaches* some translated output (input) observable, denoted as $T \Downarrow^3_\mu$, if there exists some $T' \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ such that $T \Longmapsto T'$ and $T' \downarrow^3_\mu$.

Obviously, by the above definitions, translated observables are preserved by administrative steps. However note that administrative steps can neither change sum locks nor reduce an out- or input on a translated source term name in $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. Moreover, by Lemmata 6.3.21 and 6.3.22, also requests are preserved modulo $\overset{\cdot}{\longmapsto}$-steps. Hence, only the successful emulation of a source term step—or some part of such an emulation—can change the translated observables by changing the value of sum locks from true to false, which also unguards some encoded source term continuation (in the source guarded by τ, input, or replicated input) and, if the emulated step was not a τ-step, an instantiation of a sender lock to unguard the encoded continuation of the source term sender.

To show that the notion of translated observables indeed captures our intuition we prove that the set of observables of a source term coincides with the set of translated observables of its encoding.

Lemma 6.3.29. *For all three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, the set of observables of any source term is equal to the set of translated observables of its encoding.*

Proof. We have to show that for all $S_1 \in \mathcal{P}_{\mathrm{s}}$, all $S_2 \in \mathcal{P}_{\mathrm{m}}$ that do not contain replicated inputs, and all $S_3 \in \mathcal{P}_{\mathrm{m}}$ it holds that

$$\forall \mu \in \mathcal{N} \cup \overline{\mathcal{N}} \,. \quad S_1 \downarrow_\mu \text{ iff } [\![S_1]\!]^{\mathrm{s}}_{\mathrm{a}} \downarrow^1_\mu \;\wedge\; S_2 \downarrow_\mu \text{ iff } [\![S_2]\!]^{\mathrm{m}}_{\mathrm{p}} \downarrow^2_\mu \;\wedge\; S_3 \downarrow_\mu \text{ iff } [\![S_3]\!]^{\mathrm{m}}_{\mathrm{a}} \downarrow^3_\mu .$$

By Lemma 6.3.10 for all unguarded sums in S there is a positively instantiated sum lock in $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ and each positively instantiated sum lock belongs to the translation of an unguarded source term sum. In the case of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, by Lemma 6.3.19 and Lemma 6.3.16, some inputs on translated source term names are guarded by a replicated input on a receiver lock but this replicated input is unguarded and the lock is instantiated iff the respective source term input is unguarded in S. By Lemma 6.3.21 and Lemma 6.3.14, in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ requests are initially guarded by replicated inputs on auxiliary

sender or receiver locks but again these are unguarded and the locks are instantiated iff the respective source term out- or input is unguarded in S. We conclude by Definition 6.3.4 and the respective invariants for links on translated source term names and requests, i.e., by Lemma 6.3.19, Lemma 6.3.21, and Lemma 6.3.22. □

Moreover, the same holds for the set of reachable observables.

Lemma 6.3.30. *For all three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, the set of reachable observables of any source term is equal to the set of reachable translated observables of its encoding.*

Proof. We have to show that for all $S_1 \in \mathcal{P}_{\mathrm{s}}$, all $S_2 \in \mathcal{P}_{\mathrm{m}}$ that do not contain replicated inputs, and all $S_3 \in \mathcal{P}_{\mathrm{m}}$ it holds that

$$\forall \mu \in \mathcal{N} \cup \overline{\mathcal{N}} \,.\quad S_1 \Downarrow_\mu \text{ iff } [\![S_1]\!]^{\mathrm{s}}_{\mathrm{a}} \Downarrow^1_\mu \wedge S_2 \Downarrow_\mu \text{ iff } [\![S_2]\!]^{\mathrm{m}}_{\mathrm{p}} \Downarrow^2_\mu \wedge S_3 \Downarrow_\mu \text{ iff } [\![S_3]\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow^3_\mu \,.$$

This follows from Lemma 6.3.29, operational correspondence, i.e., by Lemmata 6.3.52 at page 276, 6.3.53 at page 276, and 6.3.54 at page 277, and the definition of the respective version of $\asymp$ as explained below.[3] □

Furthermore, we show that administrative steps do not only preserve translated observables but also do not remove any.

Lemma 6.3.31. *Administrative steps do not influence the set of translated observables.*

Proof. Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ such that $T_1 \overset{\cdot}{\longmapsto} T_2$. Then, by Definition 6.3.26, $T_2 \downarrow^1_\mu$ implies $T_1 \downarrow^1_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Assume $T_1 \downarrow^1_\mu$ for some arbitrary $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. By Definition 6.3.4, administrative steps neither reduce positive instantiations of sum locks nor capabilities on translated source term names. Moreover, by Lemma 6.3.25, administrative steps are not in conflict with any other step. Hence, by Definition 6.3.26, $T_2 \downarrow^1_\mu$. We conclude that $T_2 \downarrow^1_\mu$ iff $T_1 \downarrow^1_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$.

Let $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$ such that $T_1 \overset{\cdot}{\longmapsto} T_2$. Then, by Definition 6.3.27, $T_2 \downarrow^2_\mu$ implies $T_1 \downarrow^2_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Assume $T_1 \downarrow^2_\mu$ for some arbitrary $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. By Definition 6.3.4, administrative steps do not reduce positive instantiations of sum locks and, by Lemma 6.3.21, requests are preserved modulo $\overset{\cdot\cdot}{\longmapsto}$-steps. Moreover, by Lemma 6.3.25, administrative steps are not in conflict with any other step. Hence, by Definition 6.3.27, $T_2 \downarrow^2_\mu$. We conclude that $T_2 \downarrow^2_\mu$ iff $T_1 \downarrow^2_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$.

Let $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ such that $T_1 \overset{\cdot}{\longmapsto} T_2$. Then, by Definition 6.3.28, $T_2 \downarrow^3_\mu$ implies $T_1 \downarrow^3_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Assume $T_1 \downarrow^3_\mu$ for some arbitrary $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. By Definition 6.3.4, administrative steps do not reduce positive instantiations of sum locks and, by Lemma 6.3.22, requests are preserved modulo $\overset{\cdot\cdot}{\longmapsto}$-steps. Moreover, by Lemma 6.3.25, administrative steps are either not in conflict with any other step or by repeating the argument in the proof of Lemma 6.3.25 they do not influence reachability of instantiations on receiver locks and thus the possibilities to unguard a test-construct. Hence, by Definition 6.3.28, $T_2 \downarrow^3_\mu$. We conclude that $T_2 \downarrow^3_\mu$ iff $T_1 \downarrow^3_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. □

[3] Note that we use this lemma only to prove Lemma 6.3.56, i.e., not to prove one of the used lemmata.

In contrast, core as well as impure steps can remove translated observables by consuming positive instantiations of sum locks. But only core steps lead to new translated observables, because by Lemma 6.3.25 they unguard encoded continuations of guarded source terms.

6.3.4. A Behavioural Equivalence

Before we can prove operational correspondence we have to fix the equivalence $\asymp$ in the settings of all three encodings. According to [Gor10b], $\asymp$ is a behavioural equivalence that is usually a congruence at least with respect to parallel composition. Moreover, by the criteria in Section 3.3, $\asymp$ has to respect success. The main purpose of $\asymp$ in the definition of operational correspondence is to abstract from junk, i.e., left over of former emulations that do not influence the abstract behaviour of a target term.

Since the target language is the asynchronous pi-calculus, it seems natural to choose weak asynchronous bisimilarity $\approx_{\mathrm{a}}$ or asynchronous barbed congruence $\dot{\cong}_{\mathrm{a}}$. Unfortunately for both choices the presented encodings are not good. Consider for example the source term $S = (\nu x)\,(x + y \mid \overline{x})$. It can perform a reduction to $\mathsf{0}$. But all derivatives of its encoding, i.e., all $T \in \mathcal{P}_{\mathrm{a}}$ with $[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} \longmapsto T$ or $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \longmapsto T$, are neither asynchronously barbed bisimilar nor asynchronously barbed congruent to the encoding of $\mathsf{0}$, i.e., $T \not\approx_{\mathrm{a}} (\nu l)\,\big(\overline{l}\langle\top\rangle\big)$ and $T \not\dot{\cong}_{\mathrm{a}} (\nu l)\,\big(\overline{l}\langle\top\rangle\big)$, where $[\![\, \mathsf{0} \,]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu l)\,\big(\overline{l}\langle\top\rangle\big) = [\![\, \mathsf{0} \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Note that this is not due to the encoding of $\mathsf{0}$, which is indeed weak asynchronously bisimilar to $\mathsf{0}$ again, but to the observable junk, which suffices to distinguish the left over of emulations from $\mathsf{0}$. Because of this, a proof of the correctness of these encodings with respect to $\approx_{\mathrm{a}}$ or $\dot{\cong}_{\mathrm{a}}$ fails due to operational correspondence (see the Definition 3.3.4). Of course, you might argue that an encoding that cannot get rid of observable junk is not a good encoding. On the other side, Nestmann in [Nes00] gives some good reasons to accept $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ as a good encoding. Moreover, the translation of observables into something different seems to be quite a natural habit of encoding functions. And indeed rephrasing a standard equivalence to take instead of observables translated observables into account suffices to turn it into an equivalence that describes the abstract behaviour of encoded terms. The same holds if we do not consider observables at all, but e.g. consider only reachability of success.

We denote the variants of barbed bisimilarity that use translated observables instead of standard observables as translated barbed bisimilarity. Because of the different formulations of translated observables for the three encoding functions we obtain three different notions of translated barbed bisimilarity. Moreover, we augment the following variants of barbed bisimilarity with an additional requirement that ensures that the equivalences respect success.

Definition 6.3.32 (Translated Barbed Bisimilarity)**.** Let $P, Q \in \mathcal{P}_{\mathrm{a}}$. Then P and Q are *translated barbed bisimilar* with respect to $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, denoted by $P \,\dot{\approx}^{\downarrow 1}\, Q$, if

1. $P \Downarrow_{\checkmark}$ iff $Q \Downarrow_{\checkmark}$,
2. for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$, $P \Downarrow^{1}_{\mu}$ iff $Q \Downarrow^{1}_{\mu}$,

3. for all $P' \in \mathcal{P}_{\mathrm{a}}$, $P \longmapsto P'$ implies $Q \Longmapsto \dot{\approx}^{\downarrow 1} P'$, and
4. for all $Q' \in \mathcal{P}_{\mathrm{a}}$, $Q \longmapsto Q'$ implies $P \Longmapsto \dot{\approx}^{\downarrow 1} Q'$.

Let $P, Q \in \mathcal{P}_{\mathrm{p}}$. Then P and Q are *translated barbed bisimilar* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, denoted by $P \dot{\approx}^{\downarrow 2} Q$, if

1. $P \Downarrow_{\checkmark}$ iff $Q \Downarrow_{\checkmark}$,
2. for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$, $P \Downarrow^2_{\mu}$ iff $Q \Downarrow^2_{\mu}$,
3. for all $P' \in \mathcal{P}_{\mathrm{p}}$, $P \longmapsto P'$ implies $Q \Longmapsto \dot{\approx}^{\downarrow 2} P'$, and
4. for all $Q' \in \mathcal{P}_{\mathrm{p}}$, $Q \longmapsto Q'$ implies $P \Longmapsto \dot{\approx}^{\downarrow 2} Q'$.

Let $P, Q \in \mathcal{P}^{=}_{\mathrm{a}}$. Then P and Q are *translated barbed bisimilar* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, denoted by $P \dot{\approx}^{\downarrow 3} Q$, if

1. $P \Downarrow_{\checkmark}$ iff $Q \Downarrow_{\checkmark}$,
2. for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$, $P \Downarrow^3_{\mu}$ iff $Q \Downarrow^3_{\mu}$,
3. for all $P' \in \mathcal{P}^{=}_{\mathrm{a}}$, $P \longmapsto P'$ implies $Q \Longmapsto \dot{\approx}^{\downarrow 3} P'$, and
4. for all $Q' \in \mathcal{P}^{=}_{\mathrm{a}}$, $Q \longmapsto Q'$ implies $P \Longmapsto \dot{\approx}^{\downarrow 3} Q'$.

Note that the first condition of each equivalence ensures that it respects success. The other conditions then define a version of weak barbed bisimilarity which uses translated observables instead of standard barbs. Note that we consider the translation of input as well as output observables, although our target language is asynchronous. However, in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ both translated output as well as translated input barbs are represented by requests, i.e., outputs on request channels. Moreover, the consideration of all standard observables of a source language can help to reason about the relationship of source and target terms. In particular it can help to reason backwards from a target term to its original source term as it is required by operational soundness.

Alternatively, we could decide not to consider barbs at all, by omitting the second condition of the above equivalences. We result then in an equivalence that considers only reachability of ✓ as abstract behaviour of a term. Note that this intuition goes along very well with the criteria defined by Gorla as they also do only require a similar reachability of ✓. This allows to base the notion of abstract behaviour on an observable that is defined independently from a specific source or target language. An advantage of such an equivalence is that it is independent of the considered encoding function. However, in the presented case the resulting equivalence is trivial, because it reduces to weak success respecting reduction bisimulation that cannot distinguish between more than three cases. Moreover, translated observables reflect a main operating principle of the encoding functions and thus ease the following argumentation on their correctness.

The encodings $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ induce also another problem concerning the choice of an appropriate equivalence, although that problem is not that crucial as observable

junk. As explained in Section 6.3.1, encodings of structurally congruent source terms can differ in the number and nature of reachable partially committed states. Operational soundness explicitly allows for intermediate states, i.e., target term states that do not map to the encodings of any of the corresponding source terms. So the presence of partially committed states in the encoding of some source term does not cause any problem. However, if $\asymp$ distinguishes target terms by the reachability of partially committed states, we have a problem with the reduction Rule Pi-Cong$_{\mathrm{m,s,a,p}}$ of Figure 2.3 and operational completeness of Definition 3.3.4. Let us consider the source terms $S = (a.S_1 + a.S_2) \mid \overline{a} \mid a.S_3$ and $S' = \overline{a} \mid (a.S_1 + a.S_2) \mid a.S_3$. The source term $b.S \mid \overline{b}$ can reduce to S but, because of Pi-Cong$_{\mathrm{m,s,a,p}}$, it can reduce to S' as well. $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ can emulate the first step modulo $\dot{\approx}^{\downarrow 3}$ but not the second step. Note that Pi-Cong$_{\mathrm{m,s,a,p}}$ is used to shorten the presentation of the reduction semantics, but it is neither necessary nor was it the original choice. So the most natural way to circumvent this problem is to rephrase the rules of the reduction semantics by avoiding Rule Pi-Cong$_{\mathrm{m,s,a,p}}$ and with it the possibility to arbitrary reorder the subprocesses during reductions. However we can also circumvent this problem by using a form of coupled simulation which does not distinguish terms by the reachability of partially committed states. Therefore we adapt the definition of weak barbed simulation in Section 2.2.2 such that it considers translated observables instead of the standard observables of the pi-calculus.

Definition 6.3.33 (Translated Barbed Simulation)**.** $\mathcal{S} \subseteq \mathcal{P}_{\mathrm{p}} \times \mathcal{P}_{\mathrm{p}}$ is a *weak translated barbed simulation* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ if $(P, Q) \in \mathcal{S}$ implies that

1. $P \Downarrow_{\checkmark}$ implies $Q \Downarrow_{\checkmark}$,
2. $P \Downarrow^2_{\mu}$ implies $Q \Downarrow^2_{\mu}$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$, and
3. for all $P' \in \mathcal{P}_{\mathrm{p}}$ such that $P \Longmapsto P'$ there is some $Q' \in \mathcal{P}_{\mathrm{p}}$ such that $Q \Longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

$P, Q \in \mathcal{P}_{\mathrm{p}}$ are *weakly translated barbed similar* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, denoted as $P \dot{\lesssim}^{\downarrow 2} Q$, if they are related by some weak translated barbed simulation.

$\mathcal{S} \subseteq \mathcal{P}^{=}_{\mathrm{a}} \times \mathcal{P}^{=}_{\mathrm{a}}$ is a *weak translated barbed simulation* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ if $(P, Q) \in \mathcal{S}$ implies that

1. $P \Downarrow_{\checkmark}$ implies $Q \Downarrow_{\checkmark}$,
2. $P \Downarrow^3_{\mu}$ implies $Q \Downarrow^3_{\mu}$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$, and
3. for all $P' \in \mathcal{P}^{=}_{\mathrm{a}}$ such that $P \Longmapsto P'$ there is some $Q' \in \mathcal{P}^{=}_{\mathrm{a}}$ such that $Q \Longmapsto Q'$ and $(P', Q') \in \mathcal{S}$.

$P, Q \in \mathcal{P}^{=}_{\mathrm{a}}$ are *weakly translated barbed similar* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, denoted as $P \dot{\lesssim}^{\downarrow 3} Q$, if they are related by some weak translated barbed simulation.

Again we consider the translations of output as well as input observables of the source term. Note that by Lemma 6.3.25 there is eventually an instantiation of each consumed sum lock. Thus $\left(\dot{\lesssim}^{\downarrow 3}, \left(\dot{\lesssim}^{\downarrow 3} \right)^{-1} \right)$ is a coupled translated barbed simulation.

Definition 6.3.34 (Coupled Translated Barbed Simulation)**.** A mutual translated barbed simulation is a pair $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ and $\mathcal{S}_2^{-1}$ are weak translated barbed simulations.

A mutual barbed simulation $(\mathcal{S}_1, \mathcal{S}_2)$ with respect to $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ is a *coupled translated barbed simulation* with respect to $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$ if

1. for all $(P, Q) \in \mathcal{S}_1$ there is some $Q' \in \mathcal{P}_{\mathrm{p}}$ such that $Q \Longmapsto Q'$ and $(P, Q') \in \mathcal{S}_2$, and
2. for all $(P, Q) \in \mathcal{S}_2$ there is some $P' \in \mathcal{P}_{\mathrm{p}}$ such that $P \Longmapsto P'$ and $(P', Q) \in \mathcal{S}_1$.

Two processes $P, Q \in \mathcal{P}_{\mathrm{p}}$ are *coupled translated barbed similar* with respect to $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$, denoted as $P \mathrel{\dot{\leftrightarrows}}^{\downarrow 2} Q$, if they are related by both components of some coupled barbed simulation with respect to $[\![\cdot]\!]_{\mathrm{p}}^{\mathrm{m}}$.

A mutual barbed simulation $(\mathcal{S}_1, \mathcal{S}_2)$ with respect to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ is a *coupled translated barbed simulation* with respect to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$ if

1. for all $(P, Q) \in \mathcal{S}_1$ there is some $Q' \in \mathcal{P}_{\mathrm{a}}^{=}$ such that $Q \Longmapsto Q'$ and $(P, Q') \in \mathcal{S}_2$, and
2. for all $(P, Q) \in \mathcal{S}_2$ there is some $P' \in \mathcal{P}_{\mathrm{a}}^{=}$ such that $P \Longmapsto P'$ and $(P', Q) \in \mathcal{S}_1$.

Two processes $P, Q \in \mathcal{P}_{\mathrm{a}}^{=}$ are *coupled translated barbed similar* with respect to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$, denoted as $P \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} Q$, if they are related by both components of some coupled barbed simulation with respect to $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$.

Note that coupled translated barbed simulation is strictly weaker than translated barbed bisimulation and that we use it to circumvent the problem with Pi-Cong$_{\mathrm{m,s,a,p}}$ in operational completeness only. More precisely, we use it to show preservation of structural congruence for the source term in Lemma 6.3.42. For the remaining results we use the stricter equivalence.

Observation 6.3.35. $\mathrel{\dot{\approx}}^{\downarrow 2} \subset \mathrel{\dot{\leftrightarrows}}^{\downarrow 2}$ and $\mathrel{\dot{\approx}}^{\downarrow 3} \subset \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}$.

Also note that already in [Nes96, NP00, Nes00] coupled simulation is used to reason about encodings of choice. There, however, coupled simulation is used in order to obtain a full-abstraction result. We observe that in the general framework of Gorla (Section 3.3) coupled simulation is not necessary to reason about $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$. And in the case of our encodings of mixed choice, it is only necessary if we do not forbid Rule Pi-Cong$_{\mathrm{m,s,a,p}}$ in the source language.

The observable junk does not only rule out standard equivalences but also congruences with respect to contexts that allow for interaction with observable junk. Such an interaction can for instance lead to a positive instantiation of a formerly negative instantiation of a sum lock and so turn an observable junk into a translated observable, or it can instantiate a sender lock and so complete emulations on junk.

Example 6.3.36. Let us consider the target term $T = (\nu l)\left([\![\overline{y}.\checkmark]\!]_{\mathrm{a}}^{\mathrm{m}} \mid \overline{l}\langle\bot\rangle\right)$ of $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{m}}$. We prove in the next section (Lemma 6.3.49) that this term is junk. Such a term can

result from an emulation of a source term step of an out- or input within choice, e.g. for a source term $\overline{x}+\overline{y}.\checkmark \mid x$. Since T does not reach any translated observable or unguarded occurrence of $\checkmark$, we have $T \dot{\approx}^{\downarrow 3} [\![\mathsf{0}]\!]^{\mathrm{m}}_{\mathrm{a}}$. However, we can distinguish T from $[\![\mathsf{0}]\!]^{\mathrm{m}}_{\mathrm{a}}$ by the context $\mathcal{C}([\cdot]) = [\cdot] \mid p_o(-,-,s,-).\overline{s}$, where p_o is the free output request channel of $[\![\overline{y}.\checkmark]\!]^{\mathrm{m}}_{\mathrm{a}}$. We have $\mathcal{C}(T) \Downarrow_{\checkmark}$ but $\mathcal{C}(\mathsf{0}) \not\Downarrow_{\checkmark}$, i.e., $\mathcal{C}(T) \not\dot{\approx}^{\downarrow 3} \mathcal{C}(\mathsf{0})$.

Because of this, we have to reduce the number of contexts we consider to obtain a congruence. Intuitively, we consider only contexts that respect the protocol of the encoding function. Thus, we consider only contexts that, if their argument is a target term as for instance the encoding of $\mathsf{0}$, result in a target term.

Definition 6.3.37 (Translated Barbed Congruence)**.** Two terms $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}$ are *translated barbed congruent* with respect to $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, denoted as $T_1 \dot{\cong}^{\downarrow 1} T_2$, if

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}} \,.\, \forall S \in \mathcal{P}_{\mathrm{s}} \,.\, \mathcal{C}([\![S]\!]^{\mathrm{s}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}} \text{ implies } \mathcal{C}(T_1) \dot{\approx}^{\downarrow 1} \mathcal{C}(T_2)\,.$$

Two target terms $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}$ are *translated barbed congruent* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, denoted as $T_1 \dot{\cong}^{\downarrow 2} T_2$, if

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{p}} \to \mathcal{P}_{\mathrm{p}} \,.\, \forall S \in \mathcal{P}_{\mathrm{m}} \,.\, \mathcal{C}\left([\![S]\!]^{\mathrm{m}}_{\mathrm{p}}\right) \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}} \text{ implies } \mathcal{C}(T_1) \dot{\approx}^{\downarrow 2} \mathcal{C}(T_2)\,.$$

Two target terms $T_1, T_2 \in \mathcal{P}_{\mathrm{p}}$ are *coupled translated barbed congruent* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, denoted as $T_1 \dot{\leftrightarrows}^{\downarrow 2}_{\mathrm{c}} T_2$, if

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{p}} \to \mathcal{P}_{\mathrm{p}} \,.\, \forall S \in \mathcal{P}_{\mathrm{m}} \,.\, \mathcal{C}\left([\![S]\!]^{\mathrm{m}}_{\mathrm{p}}\right) \in \mathcal{P}_{\mathrm{p}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}} \text{ implies } \mathcal{C}(T_1) \dot{\leftrightarrows}^{\downarrow 2} \mathcal{C}(T_2)\,.$$

Two target terms $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}$ are *translated barbed congruent* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, denoted as $T_1 \dot{\cong}^{\downarrow 3} T_2$, if

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}} \,.\, \forall S \in \mathcal{P}_{\mathrm{m}} \,.\, \mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}} \text{ implies } \mathcal{C}(T_1) \dot{\approx}^{\downarrow 3} \mathcal{C}(T_2)\,.$$

Two target terms $T_1, T_2 \in \mathcal{P}^{=}_{\mathrm{a}}$ are *coupled translated barbed congruent* with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, denoted as $T_1 \dot{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} T_2$, if

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}} \,.\, \forall S \in \mathcal{P}_{\mathrm{m}} \,.\, \mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}} \text{ implies } \mathcal{C}(T_1) \dot{\leftrightarrows}^{\downarrow 3} \mathcal{C}(T_2)\,.$$

Operational correspondence considers only target terms, so it would suffice to define the congruence over target terms only. However, in defining it over all terms of the target language we gain more flexibility. In particular, it allows us to stepwise reduce junk which in some cases leads to non target terms. Since these non target terms are behaviourally equivalent to the considered target terms, they serve as connecting pieces to link the target terms. Moreover, as the example above shows, the congruence relations are strictly weaker than their corresponding equivalences.

Observation 6.3.38. $\dot{\cong}^{\downarrow 1} \subset \dot{\approx}^{\downarrow 1}$, $\dot{\cong}^{\downarrow 2} \subset \dot{\approx}^{\downarrow 2}$, $\dot{\cong}^{\downarrow 2} \subset \dot{\leftrightarrows}^{\downarrow 2}_{\mathrm{c}} \subset \dot{\leftrightarrows}^{\downarrow 2}$, $\dot{\cong}^{\downarrow 3} \subset \dot{\approx}^{\downarrow 3}$, and $\dot{\cong}^{\downarrow 3} \subset \dot{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} \subset \dot{\leftrightarrows}^{\downarrow 3}$.

Of course all the presented congruences contain structural congruence of target terms.

Lemma 6.3.39. *Translated barbed congruence includes structural congruence.*

Proof. Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}^{=}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$. By Definitions 3.2.2 and 6.3.28, $T_1 \equiv T_2$ implies $T_1 \Downarrow_{\checkmark}$ iff $T_2 \Downarrow_{\checkmark}$ and $T_1 \downarrow_{\mu}^{3}$ iff $T_2 \downarrow_{\mu}^{3}$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. By Pi-Cong$_{\mathrm{m,s,a,p}}$ in the reduction semantics of $\pi_{\mathrm{a}}^{=}$ in Figure 2.3, then $T_1 \mathrel{\dot{\approx}}^{\downarrow 3} T_2$. Hence, $\equiv \subset \mathrel{\dot{\approx}}^{\downarrow 3}$

Moreover, let $\mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{a}}^{=} \to \mathcal{P}_{\mathrm{a}}^{=}$ such that $\mathcal{C}([\![\, S \,]\!]_{\mathrm{a}}^{\mathrm{m}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{s}}}$ for all source terms $S \in \mathcal{P}_{\mathrm{m}}$. Then obviously $\mathcal{C}(T_1) \equiv \mathcal{C}(T_2)$ and thus, by the above argument, $\mathcal{C}(T_1) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}(T_2)$.

The argument for the other encodings is similar. Hence, $\equiv \subset \mathrel{\dot{\cong}}^{\downarrow 1}$, $\equiv \subset \mathrel{\dot{\cong}}^{\downarrow 2}$, and $\equiv \subset \mathrel{\dot{\cong}}^{\downarrow 3}$. □

Remember that to our intuition administrative steps are only pre- or postprocessing steps that do not influence which emulations can be completed. To underpin that intuition, we prove that administrative steps do not change the state of a target term modulo the considered equivalences and congruences.

Lemma 6.3.40. *Administrative steps do not influence the state of a target term modulo translated barbed congruence.*

Proof. Let $T_1, T_2 \in \mathcal{P}_{\mathrm{a}}^{=}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$ such that $T_1 \mathrel{\dot{\longmapsto}} T_2$. By Lemma 6.3.25, $T_1 \Downarrow_{\checkmark}$ iff $T_2 \Downarrow_{\checkmark}$. By Lemma 6.3.31, $T_1 \downarrow_{\mu}^{3}$ iff $T_2 \downarrow_{\mu}^{3}$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Moreover, by Lemma 6.3.25, most of the administrative steps are not in conflict with any other step and the remaining steps do only influence the order of elements in chains but do not influence reachability of instantiations on receiver locks and thus reachability of unguarded outputs on sum locks. Hence, $T_1 \mathrel{\dot{\approx}}^{\downarrow 3} T_2$.

Moreover, let $\mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{a}}^{=} \to \mathcal{P}_{\mathrm{a}}^{=}$ such that $\mathcal{C}([\![\, S \,]\!]_{\mathrm{a}}^{\mathrm{m}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{s}}}$ for all source terms $S \in \mathcal{P}_{\mathrm{m}}$. Then obviously $\mathcal{C}(T_1), \mathcal{C}(T_2) \in \mathcal{P}_{\mathrm{a}}^{=}\restriction_{[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{m}}}$. If T_1 is unguarded in $\mathcal{C}$ then $\mathcal{C}(T_1) \mathrel{\dot{\longmapsto}} \mathcal{C}(T_2)$ and, by the argument above, $\mathcal{C}(T_1) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}(T_2)$. If there is no way to unguard T_1 in $\mathcal{C}$ then there is also no way for the context to unguard T_2 or to interact with one of the terms, i.e., again $\mathcal{C}(T_1) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}(T_2)$. Else there is an execution $\mathcal{C}(T_1) \Longmapsto \mathcal{C}'(T_1)$ and so $\mathcal{C}(T_2) \Longmapsto \mathcal{C}'(T_2)$. By the above argument, $\mathcal{C}'(T_1) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}'(T_2)$. By an induction on the number of steps in $\mathcal{C}(T_1) \Longmapsto \mathcal{C}'(T_1)$, then also $\mathcal{C}(T_1) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}(T_2)$. Hence, by Definition 6.3.37, also $T_1 \mathrel{\dot{\cong}}^{\downarrow 3} T_2$.

The argumentation for the other encodings is similar. □

Note that, because of this lemma, we can mostly ignore administrative steps in the following proofs. To deal with the reduction Rule Pi-Cong$_{\mathrm{m,s,a,p}}$ in the proof of operational completeness, we prove that the encodings preserve structural congruence of source terms modulo the presented congruences.

Lemma 6.3.41. *The encoding $[\![\,\cdot\,]\!]_{\mathrm{a}}^{\mathrm{s}}$ preserves structural congruence of source terms modulo translated barbed congruence, i.e.,*

$$\forall S, S' \in \mathcal{P}_{\mathrm{s}} \,.\, S \equiv S' \text{ implies } [\![\, S \,]\!]_{\mathrm{a}}^{\mathrm{s}} \mathrel{\dot{\cong}}^{\downarrow 1} [\![\, S' \,]\!]_{\mathrm{a}}^{\mathrm{s}}.$$

Proof. The strict use of the renaming policy $\varphi^{\mathrm{s}}_{\mathrm{a}}$, i.e., the fact that source term names are translated into single names not used by the encoding function for special purposes, ensures the preservation of equality modulo α-conversion. Since the parallel operator and restriction are translated homomorphically, the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves structural congruence of source terms for all rules except for $P \mid \mathsf{0} \equiv P$, i.e., if S and S' are structurally congruent without using the rule $P \mid \mathsf{0} \equiv P$, then $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv [\![S']\!]^{\mathrm{s}}_{\mathrm{a}}$. By Lemma 6.3.39, then $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\cong}}^{\downarrow 1} [\![S']\!]^{\mathrm{s}}_{\mathrm{a}}$.

The rule $P \mid \mathsf{0} \equiv P$ is not preserved, because the empty sum $\mathsf{0}$ is not translated homomorphically, so e.g. $\mathsf{0} \mid \mathsf{0} \equiv \mathsf{0}$ but $[\![\mathsf{0} \mid \mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu l)\, \overline{l}\langle \top \rangle \mid (\nu l)\, \overline{l}\langle \top \rangle \not\equiv (\nu l)\, \overline{l}\langle \top \rangle = [\![\mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}}$. Note that, because of the renaming policy $\varphi^{\mathrm{s}}_{\mathrm{a}}$ and the homomorphic translation of restriction, the rule $(\nu n)\, \mathsf{0} \equiv \mathsf{0}$ is preserved, i.e., since $\varphi^{\mathrm{s}}_{\mathrm{a}}(n) \notin \mathsf{fn}([\![\mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}})$, we have $[\![(\nu n)\, \mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(n))\, (\nu l)\, \overline{l}\langle \top \rangle \equiv (\nu l)\, \overline{l}\langle \top \rangle = [\![\mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}}$. However, since $\mathsf{0}$ is translated into a closed term that cannot perform any step, its encoding behaves as $\mathsf{0}$. In particular $[\![\mathsf{0}]\!]^{\mathrm{s}}_{\mathrm{a}}$ cannot interact with any context and does not reach success or any translated observable on its own. So, even in this case, we have $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\cong}}^{\downarrow 1} [\![S']\!]^{\mathrm{s}}_{\mathrm{a}}$. □

Since $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ do not translate the parallel operator homomorphically, structural congruence of source terms is not preserved. But, since the encoding preserves the abstract behaviour of source terms, the encodings of structurally congruent source terms are similar modulo equivalences measuring only the abstract behaviour. As already explained, to prove the following statement, the equivalence must not distinguish terms by their reachable partially committed states.

Lemma 6.3.42. *The encodings $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ preserve structural congruence of source terms modulo coupled translated barbed congruence, i.e.,*

$$\forall S, S' \in \mathcal{P}_{\mathrm{m}} \,.\, S \equiv S' \text{ implies } [\![S]\!]^{\mathrm{m}}_{\mathrm{p}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 2}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{p}} \wedge [\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}} \,.$$

Proof. Again, the strict use of the renaming policy $\varphi^{\mathrm{m}}_{\mathrm{a}}$ ensures the preservation of equality modulo α-conversion. So $S \equiv_\alpha S'$ implies $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv_\alpha [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$. Also, the homomorphic translation of restriction ensures the preservation of structural congruence modulo the rules $(\nu n)\, \mathsf{0} = \mathsf{0}$, $(\nu n)\, (\nu m)\, P \equiv (\nu m)\, (\nu n)\, P$, and $P \mid (\nu n)\, Q \equiv (\nu n)\, (P \mid Q)$ if $n \notin \mathsf{fn}(P)$. So, if S and S' do only differ due to one or more of these three rules, then $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$. By Lemma 6.3.39, we conclude $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ for the above cases.

Now consider a single application of the remaining structural congruence rules. The proof is then by induction on the number of structural congruence rules necessary to obtain $S \equiv S'$.

Case of $P \mid \mathsf{0} \equiv P$: In this case $S = P \mid \mathsf{0}$ and $S' = P$ for some $P \in \mathcal{P}_{\mathrm{m}}$. By Figure 5.8,

we have

$$\begin{aligned}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(&\\ (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})&\\ \mid (\nu p_o, p_i)\,\big((\nu l)\,\overline{l}\langle\top\rangle \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big)&\\ \mid \mathsf{pushReq}\big)&\end{aligned}$$

and $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}} = [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Obviously $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ are not structurally congruent. However, $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ appears unguarded within $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, so if $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ reaches $\checkmark$ or a translated observable then so does $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and vice versa. Moreover we observe that, since the encoding of $\mathsf{0}$ does not emit any requests, the right branch of $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$

$$(\nu p_o, p_i)\,\big((\nu l)\,\overline{l}\langle\top\rangle \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big)$$

can do nothing but two steps on chain locks. Because of that, requests of $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ are prepared to be transmitted to the right side by $\mathsf{procLeftOutReq}$ and $\mathsf{procLeftInReq}$ but they are never received at the right side. What remains is the upward pushing of all requests of $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ by the interplay of $\mathsf{procLeftOutReq}$, $\mathsf{procLeftInReq}$, and $\mathsf{pushReq}$. Because of that, for all target term contexts $[\![\, P \mid \mathsf{0} \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ differ only by administrative steps. By Lemma 6.3.40, $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \overset{\therefore}{\leftrightarrows}{}^{\downarrow 3}_{\mathrm{c}}\ [\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of $P \mid Q \equiv Q \mid P$: In this case $S = P \mid Q$ and $S' = Q \mid P$ for some $P, Q \in \mathcal{P}_{\mathrm{m}}$. Their encodings are given by:

$$\begin{aligned}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(&\\ (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})&\\ \mid (\nu p_o, p_i)\,([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})&\\ \mid \mathsf{pushReq}\big)&\\ [\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(&\\ (\nu p_o, p_i)\,([\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})&\\ \mid (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})&\\ \mid \mathsf{pushReq}\big)&\end{aligned}$$

Since all combinations of left and right requests are checked, $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ can emulate the same source term steps as $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. However, since $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ are swapped at the outermost parallel operator the roles of left and right requests are swapped. As a consequence, if a combination of requests from $[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, Q \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ leads to a test on the respective sum locks, the order in which these locks are tested is different in $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. So $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ differ in their total ordering of sum locks. The ordering in $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ is based on the structure induced by the nesting of parallel operators in $P \mid Q$, whereas the ordering in $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ is based on the structure induced by the parallel operator nesting in $Q \mid P$. Note that, since in both cases this structure is a binary tree, by Lemma 6.3.25, the encoding

does not introduce deadlock. But as explained in Example 6.3.6 the different orderings may lead to different reachable partially committed states. Apart from intermediate states, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ are similar, i.e., they have the same chance to reach success or translated observables. Note that consecutive impure steps may lead to a partially committed state that cannot be turned into a normal state by a single core step. But, because of Lemma 6.3.10, eventually for each consumed instantiation of a sum lock a new one is unguarded. Thus there is always a sequence of steps that resolves all started nested test-constructs, unguards an instantiation of all consumed instantiations of sum locks, and leads back to a normal state. Because of that, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$. Note that a context cannot change the order of sum locks within its parameter but only include this order within the order of its own sum locks. Because of that, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$: In this case $S = P \mid (Q \mid R)$ and $S' = (P \mid Q) \mid R$ for some $P, Q, R \in \mathcal{P}_{\mathrm{m}}$. Their encodings are given by

$$
\begin{aligned}
[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(&\\
&(\nu p_o, p_i)\,([\![P]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
&\mid (\nu p_o, p_i)\,\big(\ (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(\\
&\qquad (\nu p_o, p_i)\,([\![Q]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
&\qquad \mid (\nu p_o, p_i)\,([\![R]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\qquad \mid \mathsf{pushReq}\big)\\
&\quad \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big)\\
&\mid \mathsf{pushReq}\big)
\end{aligned}
$$

and

$$
\begin{aligned}
[\![S']\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(&\mathsf{pushReq}\\
&\mid (\nu p_o, p_i)\,\big(\ (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(\mathsf{pushReq}\\
&\qquad \mid (\nu p_o, p_i)\,([\![P]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
&\qquad \mid (\nu p_o, p_i)\,([\![Q]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\big)\\
&\quad \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big)\\
&\mid (\nu p_o, p_i)\,([\![R]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\big).
\end{aligned}
$$

In $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ the encoding of Q appears left and the encoding of R appears right within the encoding of a parallel operator. Together they form the right branch of a surrounding encoding of a parallel operator, there the left branch is filled with $[\![P]\!]^{\mathrm{m}}_{\mathrm{a}}$. In contrast, in $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ the terms $[\![P]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![Q]\!]^{\mathrm{m}}_{\mathrm{a}}$ are left and right of a parallel operator encoding which is the left branch of a surrounding parallel operator encoding, where $[\![R]\!]^{\mathrm{m}}_{\mathrm{a}}$ appears right. However, by Lemma 6.3.25, all requests are pushed upwards to each surrounding parallel operator encoding. Thus all combinations of requests among the three encoded subterms $[\![P]\!]^{\mathrm{m}}_{\mathrm{a}}$, $[\![Q]\!]^{\mathrm{m}}_{\mathrm{a}}$, and

$[\![R]\!]^{\mathrm{m}}_{\mathrm{a}}$ are checked in $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ as well as in $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$. Moreover, we observe that in both encodings $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ the encoding of P is always left to the encodings of Q and R, and the encoding of Q is always left to the encoding of R. So in this case $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ do not differ in the underlying total ordering of sum locks, i.e., they reach the same intermediate or partially committed states. So the behaviour of $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ does only differ in administrative steps on requests but they have in all contexts the same chance to reach ✓ and translated observables. By Lemma 6.3.40, then $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$.

The argumentation for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is similar. □

These two lemmata finally prove that partially committed states do not forbid for operational completeness even if we allow to have the reduction Rule Pi-Cong$_{\mathrm{m,s,a,p}}$ in the source language.

6.3.5. Junk

We consider left over of emulations that behave modulo $\dot{\approx}^{\downarrow 1}$, $\dot{\approx}^{\downarrow 2}$, and $\dot{\approx}^{\downarrow 3}$ like $\mathsf{0}$ and do not influence the possibility or inability to emulate further source term steps as junk. The emulation of source term steps may leave different kinds of junk. Note that, to prove that junk does no harm, is in particular necessary to show operational completeness.

Of course we are only interested in kinds of junk that appear in target terms, i.e., that are pieces of target terms. However, to ease the argumentation on the proof of operational completeness we want to allow to stepwise reduce junk. Unfortunately, as soon as we reduce a target term by the first piece of junk, it is often not a target term any more. So, in order to allow for a stepwise reduction of junk, we give a recursive definition of what it means to be a piece of a target term.

Definition 6.3.43 (Piece of a Target Term)**.** A term $T \in \mathcal{P}_{\mathrm{a}}$ is a *piece of a target term* in $\mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$, denoted by $T \in \mathfrak{P}\left(\mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}\right)$, if $T \in \mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}$ or there exists $T', J \in \mathcal{P}_{\mathrm{a}}$, and a sequence of names $\tilde{x}$ such that

$$T \equiv (\nu\tilde{x})\, T' \wedge T \stackrel{\cdot}{\cong}^{\downarrow 1} (\nu\tilde{x})\left(T' \mid J\right) \wedge (\nu\tilde{x})\left(T' \mid J\right) \in \mathfrak{P}\left(\mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}\right).$$

A term $T \in \mathcal{P}_{\mathrm{p}}$ is a *piece of a target term* in $\mathcal{P}_{\mathrm{p}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$, denoted by $T \in \mathfrak{P}\left(\mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}\right)$, if $T \in \mathcal{P}_{\mathrm{p}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}$ or there exists $T', J \in \mathcal{P}_{\mathrm{p}}$, and a sequence of names $\tilde{x}$ such that

$$T \equiv (\nu\tilde{x})\, T' \wedge T \stackrel{\cdot}{\cong}^{\downarrow 2} (\nu\tilde{x})\left(T' \mid J\right) \wedge (\nu\tilde{x})\left(T' \mid J\right) \in \mathfrak{P}\left(\mathcal{P}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}\right).$$

A term $T \in \mathcal{P}^{=}_{\mathrm{a}}$ is a *piece of a target term* in $\mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, denoted by $T \in \mathfrak{P}\left(\mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\right)$, if $T \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ or there exists $T', J \in \mathcal{P}^{=}_{\mathrm{a}}$, and a sequence of names $\tilde{x}$ such that

$$T \equiv (\nu\tilde{x})\, T' \wedge T \stackrel{\cdot}{\cong}^{\downarrow 3} (\nu\tilde{x})\left(T' \mid J\right) \wedge (\nu\tilde{x})\left(T' \mid J\right) \in \mathfrak{P}\left(\mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\right).$$

Intuitively, the definition above allows for a piece of a target term to recover the corresponding target term by stepwise restoring the reduced junk. Moreover note that, although the relations $\dot{\cong}^{\downarrow 1}$, $\dot{\cong}^{\downarrow 2}$, and $\dot{\cong}^{\downarrow 3}$ are not sensitive to divergence, they are sensitive to deadlock. Because of that, we require $T \dot{\cong}^{\downarrow 1} (\nu\tilde{x})\,(T' \mid J)$ to ensure that indeed only junk is removed and no deadlock is introduced.

Definition 6.3.44 (Junk)**.** A term $J \in \mathcal{P}_{\mathrm{a}}$ is called *junk* of the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ modulo $\dot{\approx}^{\downarrow 1}$ if J behaves modulo $\dot{\approx}^{\downarrow 1}$ like $\mathsf{0}$ for all pieces of target terms, i.e.,

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}} \,.\, \mathcal{C}(J) \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}\Big) \text{ implies } \mathcal{C}(J) \;\dot{\approx}^{\downarrow 1}\; \mathcal{C}(\mathsf{0})\,.$$

A term $J \in \mathcal{P}_{\mathrm{p}}$ is called *junk* of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ modulo $\dot{\approx}^{\downarrow 2}$ if J behaves modulo $\dot{\approx}^{\downarrow 2}$ like $\mathsf{0}$ for all pieces of target terms, i.e.,

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}_{\mathrm{p}} \to \mathcal{P}_{\mathrm{p}} \,.\, \mathcal{C}(J) \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}}\Big) \text{ implies } \mathcal{C}(J) \;\dot{\approx}^{\downarrow 2}\; \mathcal{C}(\mathsf{0})\,.$$

A term $J \in \mathcal{P}^{=}_{\mathrm{a}}$ is called *junk* of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ modulo $\dot{\approx}^{\downarrow 3}$ if J behaves modulo $\dot{\approx}^{\downarrow 3}$ like $\mathsf{0}$ for all pieces of target terms, i.e.,

$$\forall \mathcal{C}([\cdot]) : \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}} \,.\, \mathcal{C}(J) \in \mathfrak{P}\Big(\mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big) \text{ implies } \mathcal{C}(J) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}(\mathsf{0})\,.$$

Since we do not consider junk modulo equivalences different from $\dot{\approx}^{\downarrow 1}$, $\dot{\approx}^{\downarrow 2}$, and $\dot{\approx}^{\downarrow 3}$, we omit the equivalence in the following. Moreover we omit the encoding function if the considered junk appears within all three encodings.

Of course, whenever we reduce a piece of a target term by removing junk, the result is again a piece of a target term. Moreover, reducing junk does not change the behaviour of such a term modulo translated barbed congruence.

Lemma 6.3.45. *Let T be a piece of a target term including some junk J. Then removing this junk results in a piece of a target term which is congruent to T.*

Proof. For $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ we have to show that

$$T \equiv (\nu\tilde{x})\,(T' \mid J) \text{ implies } (\nu\tilde{x})\,T' \in \mathfrak{P}\Big(\mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big) \wedge T \;\dot{\cong}^{\downarrow 3}\; (\nu\tilde{x})\,T'$$

for all $T \in \mathfrak{P}\Big(\mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$, all $T' \in \mathcal{P}^{=}_{\mathrm{a}}$, all $J \in \mathcal{P}^{=}_{\mathrm{a}}$ that are junk of the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ modulo $\dot{\approx}^{\downarrow 3}$, and all sequences of names $\tilde{x}$.

Let $J \in \mathcal{P}^{=}_{\mathrm{a}}$ be junk. And let $T \in \mathfrak{P}\Big(\mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$, $T' \in \mathcal{P}^{=}_{\mathrm{a}}$, and $\tilde{x}$ a sequence of names such that $T \equiv (\nu\tilde{x})\,(T' \mid J)$. By Lemma 6.3.39, $T \equiv (\nu\tilde{x})\,(T' \mid J)$ implies $T \;\dot{\cong}^{\downarrow 3}\; (\nu\tilde{x})\,(T' \mid J)$. By Definition 6.3.37, then $\mathcal{C}(T) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}((\nu\tilde{x})\,(T' \mid J))$ for all contexts $\mathcal{C}([\cdot]) : \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}}$ such that $\mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $S \in \mathcal{P}_{\mathrm{m}}$. Let $\mathcal{C}'([\cdot]) = (\nu\tilde{x})\,(T' \mid [\cdot])$. Then $\mathcal{C}'([\cdot]) \in \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}}$, $T \equiv \mathcal{C}'(J)$, and $\mathcal{C}'(J) \in \mathfrak{P}\Big(\mathcal{P}^{=}_{\mathrm{a}}\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$. By Definition 6.3.37, $T \;\dot{\cong}^{\downarrow 3}\; (\nu\tilde{x})\,(T' \mid J)$ implies $\mathcal{C}(T) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}\Big(\mathcal{C}'(J)\Big)$ for all $\mathcal{C}([\cdot]) : \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}}$

such that $\mathcal{C}([\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $S \in \mathcal{P}_{\mathrm{m}}$. Moreover, $\mathcal{C}'(J) \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$ implies $\mathcal{C}\Big(\mathcal{C}'(J)\Big) \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$ for all such contexts $\mathcal{C}$. By Definition 6.3.44 of junk, then $\mathcal{C}\Big(\mathcal{C}'(J)\Big) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}\Big(\mathcal{C}'(0)\Big)$ for all such contexts $\mathcal{C}$. Since $\mathcal{C}\Big(\mathcal{C}'(0)\Big) = \mathcal{C}((\nu\tilde{x})\,(T' \mid 0)) \equiv \mathcal{C}((\nu\tilde{x})\,T')$ and by Lemma 6.3.39, then $\mathcal{C}\Big(\mathcal{C}'(J)\Big) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}((\nu\tilde{x})\,T')$ for all such contexts $\mathcal{C}$. Then $\mathcal{C}(T) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}\Big(\mathcal{C}'(J)\Big)$ for all such contexts $\mathcal{C}$ and $\mathcal{C}\Big(\mathcal{C}'(J)\Big) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}((\nu\tilde{x})\,T')$ for all such contexts $\mathcal{C}$ imply $\mathcal{C}(T) \mathrel{\dot{\approx}}^{\downarrow 3} \mathcal{C}((\nu\tilde{x})\,T')$ for all such contexts $\mathcal{C}$. Thus, by Definition 6.3.37, we conclude $T \mathrel{\dot{\cong}}^{\downarrow 3} (\nu\tilde{x})\,T'$.

Finally, by Lemma 6.3.39, $T \mathrel{\dot{\cong}}^{\downarrow 3} (\nu\tilde{x})\,T'$ implies $(\nu\tilde{x})\,T' \mathrel{\dot{\cong}}^{\downarrow 3} (\nu\tilde{x})\,(T' \mid J)$. Thus, since $(\nu\tilde{x})\,(T' \mid J) \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$, we conclude $(\nu\tilde{x})\,T' \in \mathfrak{P}\Big(\mathcal{P}_{\mathrm{a}}^{=}\!\upharpoonright_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}\Big)$.

The argumentation for the other encodings is similar. □

Using this lemma we can remove junk from a target term T. As result we obtain a piece of a target term T' such that $T \mathrel{\dot{\cong}}^{\downarrow 1} T'$, $T \mathrel{\dot{\cong}}^{\downarrow 2} T'$, or $T \mathrel{\dot{\cong}}^{\downarrow 3} T'$, respectively. Then we can further reduce T' by removing junk such that we result in a piece of a target term T'' with $T' \mathrel{\dot{\cong}}^{\downarrow 1} T''$, $T' \mathrel{\dot{\cong}}^{\downarrow 2} T''$, or $T' \mathrel{\dot{\cong}}^{\downarrow 3} T''$ and so forth. Note that we spend some effort in defining the notion of a piece of a target term to allow the stepwise removal of junk. This allows us to consider different kinds of junk separately. If we instead consider all possible junk of a target term at one go, then for the definition of junk it suffices to require that the context $\mathcal{C}$ is such that $\mathcal{C}(J)$ is a target term. However, it seems to be more efficient and more descriptive to consider the different kinds of junk separately.

In the simplest case junk is a closed process that cannot perform any step, i.e., junk is invisible and inactive. Such a kind of junk is produced e.g. as left over of a test-construct. By Definition 5.1.1, a test-construct and the corresponding instantiations of booleans are defined as:

$$\begin{aligned}
\overline{l}\langle\top\rangle &\triangleq l(t,f).\,\overline{t}\\
\overline{l}\langle\bot\rangle &\triangleq l(t,f).\,\overline{f}\\
\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q &\triangleq (\nu t,f)\,\big(\overline{l}\langle t,f\rangle \mid t.P \mid f.Q\big) \qquad \text{for some } t,f \notin \mathsf{fn}(P)\cup\mathsf{fn}(Q)
\end{aligned}$$

If we test a positive instantiation of sum lock we result in the then-case $P \mid (\nu t,f)\,(f.Q)$, else a test results in the else-case $Q \mid (\nu t,f)\,(t.P)$. The terms $(\nu t,f)\,(f.Q)$ or $(\nu t,f)\,(t.P)$ remain as unobservable and inactive junk.

Lemma 6.3.46. *The terms $(\nu t,f)\,(f.Q)$ or $(\nu t,f)\,(t.P)$ are junk.*

Proof. Let $J_1 = (\nu t,f)\,(f.Q)$ and $J_2 = (\nu t,f)\,(t.P)$. J_1 as well as J_2 are closed terms, which cannot perform any step. Moreover, they reach neither success nor translated observables, i.e., $J_i \not\Downarrow_{\checkmark}$, $J_i \not\Downarrow^1_{\mu}$, $J_i \not\Downarrow^2_{\mu}$, and $J_i \not\Downarrow^3_{\mu}$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$ and all $1 \leq i \leq 2$. Because of that, for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ we have $\mathcal{C}(J_1) \mathrel{\dot{\approx}}^{\downarrow 1} \mathcal{C}(0) \mathrel{\dot{\approx}}^{\downarrow 1} \mathcal{C}(J_2)$ and similarly for the other encodings. Thus, by Definition 6.3.44, J_1 and J_2 are junk. □

Note that, due to this lemma, we can securely omit the left over of test-constructs in the following. Another kind of unobservable and inactive junk is produced by the translation of the empty sum 0. It results in a positive instantiation of a sum lock that is not used anywhere. However, let us generalise this case a little bit to an arbitrary instantiation of a sum lock (either positive or negative) that is not used anywhere.

Lemma 6.3.47. *For any name l the term $(\nu l)\left(\overline{l}\langle z\rangle\right)$, where $z \in \{\top, \bot\}$, is junk.*

Proof. Let $J = (\nu l)\left(\overline{l}\langle z\rangle\right)$. J is a closed term, which cannot perform any step. Moreover, this term can reach neither success nor any translated observable, i.e., $J \not\Downarrow_{\surd}$ and $J \not\Downarrow^3_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. So for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}}$ we have $\mathcal{C}(J) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}(0)$. The argumentation for the other encodings is similar. Thus, by Definition 6.3.44, J is junk. □

Note that this lemma shows that the translation of the empty sum is junk, i.e., we translate nothing into nothing but junk. Moreover we use it to reduce the left over of emulations. In the following lemma we prove, that requests on negative instantiations of sum locks are junk of the encodings $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Note that in this case we consider junk that is potentially observable and active.

Lemma 6.3.48. *Requests on negative instantiations of sum locks are junk.*

Proof. By Lemma 6.3.10, a negative instantiation on a sum lock never becomes positive and all tests on a negatively instantiated sum lock result in the respective else-case. By Lemma 6.3.25, interactions with J_1 or J_2 do not cause deadlocks and all (nested) test-constructs that are unguarded due to an interaction with J_1 or J_2 test the negatively instantiated sum lock. By Lemma 6.2.53, all else-cases of a single or nested test-construct reduce the respective (nested) test-construct to some kind of forwarder that consumes one or two instantiations of sum locks and, by Lemma 6.3.25, eventually unguards new instantiations on the consumed locks with exactly the same values. So, J_1 and J_2 do neither influence the reachability of success nor the reachability of translated observables. Thus for all contexts $\mathcal{C}_1([\cdot]), \mathcal{C}_2([\cdot]) \in \mathcal{P}^{=}_{\mathrm{a}} \to \mathcal{P}^{=}_{\mathrm{a}}$, such that $\mathcal{C}_1(J_1), \mathcal{C}_2(J_2) \in \mathfrak{P}\left(\mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}\right)$, we have $\mathcal{C}_1(J_1) \Downarrow^3_\mu$ iff $\mathcal{C}_1(0) \Downarrow^3_\mu$ and $\mathcal{C}_2(J_2) \Downarrow^3_\mu$ iff $\mathcal{C}_2(0) \Downarrow^3_\mu$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. All steps that result from an interaction with J_1 and J_2 are administrative steps or impure steps that behave as administrative steps, because they do not change a sum lock instantiation. By Lemma 6.3.40, then $\mathcal{C}_1(J_1) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}_1(0)$ and $\mathcal{C}_2(J_2) \;\dot{\approx}^{\downarrow 3}\; \mathcal{C}_2(0)$.

The argumentation for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is similar. □

Next we show, that—for all encodings—encoded guarded terms linked to negative instantiations of sum locks are junk as well. Note that such encoded guarded terms linked to negative instantiations of sum locks result from encoded sums of which already one branch was used to emulate a source term step.

Lemma 6.3.49. *Let $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, or $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$.*
Then $(\nu l)\left(\prod_{i\in I} [\![\, \pi_i.P_i \,]\!] \mid \overline{l}\langle\bot\rangle\right)$ is junk.

Proof. Let $J = (\nu l)\left(\prod_{i\in I} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$. By Lemma 6.3.10, there is no positive instantiation of l and the negative instantiation cannot be changed by the context into a positive instantiation. By Lemma 6.3.22 and Definition 6.3.28, J has no translated observables. Moreover, because of the guards π_i, J has no unguarded occurrence of $\checkmark$ and cannot reach some on its own, i.e., $J \not\Downarrow_{\checkmark}$. By Figure 5.8, J can perform a step on its own if for some $j \in I$ the guard π_j is equal to τ. By Lemma 6.3.10, $\overline{l}\langle\bot\rangle \mid [\![\, \tau.P_j \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ eventually reduces to $\overline{l}\langle\bot\rangle$. Because of that, we can ignore all $[\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ for $\pi_i = \tau$, i.e., they are junk. Let $J' = (\nu l)\left(\prod_{i\in I,\pi_i\neq\tau} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$.

Then $J' \not\longmapsto$. By Lemma 6.2.53, there are no unguarded (replicated) inputs on free names of J' and all free outputs are requests. If the index set $\{\, i \mid i \in I \wedge \pi_i \neq \tau \,\}$ is empty, we can apply Lemma 6.3.47. By Lemma 6.3.48 all requests of J are junk, because of the negative instantiation of the sum lock. As a consequence, no interaction with J or J' can influence the ability of the context to reach success or translated observables and no step that results from an interaction with J or J' influences the state of the context modulo $\dot{\approx}^{\downarrow 3}$. Thus for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$, such that $\mathcal{C}(J) \in \mathfrak{P}\left(\mathcal{P}_{\mathrm{a}}^{\overline{=}\uparrow_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}}\right)$, we have $(\mathcal{C}(J) \Downarrow_{\checkmark}$ iff $\mathcal{C}(\mathsf{0}) \Downarrow_{\checkmark})$ and $\left(\mathcal{C}(J) \Downarrow^3_{\mu}\right.$ iff $\left.\mathcal{C}(\mathsf{0}) \Downarrow^3_{\mu}\right)$ for all $\mu \in \mathcal{N} \cup \overline{\mathcal{N}}$. Moreover no step of J on its own or that results from an interaction with J does influence the state of the context modulo $\dot{\approx}^{\downarrow 3}$. So, $\mathcal{C}(J) \dot{\approx}^{\downarrow 3} \mathcal{C}(\mathsf{0})$.

The argumentation for the other encodings is similar. □

Analysing the encoding function we observe that the input on a receiver lock of an encoded input guarded source term, that guards a test-statement, is a replicated input. Of course, such a test-statement can only be used once to emulate a source term step. After such an emulation this replicated input becomes junk.

Lemma 6.3.50. *All replicated inputs on receiver locks r such that there is a negative instantiation of a sum lock l and some input request that carries r and l as parameter are junk in $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$.*

Proof. Note that, because of the Lemmata 6.3.17, 6.3.18, 6.3.21, 6.3.22, and 6.3.25, the sum locks introduced by translations of replicated inputs are never tested and thus cannot become false. By Lemma 6.3.25, all instantiations of r are caused by the mentioned input request and hence contain l. Because of the negative instantiation of l, we can complete the proof by repeating the argumentation of the proofs of Lemma 6.3.48 and Lemma 6.3.49. □

As a consequence also the corresponding outputs on receiver locks are junk.

Lemma 6.3.51. *For all receiver locks r the terms $[y' = y]\,\overline{r}\langle l_r, l, l, s, z\rangle$ (contained in procRightOutReq) and $[y' = y]\,\overline{r}\langle l_s, l, l_s, s, z\rangle$ (contained in procRightInReq) are junk of $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$, if there is a negative instantiation of the sum lock l.*

For all receiver locks r_2 the terms $y\cdot i(l_r, r_1, r_2)\,.\overline{r_2}\langle l_r, l, l, s_2, z, r_1, s_1\rangle$ (contained in procRightOutReq) and $y\cdot o(l_s, s_1, s_2, z)\,.\overline{r_2}\langle l_s, l, l_s, s_2, z, s_1, r_1\rangle$ (contained in procRightInReq) are junk of $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{p}}$, if there is a negative instantiation of the sum lock l.

Proof. Note that if these terms are omitted then the processing of left requests in the respective member of a chain of right requests reduces to the forwarder $m_i \twoheadrightarrow m_{i,up}$ or $m_o \twoheadrightarrow m_{o,up}$.

In the second case, by Lemma 6.3.25 and Lemma 6.3.22, there is an input request that carries r and l as parameter. Hence, by Lemma 6.3.50 and Lemma 6.3.18, all matching (replicated) inputs are junk. Because of that, also the output is junk.

In the first case, by Lemma 6.3.25 and Lemma 6.3.22, there is an output request that carries l as parameter. We conclude by repeating the argumentation of the proof of Lemma 6.3.48.

By a similar argumentation, also the respective outputs on receiver locks r_2 are junk of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$. Since the inputs on the polyadic channels $y \cdot i$ and $y \cdot o$ do not lead to translated observables, we can remove also them. □

Unfortunately, we cannot declare any left over of emulations as junk, because we cannot ignore the forwarding of left requests in the chains of right requests which is left over by formerly considered right requests. However, after removing the junk as described by the above lemma, there is indeed nothing more left than simple forwarders, which cannot influence the state of the process modulo $\dot{\approx}^{\downarrow 2}$ or $\dot{\approx}^{\downarrow 3}$. That suffice to prove operational completeness.

6.3.6. Semantic Criteria

Among the semantic criteria operational correspondence is the most elaborate to prove. Therefore we show both its conditions, operational completeness and operational soundness, separately. In order to show operational completeness, we have to show how source terms steps are emulated by the encodings and that all terms left over by a emulations are junk. Hence, operational completeness follows for all three encodings by an induction on the number and kind of source term steps and the Lemmata of Section 6.3.5. Because of the complexity of the emulations the proofs are rather long. Because of that we postpone the proofs of operational completeness to the appendix in Section A.2.

Lemma 6.3.52 (Operational Completeness)**.** *The encoding* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ *satisfies operational completeness.*

Because $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is an intermediate step in order to obtain $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$, the proof of operational completeness of these two encodings are quite similar. We present the more complex case of $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ in Section A.2.

Lemma 6.3.53 (Operational Completeness)**.** *The encodings* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *satisfy operational completeness.*

We observe that for each emulation there is exactly one core step, i.e., there is exactly one core step for each of the rules Pi-Tau$_{\mathrm{m,s}}$, Pi-Com$_{\mathrm{m,s}}$, and Pi-Rep$_{\mathrm{m,s}}$ and the emulation of the remaining rules does not introduce additional core steps. Note that the connection between emulations and core steps is already shown by the first two conditions of the invariant in Lemma 6.3.25. This underpins our intuition of core steps

(see Section 6.3.1). Any emulation of a source term step is connected to exactly one core step. Moreover, any core step marks exactly one emulation of a source term step by steering the emulation to the "point of no return", i.e., to a point from where no other sequence of steps can disable the completion of that emulation and from where any possibility to emulate a conflicting source term step is ultimately withdrawn. Based on this we prove operational soundness, by showing that each target term is part of an emulation of some source term step. To do so, we have to reason backwards, i.e., we have to conclude from the steps—in particular core steps—of the target terms on the structure of the source term.

Lemma 6.3.54 (Operational Soundness)**.** *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *satisfy operational soundness.*

Proof. We start with $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$. By Definition 3.3.4, we have to show that:

$$\forall S \in \mathcal{P}_{\mathrm{s}} \,.\, \forall T \in \mathcal{P}_{\mathrm{a}} \,.\, [\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T$$
$$\text{implies } \exists S' \in \mathcal{P}_{\mathrm{s}} \,.\, \exists T' \in \mathcal{P}_{\mathrm{a}} \,.\, S \Longmapsto S' \wedge T \Longmapsto T' \wedge T' \mathrel{\dot{\cong}^{\downarrow 1}} [\![S']\!]^{\mathrm{s}}_{\mathrm{a}}$$

Note that T is a target term, i.e., $T \in \mathcal{P}_{\mathrm{a}}{\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}}$. By Lemma 6.3.40, administrative steps do not influence the state of a target term modulo $\mathrel{\dot{\cong}^{\downarrow 1}}$, i.e., $\forall T, T' \in \mathcal{P}_{\mathrm{a}}{\restriction_{[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}}} \,.\, T \mathrel{\dot{\Longmapsto}} T'$ implies $T \mathrel{\dot{\cong}^{\downarrow 1}} T'$. Because of that, it suffice to consider impure and core steps, i.e., steps on translated source term names or sum locks. By Lemma 6.3.10 and Lemma 6.2.51, steps on negative instantiations of sum locks reduce the corresponding test-constructs to simple forwarders, that eventually restore the consumed instantiations of sum locks. Thus impure steps that reduce a positive instantiation of a sum lock followed by administrative steps that restore this positive instantiation do not change the state of a target term modulo $\mathrel{\dot{\cong}^{\downarrow 1}}$.

Consider the sequence $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\Longmapsto}} T \overset{\mapsto}{\longmapsto} T' \mathrel{\dot{\Longmapsto}} T''$ for a source term $S \in \mathcal{P}_{\mathrm{s}}$. By Lemma 6.3.40, all target terms in the sequence $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\Longmapsto}} T$ (including T) are congruent to $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ modulo $\mathrel{\dot{\cong}^{\downarrow 1}}$. Similarly, all target terms in the sequence $T' \mathrel{\dot{\Longmapsto}} T''$, including T', are congruent to T'' modulo $\mathrel{\dot{\cong}^{\downarrow 1}}$. By Definition 6.3.3, Lemma 6.3.25, and Lemma 6.3.10, the core step unguards some encoded source term by reducing a (nested) test-construct to its then-case. By Lemma 6.2.51, each single test-construct results from the translation of a source term guarded by τ or a replicated source term input and each nested test-construct results from the translation of a source term input. Moreover, the respective encoded source term parts are guarded in the encoding iff the parts are guarded in the source. By Lemma 6.2.51, the test-construct has to result from the encoding of a τ-prefix, because in the other cases the test-constructs are guarded by an input or replicated input on a translated source term name. By Lemma 6.3.25, this subterm is not guarded in S, because else $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\Longmapsto}} T$ would need another core step to unguard the encoding of this subterm. Hence, $S \equiv (\nu \tilde{x}) \left(\sum_{i \in I} \pi_i.S_i \mid S' \right)$ for some sequence of names $\tilde{x}$, a finite index set I, some terms $\pi_i.S_i, S' \in \mathcal{P}_{\mathrm{s}}$, and there is some $j \in I$ such that $\pi_j = \tau$. Then $S \longmapsto (\nu \tilde{x}) \, (S_j \mid S')$. By Lemma 6.2.53, the source term part

that is unguarded by the core step is $[\![S_j]\!]^{\mathrm{s}}_{\mathrm{a}}$. Moreover, by Lemma 6.3.10, the administrative step consumes the positive instantiation of the sum lock introduced by the encoding of $\sum_{i\in I} \pi_i.S_i$ and eventually a negative instantiation is unguarded. Hence, by Lemma 6.3.40 and Lemma 6.3.39, all target terms in the sequence $T' \stackrel{\cdot}{\Longmapsto} T''$, including T', are congruent to $[\![(\nu\tilde{x})\,(S_j \mid S')]\!]^{\mathrm{s}}_{\mathrm{a}}$ modulo $\dot{\cong}^{\downarrow 1}$.

To capture the case of single test-construct that results from the translation of a replicated source term, we have to add an impure step to the above sequence. Hence, consider $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \stackrel{\cdot}{\Longmapsto} T_1 \stackrel{\cdot/\mapsto}{\longmapsto} T_2 \stackrel{\cdot}{\Longmapsto} T_3 \stackrel{\mapsto}{\longmapsto} T_4 \stackrel{\cdot}{\Longmapsto} T_5$. Again, by Lemma 6.3.40, all target terms in the sequence $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \stackrel{\cdot}{\Longmapsto} T_1$ are congruent to $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ modulo $\dot{\cong}^{\downarrow 1}$, all target terms in the sequence $T_2 \stackrel{\cdot}{\Longmapsto} T_3$ are congruent to T_2 modulo $\dot{\cong}^{\downarrow 1}$, and all target terms in the sequence $T_4 \stackrel{\cdot}{\Longmapsto} T_5$ are congruent to T_4 modulo $\dot{\cong}^{\downarrow 1}$. Since the considered impure step is the first impure step it cannot reduce the positive instantiation of a sum lock, because all nested test-constructs are initially guarded by an input on a translated source term name. Hence, the impure step is a step on a translated source term name. Let us ignore for the moment that the impure step and the core step are not necessarily related, e.g. the impure step can reduce an input on a translated source term name that guards a nested test-construct and the core step can be as like in the previous case. We consider such cases later. Hence, the impure step reduces a replicated input on a translated source term name $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ which after some $\stackrel{\cdot}{\longmapsto}$-steps unguards a single test-construct that results from the translation of a replicated input in S and the administrative step reduces this test-construct. By Lemma 6.3.19 and Lemma 6.3.25, then $S \equiv (\nu\tilde{n})\,\big(y^{\star}(x)\,.S_1 \mid \sum_{i\in I} \pi_i.S_{2,i} \mid S_3\big)$ for some index set I such that $\pi_j.S_{2,j} = \overline{y}\langle z\rangle\,.S_2$ for some $j \in I$, some names x, z, a sequence of names $\tilde{n}$, and some terms $S_1, S_{2,i}, S_3 \in \mathcal{P}_{\mathrm{s}}$. Hence, $S \longmapsto (\nu\tilde{n})\,(y^{\star}(x)\,.S_1 \mid \{\, z/x \,\}\, S_1 \mid S_2 \mid S_3)$. By Lemma 6.3.10, Lemma 6.3.12, and Lemma 6.3.19, the reduction of the positive sum lock after some $\stackrel{\cdot}{\longmapsto}$-steps leads to a negative instantiation of the sum lock, an instantiation of a sender locks that unguards after another $\stackrel{\cdot}{\longmapsto}$-step the encoding of S_2 in T_4 and the encoding of S_1 in T_4. By Lemma 6.3.49 and Lemma 6.3.45, the negative instantiation of the sum lock ensures that we can remove junk. Hence, $T_4 \dot{\cong}^{\downarrow 1} [\![(\nu\tilde{n})\,(y^{\star}(x)\,.S_1 \mid \{\, z/x \,\}\, S_1 \mid S_2 \mid S_3)]\!]^{\mathrm{s}}_{\mathrm{a}}$.

For the case of nested test-constructs we add another impure step. Hence, consider $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \stackrel{\cdot}{\Longmapsto} T_1 \stackrel{\cdot/\mapsto}{\longmapsto} T_2 \stackrel{\cdot}{\Longmapsto} T_3 \stackrel{\cdot/\mapsto}{\longmapsto} T_4 \stackrel{\cdot}{\Longmapsto} T_5 \stackrel{\mapsto}{\longmapsto} T_6 \stackrel{\cdot}{\Longmapsto} T_7$. Again, by Lemma 6.3.40, all target terms in the sequence $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \stackrel{\cdot}{\Longmapsto} T_1$ are congruent to $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ modulo $\dot{\cong}^{\downarrow 1}$, all target terms in the sequence $T_2 \stackrel{\cdot}{\Longmapsto} T_3$ are congruent to T_2 modulo $\dot{\cong}^{\downarrow 1}$, all target terms in the sequence $T_4 \stackrel{\cdot}{\Longmapsto} T_5$ are congruent to T_4 modulo $\dot{\cong}^{\downarrow 1}$, and all target terms in the sequence $T_6 \stackrel{\cdot}{\Longmapsto} T_7$ are congruent to T_6 modulo $\dot{\cong}^{\downarrow 1}$. Moreover, let us assume that the core step reduces a nested test-construct. Then the first impure step is on a translated source term name $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$ and (modulo $\stackrel{\cdot\cdot}{\longmapsto}$-steps) unguards the nested test-construct, the second impure step consumes the first positive instantiation of the tested sum locks, and the core step consumes the second positive instantiation. By Lemma 6.3.19 and Lemma 6.3.25, then $S \equiv (\nu\tilde{n})\,\big(\sum_{i\in I_1} \pi_i.S_{1,i} \mid \sum_{i\in I_2} \pi_i.S_{2,i} \mid S_3\big)$ for some index sets I_1, I_2 such that $\pi_j.S_{1,j} = y(x)\,.S_1$ and $\pi_k.S_{2,k} = \overline{y}\langle z\rangle\,.S_2$ for some $j \in I_1, k \in I_2$, some

names x, z, a sequence of names $\tilde{n}$, and some terms $S_{1,i}, S_{2,j}, S_3 \in \mathcal{P}_{\mathrm{s}}$. Hence, $S \longmapsto (\nu\tilde{n})\,(\{\, {}^{z}/_{x} \,\}\, S_1 \mid S_2 \mid S_3)$. By Lemma 6.3.10, Lemma 6.3.12, and Lemma 6.3.19, the reduction of the positive sum locks after some $\overset{\cdot}{\longmapsto}$-steps leads to negative instantiations of the sum locks, an instantiation of a sender locks that unguards after another $\overset{\cdot}{\longmapsto}$-step the encoding of S_2 in T_6 and the encoding of S_1 in T_6. By Lemma 6.3.49 and Lemma 6.3.45, the negative instantiations of the sum locks ensure that we can remove junk. Hence, $T_6 \mathrel{\dot{\cong}}^{\downarrow 1} [\![\, (\nu\tilde{n})\,(\{\, {}^{z}/_{x} \,\}\, S_1 \mid S_2 \mid S_3) \,]\!]^{\mathrm{s}}_{\mathrm{a}}$.

We conclude by an induction on the number of impure and core steps in the sequence $[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} \longmapsto\!\!\!\!\Rightarrow T$. All occurring cases can be derived from one of the three cases above or from combinations of these cases. The latter is necessary to show that for each sequence that starts more than a single emulation attempt eventually all emulation attempts are either completed or aborted. Note that this is always possible, because by Lemma 6.3.25 all consumed sum locks are eventually restored and thus the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ does not introduce deadlock. Intuitively, after restoring all consumed instantiations of a sum lock the target term is always in some kind of stable state (see e.g. [PS92]) that can directly be compared to the state of a source term. Moreover, note that aborted emulation attempts do, by Lemma 6.3.25, not result from core steps and all impure steps behave as administrative steps.

The argumentation for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is similar to the argumentation for $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ above. Note that in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ there are no steps on translated source term names, so there are less impure steps. Also note that in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ we cannot directly reason backwards from the reduction of a (nested) test-construct to the look of the source term but have to perform an intermediate step by reasoning over the requests. All necessary information to do so are covered by the invariants in Lemmata 6.3.25, 6.3.17, 6.3.18, 6.3.21, 6.3.22. □

The lemmata for operational completeness and soundness together imply operational correspondence. It remains to show that the encodings reflect divergence and are sensitive to success. For the former we rely on the observation that for each core step in the target there is some step in source.

Lemma 6.3.55 (Divergence Reflection)**.** *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *reflect divergence.*

Proof. By Lemma 6.3.25 and as shown in the proof of operational soundness, for all core steps there is a step of the source term. Thus we have to show that there is no infinite sequence of administrative and impure steps between two core steps. The argumentation for the sequence of administrative steps before the first core step and for the sequence of administrative steps after the last core step—in the case of a terminating process—is then similar.

Auxiliary Links: Auxiliary links are used to unfold the polyadic communications on sum locks, sender locks, receiver locks, output requests, input requests, and chain locks of type $\sharp(\mathfrak{o}, \mathfrak{i})$. By Lemma 6.2.39, Lemma 6.2.56, and Theorem 6.2.54, there are infinitely many steps on auxiliary links only if there are infinitely many steps on the other links.

Sender Locks: Note that each target term has a finite representation. Hence, because of Lemma 6.3.12, there are modulo $\overset{\cdot\cdot}{\longmapsto}$-steps only finitely many sender locks in each target term. Moreover, new sender locks do only result from emulations of steps on replicated source term inputs. By Lemma 6.3.25, such emulations require core steps. Hence for each sequence of steps between two core steps, there are only finitely many different sender locks. By Lemma 6.3.12 there is at most one input for each sender lock in the considered sequence and thus there are only finitely many steps on sender locks.

Chain Locks of Type $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ and $\sharp(\mathfrak{o},\mathfrak{i})$: By Lemma 6.3.18 and Lemma 6.3.24, instantiations of chain locks of type $\updownarrow^{\omega}_{*}(\mathfrak{v}_{\mathfrak{n}})$ result (like instantiations on sender locks) from core steps. Hence, between two core steps there are only finitely many instantiations on these chain locks that unguard only finitely many inputs on chain locks of type $\sharp(\mathfrak{o},\mathfrak{i})$ that in turn unguard only finitely many branches that result from emulations of steps on replicated source term inputs. Hence, there are only finitely many steps on these chain locks.

Requests and Chain Locks of Type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ and $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$: By Lemma 6.3.22 there are only finitely many different requests in each target term. Requests are copied to be pushed upwards in the parallel structure of the source term, i.e., between two core steps there can be more than one step on each request. By Lemma 6.3.22 and Lemma 6.3.25 the flow of requests within the target term is guided by the maintained parallel structure. Since, this is finite requests cannot be pushed infinitely often upwards. By the above case, requests can also only be forwarded to finitely many branches of chains of encoded continuations of replicated source term inputs. Because of that there are only finitely many requests that are right or left to such a branch or encodings of parallel operators. By Lemma 6.3.24 and Lemma 6.3.25, there are only finitely many instantiations chain locks of type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ and $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$ induced by the actual state of the maintained parallel structure. Hence, only finitely many right requests can be added to a chain of such requests. Hence there are only finitely many steps on requests and only finitely many steps on chain locks of type $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{i}_{\circ}))$ and $\updownarrow^{\omega}_{*}(\downarrow_{*}(\mathfrak{o}_{\circ}))$.

Receiver Locks: By Lemma 6.3.25 and Lemma 6.3.18 there is at most one instantiation of each receiver lock for each combination of left and right requests. Hence by the case before, there are only finitely many instantiations of receiver locks and because of this only finitely many steps on receiver locks.

Sum Locks: For each target term there are only finitely many unguarded test-constructs. By the above case and Lemma 6.3.18, only finitely new test-constructs can be unguarded by steps on receiver locks within a sequence between two core steps. By Lemma 6.3.10, there are only finitely many steps on sum locks that are not core steps.

Booleans: First, it may be necessary to complete the consumptions of some positive instantiations of sum locks that are reduced by former core steps. But since there

cannot be infinitely many sum locks and thus, by Lemma 6.3.10, not infinitely many instantiations on sum locks, only finitely many steps are necessary to complete these consumptions. By the case above, there are only finitely many new reductions on sum locks. Hence, by Lemma 6.3.10, there are only finitely many steps on booleans.

The argumentation for the other encodings is similar. □

Success sensitiveness basically follows from operational correspondence.

Lemma 6.3.56 (Success Sensitiveness)**.** *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *are success sensitive.*

Proof. We start with $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. By Definition 3.3.6, we have to show that $S \Downarrow_{\checkmark}$ iff $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow_{\checkmark}$ for all $S \in \mathcal{P}_{\mathrm{m}}$. Let $S \in \mathcal{P}_{\mathrm{m}}$ be an arbitrary source term. We prove both directions separately.

By Definition 3.2.2 and Pi-Cong$_{\mathrm{m,s,a,p}}$, $S \Downarrow_{\checkmark}$ implies that there is some $S' \in \mathcal{P}_{\mathrm{m}}$ such that $S \Longrightarrow S' \mid \checkmark$. By operational completeness, the sequence $S \Longrightarrow S' \mid \checkmark$ can be emulated by $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$, i.e., there exists some $T \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}}$ such that $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Longrightarrow T$ and $T \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S' \mid \checkmark]\!]^{\mathrm{m}}_{\mathrm{a}}$. By Figure 5.8, the encodings of the subterms of a parallel operator are guarded within the encoded parallel operator iff these subterms are guarded within the source term parallel operator. Hence, $[\![S' \mid \checkmark]\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow_{\checkmark}$. Because $\stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$ is sensitive to success, $[\![S' \mid \checkmark]\!]^{\mathrm{m}}_{\mathrm{a}} \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} T$ implies $T \Downarrow_{\checkmark}$. Since $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Longrightarrow T$, we conclude $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow_{\checkmark}$.

By Definition 3.2.2, $S \not\Downarrow_{\checkmark}$ implies that either there is no occurrence of $\checkmark$ in S or all such occurrences are guarded and there is no execution of S that unguards one of them. In the first case, by Figure 5.8, there is no $\checkmark$ in $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$, i.e., $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \not\Downarrow_{\checkmark}$. In the other case, let $a \notin \mathsf{n}(S) \cup \mathsf{n}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}})$ and let S' be the result of replacing all occurrences of $\checkmark$ by $\overline{a}$. Hence, $S \Downarrow_{\checkmark}$ iff $S' \Downarrow_{\overline{a}}$. Let $b = \varphi^{\mathrm{m}}_{\mathrm{a}}(a)$. By Figure 5.8, Definition 6.3.28, Lemma 6.3.10, Lemma 6.3.12, and Lemma 6.3.25, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow_{\checkmark}$ iff $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}} \Downarrow^{3}_{\overline{b}}$. By Lemma 6.3.3, $S' \not\Downarrow_{\overline{a}}$ implies $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}} \not\Downarrow^{3}_{\overline{b}}$. Thus, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \not\Downarrow_{\checkmark}$.

The argumentation for the other encodings is similar. □

By the lemmata above all three encodings satisfy all requirements for a good encoding as they are presented in Section 3.3.

Theorem 6.3.57. *The encodings* $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, *and* $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ *are good.*

Proof. By Observation 6.1.1, all three encodings are compositional. By Corollary 6.1.5, they are name invariant. Operational correspondence follows from the Lemmata 6.3.52, 6.3.53, and 6.3.54. By Lemma 6.3.55, all three encodings reflect divergence. And, by Lemma 6.3.56, they are success sensitive. □

6.4. Domain-Specific Criteria

In Sections 6.1 and 6.3 we proved correctness of the three encodings with respect to the general framework of Gorla as described in Section 3.3. In this section we analyse the

encodings with respect to the additional criterion on the preservation of distributability formalised in Section 3.4. More precisely, we prove that $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ preserve distributability, whereas $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ does not.

As shown in Lemma 3.4.4, the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves distributability because it translates restriction and the parallel operator homomorphically and preserves (enough of) the structural congruence of source terms.

Lemma 6.4.1. *The encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves distributability.*

Proof. By Definition 3.4.3, $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves distributability if for every source term S and for all terms $S_1, \ldots, S_n$ that are distributable within S there are some target terms $T_1, \ldots, T_n$ that are distributable within $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ and are related modulo $\dot{\cong}^{\downarrow 1}$ to the encoded distributable source term parts, i.e., if $T_i \dot{\cong}^{\downarrow 1} [\![S_i]\!]^{\mathrm{s}}_{\mathrm{a}}$ for all $1 \leq i \leq n$. Note that, by Definition 3.4.2, empty processes $\mathsf{0}$ are not considered as distributable.

Accordingly, consider an arbitrary source term $S \in \mathcal{P}_{\mathrm{s}}$. By Definition 3.4.2, S is distributable into $S_1, \ldots, S_n \in \mathcal{P}_{\mathrm{s}}$ if there exists a sequence of names $x_1, \ldots, x_m$ such that $S \equiv (\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)$. By the argumentation in the proof of Lemma 6.3.41 and because $\mathsf{0}$ is not distributable, $S \equiv (\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)$ implies $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv [\![(\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)]\!]^{\mathrm{s}}_{\mathrm{a}}$. Moreover, by Figure 5.1, we have

$$\begin{aligned}[\![(\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)]\!]^{\mathrm{s}}_{\mathrm{a}} &= (\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(x_1), \ldots, \varphi^{\mathrm{s}}_{\mathrm{a}}(x_m))\,([\![S_1 \mid \ldots \mid S_n]\!]^{\mathrm{s}}_{\mathrm{a}}) \\ &= (\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(x_1), \ldots, \varphi^{\mathrm{s}}_{\mathrm{a}}(x_m))\,([\![S_1]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \ldots \mid [\![S_n]\!]^{\mathrm{s}}_{\mathrm{a}})\,.\end{aligned}$$

Thus, $[\![S]\!]^{\mathrm{s}}_{\mathrm{a}}$ is distributable into $[\![S_1]\!]^{\mathrm{s}}_{\mathrm{a}}, \ldots, [\![S_n]\!]^{\mathrm{s}}_{\mathrm{a}}$. Since obviously $[\![S_i]\!]^{\mathrm{s}}_{\mathrm{a}} \dot{\cong}^{\downarrow 1} [\![S_i]\!]^{\mathrm{s}}_{\mathrm{a}}$ for all $1 \leq i \leq n$, the encoding $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves distributability. □

The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ does not translate the parallel operator homomorphically. Instead it implements a protocol within the encoding of the parallel operator that allows for the combination of requests. However, a closer look reveals that the encoded subterms of the parallel operator still appear unguarded and that all additional terms introduced within the encoding of a parallel operator are guarded by a replicated input. Hence, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ is distributable, because within the pi-calculus replicated input is distributable as discussed in Section 3.4.

Lemma 6.4.2. *The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ preserves distributability.*

Proof. Consider an arbitrary source term $S \in \mathcal{P}_{\mathrm{m}}$. By Definition 3.4.2, S is distributable into $S_1, \ldots, S_n \in \mathcal{P}_{\mathrm{m}}$ if there exists a sequence of names $x_1, \ldots, x_m$ such that $S \equiv (\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)$. By the argumentation in the proof of Lemma 6.3.42, structural congruence of source terms is preserved for all rules except for commutativity and associativity of the parallel operator and $P \mid \mathsf{0} \equiv P$. In particular the structural congruence rules that allow to pull restriction outwards are preserved. Let $S' \in \mathcal{P}_{\mathrm{m}}$ be such that $S \equiv (\nu x_1, \ldots, x_m)\,S' \equiv (\nu x_1, \ldots, x_m)\,(S_1 \mid \ldots \mid S_n)$ and S' and $S_1 \mid \ldots \mid S_n$ differ only by applications of the structural congruence rules for commutativity and associativity of the parallel operator and $P \mid \mathsf{0} \equiv P$. Then $S \equiv (\nu x_1, \ldots, x_m)\,S'$ implies

$[\![S]\!]^{\mathrm{m}}_{\mathrm{p}} \equiv [\![(\nu x_1, \ldots, x_m)\, S']\!]^{\mathrm{m}}_{\mathrm{p}} = \left(\nu \varphi^{\mathrm{m}}_{\mathrm{p}}(x_1), \ldots, \varphi^{\mathrm{m}}_{\mathrm{p}}(x_m)\right) [\![S']\!]^{\mathrm{m}}_{\mathrm{p}}$. By Definition 3.4.2, $\mathsf{0}$ is not distributable. Hence, $S' \equiv S_1 \mid \ldots \mid S_n$ can be obtained without the rule $P \mid \mathsf{0} \equiv P$. Since this rule is the only structural congruence rule that allows for the introduction of additional parallel operators, we conclude that S' and $S_1 \mid \ldots \mid S_n$ contain the same number of parallel operators and differ only in the order of parallel components and their structure among each other. By Figure 5.4 and Figure 5.5,

$$
\begin{aligned}
[\![P \mid Q]\!]^{\mathrm{m}}_{\mathrm{p}} \triangleq\ & (\nu p_{o,up}, p_{i,up}, o, i)\,\big(\\
& (\nu p_o, p_i)\,\big([\![P]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big) \\
& \mid (\nu p_o, p_i)\,\big([\![Q]\!]^{\mathrm{m}}_{\mathrm{p}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big) \\
& \mid \mathsf{pushReq}\big)
\end{aligned}
$$

and all terms in $\mathsf{procLeftOutReq}$, $\mathsf{procLeftInReq}$, $\mathsf{procRightOutReq}$, $\mathsf{procRightInReq}$, and $\mathsf{pushReq}$ are guarded by a replicated input. Remember that replicated input in the pi-calculus is distributable, i.e., we can use the additional rule $y^\star(x).P \equiv y^\star(x).P \mid y^\star(x).P$ to derive the distributable components of a term. We conclude that all encoded source terms with respect to $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ are distributable into the encodings of the parallel components of the respective source term. Hence $\left(\nu \varphi^{\mathrm{m}}_{\mathrm{p}}(x_1), \ldots, \varphi^{\mathrm{m}}_{\mathrm{p}}(x_m)\right) [\![S']\!]^{\mathrm{m}}_{\mathrm{p}}$ is distributable into $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{p}}, \ldots, [\![S_n]\!]^{\mathrm{m}}_{\mathrm{p}}$. Since obviously $[\![S_i]\!]^{\mathrm{m}}_{\mathrm{p}} \overset{\cdot}{\leftrightarrows}{}^{\downarrow 2}_{\mathrm{c}} [\![S_i]\!]^{\mathrm{m}}_{\mathrm{p}}$ for all $1 \leq i \leq n$, the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ preserves distributability. □

In contrast, Theorem 4.2.18 and Theorem 4.4.8 show that $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ cannot preserve distributability, because by Theorem 6.3.57 it is a good encoding. Intuitively, the problem are the chains of right requests in the encoding of the parallel operator. Remember that they are introduced to avoid divergence or deadlock but as shown in the following example they forbid for the preservation of distributability.

Example 6.4.3. Consider the counterexample $S = \left(\overline{a} \mid \overline{b}\right) \mid (a \mid b)$. $S \equiv S_1 \mid S_2$ with $S_1 = (\overline{a} \mid a)$ and $S_2 = \left(\overline{b} \mid b\right)$ such that $S_1 \longmapsto \mathsf{0}$ and $S_2 \longmapsto \mathsf{0}$. Of course, since $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is good, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ can emulate both steps in either order. But it cannot emulate both truly in parallel.

The encoding of S is given by:

$$
\begin{aligned}
[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} = \ & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i)\,\big(\\
& (\nu p_o, p_i)\,\Big([\![\overline{a} \mid \overline{b}]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\Big) \\
& \mid (\nu p_o, p_i)\,([\![a \mid b]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \mid \mathsf{pushReq}\big)
\end{aligned}
$$

We observe that for both source term steps their emulations require that the corresponding requests are combined at the outermost parallel operator encoding, i.e., the one given above. Moreover, in both emulations the output requests arrive at the left and the input requests arrive at the right hand side of this parallel operator encoding. Thus, for both

emulations the requests are combined by procRightInReq.

$$\begin{aligned}\mathsf{procRightInReq} = \overline{c_i}\langle m_o\rangle \mid c_i{}^{\star}(m_o)\,.p_i(y, l_r, r)\,.\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \\ \mid (\nu m_{o,up})\,\big(\, m_o{}^{\star}\big(y', l_s, s, z\big)\,.\,\big([y' = y]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big) \\ \mid (\nu m_o)\,(m_{o,up} \twoheadrightarrow m_o \mid \overline{c_i}\langle m_o\rangle)\,\big)\big)\end{aligned}$$

We observe that the main part of procRightInReq is guarded by a replicated input and can thus be distributed as for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ in the lemma above. But there is only one unguarded instantiation of the corresponding chain lock $\overline{c_i}\langle m_o\rangle$ and without it the remaining term guarded by the replicated input on this chain lock is useless. Note that if we provide two such outputs initially we obtain two chains of right input requests and cannot ensure any more that all left and right requests of opposite kind can be combined. Because of that, procRightInReq is not distributable. Hence, the encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ does not preserve distributability.

Because of that, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ cannot emulate both source term steps without sequentialising the linking of the respective right requests within the required chain. So $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$ cannot completely emulate the independent source term steps in parallel.

6.5. Summary and Related Work

Within this chapter we discussed how to validate an encoding in general and in particular we proved correctness of the encodings introduced in the last chapter. We started with the structural criteria. Compositionality restricts the look of the encoding function and can thus be checked by observation. Also name invariance follows in some settings directly from the look of an encoding. In Section 6.1 we show that this criterion holds whenever the encoding function makes strict use of the renaming policy and preserves the binding on names.

In contrast, the proof of the remaining criteria is very elaborated, because of the complexity of the encoding functions. In Section 6.2 we make use of different type systems, to abbreviate some of the proofs. The type system in [Mil93b] can be considered as the first type system for the pi-calculus. Based on this type system and one of the type systems in [SW01] we introduced a basic type system in Section 6.2.2 to capture the distinction of links introduced for different purposes by the encoding functions. The main advantage of this is that we can unambiguously identify particular links and sum up links that are introduced for the same purpose. In Section 6.2.3 we discussed type systems for the monadic variants of the pi-calculus—as the target languages of the considered languages—and by the way demonstrated that types can also be used to express some kind of "behaviour" as long as it is predictable. We reviewed the type systems introduced in [Yos96, QW00, QW05] to type unfoldings of polyadic communications and adapt them to our setting. These type systems describe some predetermined sequence of consecutive states for some of the types. The actual state of the type evolves during the derivation following the syntax tree of a term. Of course, this kind of behavioural types can also be used to prove more advanced properties. In Section 6.2.4 we proved some properties of links using polarities and multiplicities. Polarities are introduced in [PS96],

[KPT99] consider linearity, and receptiveness is introduced in [San97, San99]. As result we were able to prove partial confluence for some of the links that are introduced by the encoding functions. Partial confluence significantly reduces the state space that has to be considered in proofs on process terms. A discussion on how type systems can be used to establish confluence can be found in [Nes96, NS97].

Nestmann in [Nes00] uses a type system to prove deadlock-freedom of $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ (or the original variant of this encoding as presented in [Nes00]). Therefore he adapts a type system of [Kob98] that introduces reliable links.[4] Reliability means that every input or output on such a channel eventually finds its communication partner. Nestmann proves that all channels introduced by the encoding function $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ are reliable. Thus the encoding does not introduce deadlock. Intuitively, Kobayashi extends earlier type systems with multiplicities and polarities by obligations that capture information about the dependencies between the usages of different links. In [Nes00] this suffices to show that all potential sources of deadlock result from links that are direct translations of the source term links. However, because $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ uses translated source term names only as values and relies on requests for the identification of source term communication partners, this type system cannot be easily extended to cover deadlock-freedom in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$. Instead, we used invariants in Section 6.3.2 to prove that the encodings do not introduce deadlock.

Also note that all type systems introduced in Section 6.2 are type systems for variants of the pi-calculus. Hence, the presented typing rules and instructions for typing a term are specific to the pi-calculus. However, there are also type systems for other process calculi as e.g. in [CG99, CGG99] for the mobile ambient calculus [CG98].

Then in Section 6.3 we proved the quality of the three encodings and also discussed some related properties. We distinguished between administrative steps, i.e., pre- and postprocessing steps of an encoding, and core steps that represent the entire translation of a source term step. Moreover, we discussed the presence of intermediate states resulting from partial commitment. As already explained in [NP00], partially committed states that result from partial emulations of steps are a frequent side effect of choice encodings. Intuitively, the problem is that the choice-constructs in the source language allow to reduce several different capabilities simultaneously and that the target terms can not mimic all these reductions within a single step. The consequence is an intermediate state, i.e., a state that cannot be directly related to one of the source term states. Of course such intermediate states do not only result from encodings of choice in the pi-calculus but occur whenever the target language (or at least the target terms of the respective encodings) have to split up effects of a single source term step onto several target term steps. Note that intermediate states do not immediately rule out the possibility of a good encoding, but they make it harder to prove correctness of the respective encoding. In Section 6.3.1 we denote steps that (potentially) lead to intermediate states—i.e.,partially committed states—as impure steps and explain why they usually need special consideration. Also note that some kind of intermediate states may indeed influence the possibility to satisfy some particular criteria. So the non-existence of distributability-preserving good encodings in Section 4.3 and Section 4.4 rely on the

[4] In [Kob06] Kobayashi extends this type system to cover recursion.

fact that the respective target languages are not able to translate a source term step such that all the conflicts it rules out simultaneously in the source are also decided within a single step in the target.

In [Nes96] Nestmann uses factorisation as a technique to improve the presentation of an encoding function and also to structure the proofs of its correctness. Moreover, factorisation is combined with decoding functions in order to obtain an equivalence and to prove full abstraction. We also use intermediate encodings and intermediate languages—the polyadic variants of the encodings—but do not use these to obtain an equivalence. Note that we also do not try to prove full abstraction, although we believe that such a result can be obtained by a suitable adaptation of the coupled simulation used in [Nes96]. Instead we rely on the quality criteria introduced by Gorla in [Gor10b]. Because of that, we gain additional flexibility in the choice of the equivalence, because in the general framework the burden on the proof is not on the equivalence. Moreover, we discussed junk. Junk can appear in different forms. While unobservable and inactive junk does usually not influence the properties of an encoding, observable or active junk may forbid for some good properties. All three considered encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ introduce observable junk which prevents the use of equivalences on standard observables on the respective target language. We showed how nonetheless quality of these encodings can be proved with respect to the general framework. For this we introduced the notion of translated observables, i.e., the translation of source term observables, and show how they can be used to obtain a suitable equivalence or congruence. Moreover, the explicit formalisation of translated observables provides further intuition on the encoding functions and visualises the connection to the related source terms. As we can observe in Section 6.3 this connection helps to guide the proofs of correctness.

Finally we consider distributability in Section 6.4, i.e., the additional domain-specific criterion defined in Section 3.4 to measure preservation of distributability of an encoding function. As expected $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves distributability, because it translates the parallel operator homomorphically. The encoding $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$ does not translate the parallel operator homomorphically but preserves nonetheless distributability. This shows that the criterion on the preservation of distributability that we formalise in Section 3.4 is indeed weaker as the commonly used criterion on the homomorphic translation of the parallel operator.

The main contribution of this chapter is the proof that all three encodings $[\![\cdot]\!]^{\mathrm{s}}_{\mathrm{a}}$, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{p}}$, and $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ satisfy all five criteria of the general framework of Gorla as presented in Section 3.3. We also discussed some drawbacks of these encodings. In particular we showed that they introduce observable junk and thus do not allow for the use of standard equivalences to show operational correspondence. In Chapter 5 we also discussed the use of the match prefix in $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ and conjecture that there can not exists a good encoding from π_{m} into π_{s} or π_{a} that does not make use of match. Moreover, $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ does not preserve distributability. This is not due to an awkward design of the encoding function but a general limitation of encodings on mixed choice. As shown by Theorems 4.2.18 and 4.4.8, there exists no good encoding of mixed choice that preserves distributability. Nonetheless the above encodability result for $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$ is meaningful and important. It is the first encodability result of this kind and provides further intuition on

the relationship between mixed choice and separate choice in particular and synchronous and asynchronous interactions in general. For the latter it basically sums up the positive results obtained in [HT92, Bou92, Nes96, NP00, Nes00] in this direction. Moreover, it is crucial for the hierarchy derived in Section 7.2, because it tells us in what π_{m} and $\pi_{\mathrm{a}}^{=}$ do *not* differ.

7. Concluding Remarks

We conclude with an overview of the obtained results in Section 7.1. Furthermore we combine our results in order to obtain a hierarchy on different levels of distributability between the considered synchronous and asynchronous variants of the pi-calculus and the join-calculus in Section 7.2. In Section 7.3 we discuss further research.

7.1. Contributions

In Chapter 3 we proposed a new criterion to measure preservation of distributability of encoding functions. In Chapter 4 we proved several absolute and translational separation results. Moreover, we presented two encodings of mixed choice in Chapter 5 and proved their correctness in Chapter 6.

Breaking Symmetries. We proved without any further assumption that the pi-calculus with separate choice lacks—in contrast to the pi-calculus with mixed choice—the ability to break initial symmetries. For this reason, we stated an absolute separation result proving that mixed choice is strictly more expressive than separate choice. Moreover, since homomorphic translation of the parallel operator preserves initial symmetries, this absolute result turned out to be well suited to prove several translational separation results for respectively different definitions of reasonableness. These results support the conjecture that there is no reasonable encoding from the pi-calculus with mixed choice into the pi-calculus with separate choice that translates the parallel operator homomorphically, where the notion of reasonableness covers at least reflection of divergence and deadlock, and some suitable notion of preservation of behaviour. Moreover, by comparing the presented translational separation results we observe that: (1) the absolute separation result plays a central role in each presented translational result, (2) the use of the absolute separation result allows us to weaken the assumptions under which the translational separation results hold in comparison to earlier proposals, (3) the use of the absolute separation result induces an intuitive way to prove quite different translational separation results. In summary, these arguments emphasize the central role of absolute separation results for language comparison. Note that even with the help of match, the pi-calculus with separate choice can not break symmetries and there is no uniform and reasonable encoding from the pi-calculus with mixed choice into the pi-calculus with separate choice or the asynchronous pi-calculus.

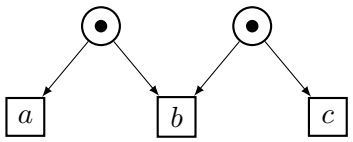
As shown in [Pal03], leader election serves to derive a translational result, but even input-output confluence suffices to separate mixed from separate choice by an absolute result. Absolute separation results like confluence, leader election, and breaking symmetries can be used to obtain translational separation results. Therefore, typically an

example is chosen that illustrates the main discriminating features of the absolute result and that can be used in the translational separation result as counterexample. To do so, the main features of this example have to be preserved by the encoding function, i.e., by the criteria required for the encoding. Thus confluence is not an adequate choice to derive a translational separation result as the above, since it is very difficult to find a discriminating counterexample based on confluence; even if such an example is found, it is intricate to argue for the preservation of its properties. In this sense leader election is much more suitable, because its main conditions are preserved under uniform encodings that preserve substitutions. However, breaking symmetries is even better suited because its properties are preserved by weaker requirements on reasonable encodings. Accordingly, confluence can be considered as a too weak property, while leader election is a little bit too specific. In short, breaking symmetries serves as a "sweet spot". It allows for the formulation of a general result: there is no reasonable encoding from the pi-calculus with mixed choice into the pi-calculus with separate choice that translates the parallel operator homomorphically, where the notion of reasonableness covers at least reflection of divergence and deadlock, and some suitable notion of preservation of behaviour.

Preservation of Distributability. In order to measure whether an encoding preserves distributability, usually the homomorphic translation of the parallel operator is used as a criterion (see e.g. [Pal03, CM03, LV10]). Such an encoding naturally preserves the parallel structure of terms and thus (at least for process calculi like CSP or the pi-calculus) the degree of distributability. However, the opposite is not true. We presented an encoding that preserves distributability although it does not translate the parallel operator homomorphically. In this sense, the homomorphic translation of the parallel operator is too strict—at least for separation results. It rightly forbids the introduction of coordinators that reduce the degree of distributability. But it also forbids protocols that handle communications of parallel components without sequentialising them or reducing the degree of distributability in another sense. Moreover, the homomorphic translation of the parallel operator is not always suited to reason about distributability in process calculi as e.g. the join-calculus in that distributable subterms are not necessarily separated by a parallel operator. To overcome this problem, we proposed a novel criterion and showed that is well-suited to reason about distributability-preservation of good encodings. In particular our new criterion is weaker than the common homomorphic translation of the parallel operator and can be applied also to reason about distributability in process calculi as the join-calculus in that distributability is not necessarily the same as parallel. Moreover, we showed that distributability of processes implies also distributability of executions. This leads to a new proof method for separation results.

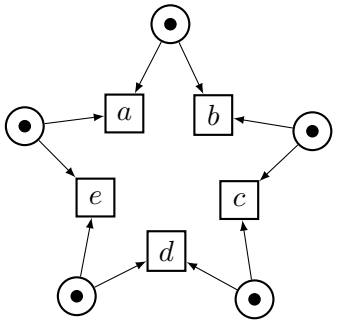
Then we proved that even under this weaker requirement no good encoding from π_{m} into π_{s} can preserve distributability, because—to avoid deadlock and divergence—each encoding of mixed choice has to sequentialise some steps of the emulation of some steps that are distributable in the source term. As a consequence every good encoding of mixed choice leads to additional causal dependencies.

Synchronisation Pattern. By adapting the synchronisation pattern M



proposed in [vGGS08, vGGS12] to reason about distributability, we were able to clarify the difference between the asynchronous pi-calculus and the join-calculus. Note that the pi-calculus is a well-known and frequently used process calculus to model concurrent systems. But practical experience has shown that it is not possible to implement every pi-calculus term—not even every asynchronous one—in an asynchronous setting while preserving its degree of distributability. To overcome these problems, the join-calculus was introduced as a model of distributed computation [Lév97]. The synchronisation pattern M describes a conflict between two distributable processes. The process can either act concurrently—reflected by the actions a and c—or they can decide to synchronise—on the action b. To implement this pattern within a fully asynchronous and distributed setting we would have to ensure that the distributable processes always coincide in their decision to either act concurrently or to synchronise. If we rely on the distributability of these processes such an implementation is in general not possible without accepting the possibility of deadlock or divergence. Because of that, by proving the impossibility to express the pattern M in the join-calculus, we presented a formal argument for the distributability of the join-calculus[1]. Moreover, we proved that the asynchronous pi-calculus is not distributable and that there can not exist a good encoding from the asynchronous pi-calculus into the join-calculus that preserves distributability.

To shed more light on the difference between the synchronous pi-calculus with mixed choice and its fragment with only separate choice we extended the synchronisation pattern M to the synchronisation pattern $\star$ that can be visualised by the Petri net:



Again this pattern describes conflicts between distributable processes, but here a circle of such processes of odd length is described. Remarkably, this pattern reflects a well-known standard problem in the area of distributed systems, namely the problem of the dining philosophers. Although the situation described by the pattern $\star$ does not look

[1] Note that we follow [vGGS08, vGGS12] if we consider calculi in which the above synchronisation pattern M is not expressible as intuitively distributable.

that complicated and despite the imposing expressive power of the pi-calculus with separate choice, we proved that it cannot express the above conflicts between distributable processes—not even modulo weak reductions. This distinguishes separate choice from mixed choice, where examples for the pattern $\star$ can be easily obtained by combining conflicts on input and output capabilities over mixed choices. We elucidated this difference by proving that there exists no good and distributability-preserving encoding.

With exemplary results in the context of CSP, we show that the presented proof method, based on synchronisation patterns, can also be applied to obtain separation results in other process calculi. Thereby the general, i.e., mainly model-independent, formulation of the above synchronisation patterns in terms of conditions on step-transition systems enables the use of these patterns to reason about and to classify synchronisation mechanisms in other process calculi.

Encoding Mixed Choice. Learning from the separation results in Chapter 4 we derived a good encoding of mixed choice, i.e., an encoding from the synchronous pi-calculus with mixed choice into the asynchronous pi-calculus without choice that is compositional, name invariant, satisfies operational correspondence, divergence reflection, and success sensitiveness. From the "breaking symmetry" results we learned that no good encoding of mixed choice can translate the parallel operator homomorphically and that the encoding function has to break initial symmetries of the source term. Hence, by abandoning the condition of homomorphic translation of the parallel operator in favour of compositionality, we proposed an encoding from the synchronous pi-calculus with mixed choice into the asynchronous pi-calculus that, as we proved, meets all five of Gorla's criteria. As a novelty, our new encoding overcomes the previous non-compositional attempts to break symmetries globally by providing a principle that breaks symmetries locally, saving true compositionality. Moreover, we analysed different properties and also drawbacks of the proposed encoding. As drawback we consider the necessity of the match prefix for the encoding of choice and the introduction of observable junk. As discussed, we believe that both drawbacks can not be avoided, i.e., that there exists no good encoding of mixed choice into the asynchronous pi-calculus without the match prefix and that every good encoding of mixed choice introduces observable junk. Moreover, we proved that no good encoding of mixed choice preserves distributability or reflects causal dependencies. Despite these limitations our positive translational result reveals new insights onto the relationship of the synchronous and asynchronous pi-calculus. The existence of this good encoding shows that—with respect to their abstract behaviour—the synchronous pi-calculus with mixed choice is *as expressible as* the asynchronous pi-calculus (without choice). This is in particular important for the hierarchy that we consider in the next section.

7.2. Hierarchy of Distributability in Pi-like Calculi

In Chapter 4 we presented several translational separation results between the full synchronous pi-calculus with mixed choice (π_{m}) and its—also synchronous—variant with

only separate choice (π_s). The most intuitive of these results is surely the last one in Section 4.4. It shows that no good encoding from π_m into π_s can preserve distributability. To separate these two languages the synchronisation pattern $\star$ was used. As explained, synchronisation pattern provide not only a method to obtain counterexamples for separation results, but in fact describe a particular kind or effect of synchronisation and can thus be used to classify the synchronisation mechanisms of a calculus. In a similar manner the asynchronous pi-calculus (π_a) and the join-calculus (J) were separated by a simpler synchronisation pattern, the pattern M, in Section 4.3. Again the non-existence of a good and distributability-preserving encoding was shown.

In Section 6.4 we proved that the encoding from π_s into π_a proposed in [Nes00] preserves distributability. Moreover, we showed that this encoding, which was proved correct in [Nes00] with respect to another setting of quality criteria, also satisfies the criteria of the general framework (Section 3.3). Hence this encoding is a good encoding from π_s into π_a that preserves distributability. Since π_s is a subcalculus of π_m and π_a is a subcalculus of π_s, there are obviously good and distributability-preserving encodings from π_s into π_m and from π_a into π_s. Remember that we consider the join-calculus as an asynchronous variant of the pi-calculus. In fact we can easily prove that the encoding from J into π_a proposed in [FG96] is a good encoding and does also preserve distributability.

In contrast, our encoding from π_m into π_a is good but does not preserve distributability. Note that [FG96] introduce also an encoding from π_a into J. But this encoding is defined in two-levels and is thus not compositional. However, as we discussed in Section 5.5, there are good reasons to nonetheless accept this encoding as good and, moreover, we believe that by adapting some concepts of the encoding from π_m into π_a this encoding can be turned into an encoding from π_a into J that satisfies again all the requirements of the general framework.

Combining these positive results and these negative translational results on the two synchronisation patterns, we obtain a hierarchy of distributability between pi-like calculi. The synchronous pi-calculus with mixed choice (π_m), the synchronous pi-calculus with separate choice (π_s), the asynchronous pi-calculus (π_a), and the join-calculus (J) all have the same expressive power, but there exists no good and distributability-preserving encoding from π_m into π_a, and neither from π_a into J.

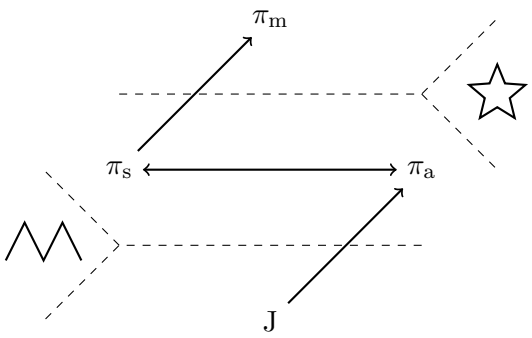


Figure 7.1.: Distributability in Pi-like Calculi.

In Figure 7.1 the mentioned positive and negative translational results are visualised by lines connecting the respective source and target language. Moreover, positive results are visualised by arrows from the source into the target language. Hence negative results are illustrated by the absence of an arrow head in the corresponding direction. Moreover, the existence of negative results is illuminated by dashed lines that visualise the borderline between the expressive power of the considered variants and that are augmented with a pictogram of the respective distinguishing synchronisation pattern. Note that there are of course also good encodings from J into π_{s} and π_{m} and from π_{a} into π_{m}.

Note that to obtain this hierarchy, i.e., to obtain the picture in Figure 7.1, we need the negative translational separation results and the good and distributability-preserving encoding from π_{s} into π_{a} of [Nes00]. However, to tell what this hierarchy is talking about, i.e., to ensure that it indeed shows different levels of distributability, we also need the other mentioned positive results, because they prove that preservation of distributability is in fact the only discriminating feature between the considered languages.

The hierarchy in Figure 7.1 tells us that the considered variants of the pi-calculus and the join-calculus represent three different levels on the expressive power of their synchronisation mechanisms with respect to distributability. The borderlines between these levels are described by the possibility or impossibility to express the synchronisation patterns M and $\star$ as visualised in Figure 7.1. Interestingly, the asynchronous pi-calculus resides on the same level as the synchronous pi-calculus with only separate choice. This shows that in order to benefit from the additional expressive power that is usually assumed with synchronous interactions we also have to include the expressive power of mixed choice.

7.3. Further Research

Of course we do not believe that the two synchronisation patterns that lead to the hierarchy in Figure 7.1 already capture all kinds of synchronisation mechanisms in process calculi. In further research we want to analyse e.g. what kind of synchronisation patterns are expressed by polyadic synchronisation in [CM03] or by the synchronisation mechanisms described in [LV10]. Also the study of distributability in extensions of the calculi e.g. by mechanisms to express real-time or failures is an interesting area of further research.

In the case of separation results, a natural next step to improve the results is to go back to particular distributions, in order to examine the problematic set of distributable terms in the source language. This way a positive result for a sublanguage of the source language can be derived. An exhaustive analysis may even lead to an exact borderline between distributable and not distributable languages. Note that the results in [vGGS12] go in this direction for the area of Petri nets. This kind of consideration is beyond the scope of the this thesis, but another interesting topic of further research.

Another direction of further research is the analysis of synchronisation mechanisms in other process calculi and other concurrency formalisms in general, and a comparison of the obtained results. As explained, the synchronisation pattern M that allows us to

separate π_a from J originates from a similar consideration in the context of Petri nets. In fact we can observe several similarities but also differences between our results and results obtained in the context of Petri nets. As an example consider the discussion in [SPG11] and [PSN11]. An analysis of these similarities and differences as well as the consideration of further concurrency formalisms may provide further insights on the nature of different synchronisation mechanisms and may even lead to some model independent answers on questions related to the implementability of synchronisation mechanisms in distributed real systems.

In Section 5.4 we discuss the composition of encoding functions. As Gorla ([Gor10a]) we find it annoying that the composition of two good encodings does not necessarily lead to a good encoding. Hence, we agree with him that the revision of the quality criteria such that they result in less demanding conditions for the composition of good encodings, is a challenging but also very interesting direction for future research.

Furthermore, the lengthy proofs in the Appendix of this thesis show the need of mechanical proof assistance. The formalisation of the pi-calculus within Isabelle/HOL [NWP02] can provide a starting point in this direction [BP07, Ben10]. Hence extending this formalisation by the concept of encodings and quality criteria as well as proof methods for the derivation of correctness proofs as for example type systems is also an interesting topic of further research.

List of Figures

Bibliography

[ACS98] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. [An extended abstract appears in the proceedings of CONCUR '96.].

[AFV01] Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural Operational Semantics. *Handbook of Process Algebra*, pages 197–292, 2001.

[AH92] S. Arun-Kummar and Matthew Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29(8):737–760, 1992.

[AM96] Luca Aceto and David Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4):317–350, 1996.

[Bae05] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2–3):131–146, 2005.

[BB90] Gerard Berry and Gerard Boudol. The Chemical Abstract Machine. In *Proceedings of POPL (Principles of Programming Languages)*, SIGPLAN-SIGACT, pages 81–94. ACM, 1990.

[BD12] Eike Best and Philippe Darondeau. Petri Net Distributability. In *Proceedings of PSI '11 (Perspectives of Systems Informatics)*, volume 7162 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012.

[Ben10] Jesper Bengtson. *Formalising process calculi*. PhD thesis, Acta Universitatis Upsaliensis, 2010.

[BG95] Nadia Busi and Roberto Gorrieri. Distributed Conflicts in Communicating Systems. In *Object-Based Models and Languages for Concurrent Systems*, volume 924 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 1995.

[BGZ00] Nadia Busi, Roberto Gorrieri, and Gianluigi Zavattaro. On the Expressiveness of Linda Coordination Primitives. *Information and Compututation*, 156(1–2):90–121, 2000.

[BK82] Jan A. Bergstra and Jan W. Klop. Fixed point semantics in process algebra. Technical Report IW 206/82, Mathematical Centre, Amsterdam, 1982.

[Bou88] Luc Bougé. On the Existence of Symmetric Algorithms to Find Leaders in Networks of Communicating Sequential Processes. *Acta Informatica*, 25(4):179–201, 1988.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus (note). Note, INRIA, 1992.

[BP91] Frank S. Boer and Catuscia Palamidessi. Embedding as a tool for Language Comparison: On the CSP hierarchy. In *Proceedings of CONCUR (Concurrency-Theory)*, volume 527 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 1991.

[BP07] Jesper Bengtson and Joachim Parrow. Formalising the π-Calculus Using Nominal Logic. In *Proceedings of FoSSaCS (Foundations of Software Science and Computational Structures)*, volume 4423 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2007.

[BPV05] Michael Baldamus, Joachim Parrow, and Björn Victor. A Fully Abstract Encoding of the π-Calculus with Data Terms (Extended Abstract). In *Proceedings of ICALP (International Colloquium on Automata, Languages and Programming)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1202–1213. Springer, 2005.

[Bru97] Glenn Bruns. *Distributed Systems Analysis with CCS*. Prentice-Hall, 1997.

[BS83] Gael Norma Buckley and Abraham Silberschatz. An Effective Implementation for the Generalized Input-Output Construct of CSP. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(2):223–235, 1983.

[BS98] Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Informatica*, 35(5):353–400, 1998.

[CCP07] Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. Separation of synchronous and asynchronous communication via testing. *Theoretical Computer Science*, 386(3):218–235, 2007.

[CCP09] Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. Explicit fairness in testing semantics. *Logical Methods in Computer Science*, 5(2):1–27, 2009.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. In *Proceedings of FoSSaCS (Foundations of Software Science and Computation Structures)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CG99] Luca Cardelli and Andrew D. Gordon. Types for Mobile Ambients. In *Proceedings of POPL (Principles of Programming Languages)*, SIGPLAN-SIGACT, pages 79–92. ACM, 1999.

[CGG99] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility Types for Mobile Ambients. In *Proceedings of ICALP (International Colloquium on Automata, Languages and Programming)*, volume 1644 of *Lecture Note in Computer Science*, pages 230–239. Springer, 1999.

[CM03] Marco Carbone and Sergio Maffeis. On the Expressive Power of Polyadic Synchronisation in π-Calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[CMT96] Bernadette Charron-Bost, Friedmann Mattern, and Gerard Tel. Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4):173–191, 1996.

[DDNM88] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. On the Consistency of "Truly Concurrent" Operational and Denotational Semantics. In *Proceedings of LICS (Logic in Computer Science)*, pages 133–141. IEEE, 1988.

[Dij71] E. W. Dijkstra. Hierarchical Ordering of Sequential Processes. *Acta Informatica*, 1(2):115–138, 1971.

[EN86] Uffe H. Engberg and Mogens Nielsen. A Calculus of Communicating Systems with Label Passing. Technical Report DAIMI PB-208, University of Aarhus, 1986.

[EN00] Uffe H. Engberg and Mogens Nielsen. A Calculus of Communicating Systems with Label Passing—Ten Years After. *Proof, Language, and Interaction; Essays in Honour of Robin Milner*, pages 599–622, 2000.

[FG96] Cédric Fournet and Georges Gonthier. The Reflexive CHAM and the Join-Calculus. In *Proceedings of POPL (Principles of Programming Languages)*, SIGPLAN-SIGACT, pages 372–385. ACM, 1996.

[FL10] Yuxi Fu and Hao Lu. On the expressiveness of interaction. *Theoretical Computer Science*, 411(11-13):1387–1451, 2010.

[FLMR96] Gonthier Georges Fournet, Cédric, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A Calculus of Mobile Agents. In *Proceedings of CONCUR (Concurrency Theory)*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 1996.

[Fok07] Wan Fokkink. *Modelling Distributed Systems.* Springer, 2007.

[Fou98] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, L'École Polytechnique, 1998.

[Gar82] Hector Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, 31(1):48–59, 1982.

[GMR06] Jan F. Groote, Mohammad R. Mousavi, and Michel A. Reniers. A Hierarchy of SOS Rule Formats. In *Proceedings of SOS '05 (Structural Operational Semantics)*, volume 156 of *Electronic Notes in Theoretical Computer Science*, pages 3–25. Elsevier, 2006.

[Gor07] Daniele Gorla. Synchrony vs Asynchrony in Communication Primitives. In *Proceedings of EXPRESS '06 (Expressivity in Concurrency)*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 87–108. Elsevier, 2007.

[Gor08a] Daniele Gorla. Comparing Communication Primitives via their Relative Expressive Power. *Information and Computation*, 206(8):931–952, 2008.

[Gor08b] Daniele Gorla. Towards a Unified Approach to Encodability and Separation Results for Process Calculi. In *Proceedings of CONCUR (Concurrency Theory)*, volume 5201 of *Lecture Notes in Computer Science*, pages 492–507. Springer, 2008.

[Gor09] Daniele Gorla. On the Relative Expressive Power of Calculi for Mobility. In *Proceedings of MFPS (Mathematical Foundations of Programming Semantics)*, volume 249 of *Electronic Notes in Theoretical Computer Science*, pages 269–286. Elsevier, 2009.

[Gor10a] Daniele Gorla. A taxonomy of process calculi for distribution and mobility. *Distributed Computing*, 23(4):273–299, 2010.

[Gor10b] Daniele Gorla. Towards a Unified Approach to Encodability and Separation Results for Process Calculi. *Information and Computation*, 208(9):1031–1053, 2010.

[HBS73] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proceedings of IJCAI (International Joint Conference on Arteficial Intelligence)*, pages 235–245. ACM, 1973.

[Hen07] Matthew Hennessy. *A Distributed Pi-Calculus*. Cambridge University Press, 2007.

[Hoa78] Sir Charles Antony Richard Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoa04] Sir Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, June 21 2004. Electronic version of Communicating Sequential Processes, first published in 1985 by Prentice Hall International.

[Hon92a] Kohei Honda. Notes on Soundness of a Mapping from π-calculus to ν-calculus. With comments added in October 1993, May 1992.

[Hon92b] Kohei Honda. Two bisimilarities in ν-calculus. CS Report 92-002, Keio University, 1992. Revised on March 31, 1993.

[HP01] Oltea Mihaela Herescu and Catuscia Palamidessi. On the generalized dining philosophers problem. In *Proceedings of PODC (Principles of Distributed Computing)*, pages 81–89. ACM, 2001.

[HP05] Oltea Mihaela Herescu and Catuscia Palamidessi. A Randomized Encoding of the π-Calculus with Mixed Choice. *Theoretical Computer Science*, 335(2–3):373–404, 2005.

[HT91] Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. In *Proceedings of ECOOP (European Conference on Object-Oriented Programming)*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[HT92] Kohei Honda and Mario Tokoro. On Asynchronous Communication Semantics. In *Object-Based Concurrent Computing*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51. Springer, 1992.

[HY95] Kohei Honda and Nobuko Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[JS85] Ralph E. Johnson and Fred B. Schneider. Symmetry and Similarity in Distributed Systems. In *Proceedings of PODC (Principles of Distributed Computing)*, pages 13–22. ACM, 1985.

[Kie98] Astrid Kiehn. Concurrency in Process Algebras. Habilitationsschrift, 1998.

[Kna93] Frederick Knabe. A Distributed Protocol for Channel-Based Communication with Choice. *Computers and Artificial Intelligence*, 12(5):475–490, 1993.

[Kob98] Naoki Kobayashi. A Partially Deadlock-Free Typed Process Calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 20(2):436–482, 1998. [This is an extended and revised version of the paper presented in proceedings of LICS '97.].

[Kob06] Naoki Kobayashi. A New Type System for Deadlock-Free Processes. In *Proceedings of CONCUR (Concurrency Theory)*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006.

[KPT99] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(5):914–947, 1999. [This is a revised and extended version of a paper presented at POPL '96.].

[Lan07] Ivan Lanese. Concurrent and Located Synchronizations in π-Calculus. In *Proceedings of SOFSEM: Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 388–399. Springer, 2007.

[Lév97] Jean-Jacques Lévy. Some Results in the Join-Calculus. In *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 233–249. Springer, 1997.

[LL77] Gérard Le Lann. Distributed Systems—Towards a Formal Approach. In *Information Processing*, pages 155–160. IFIP, 1977.

[LR81] Daniel Lehmann and Michael O. Rabin. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of POPL (Principles of Programming Languages)*, SIGPLAN-SIGACT, pages 133–138. ACM, 1981.

[LSZ74] Richard J. Lipton, Lawrence Snyder, and Yechezkel Zalcstein. A Comparative Study of Models of Parallel Computation. In *Proceedings of SWAT (Annual Symposium on Switching and Automata Theory)*, pages 145–155. IEEE, 1974.

[LV10] Cosimo Laneve and Antonio Vitale. The Expressive Power of Synchronizations. In *Proceedings of LICS (Logics in Computer Science)*, pages 382–391. IEEE, 2010.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1996.

[Mil80] Robin Milner. A Calculus of Communicating Systems. volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[Mil92] Robin Milner. Functions as Processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil93a] Robin Milner. Elements of Interaction: Turing Award Lecture. *Communications of the ACM*, 36(1):78–89, 1993.

[Mil93b] Robin Milner. The Polyadic π-Calculus: a Tutorial. *Logic and Algebra of Specification*, 94:203–246, 1993.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, New York, 1999.

[MP95] Ugo Montanari and Marco Pistore. Concurrent Semantics for the π-calculus. In *Proceedings of MFPS (Mathematical Foundations of Programming Semantics)*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 411–429, 1995.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I and II. *Information and Computation*, 100(1):1–77, 1992.

[MRG07] Mohammad R. Mousavi, Michel A. Reniers, and Jan F. Groote. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3):238–272, 2007.

[MS92] Robin Milner and Davide Sangiorgi. Barbed Bisimulation. In *Proceedings of ICALP (International Coloquium on Automata, Languages and Programming)*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[Nes96] Uwe Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming*. PhD thesis, Universität Erlangen-Nürnberg, 1996.

[Nes00] Uwe Nestmann. What is a "Good" Encoding of Guarded Choice? *Information and Computation*, 156(1-2):287–319, 2000.

[Nes06] Uwe Nestmann. Welcome to the Jungle: A subjective Guide to Mobile Process Calculi. In *Proceedings of CONCUR Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2006.

[NP00] Uwe Nestmann and Benjamin C. Pierce. Decoding Choice Encodings. *Information and Computation*, 163(1):1–59, 2000. An extended abstract appeared in the *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*.

[NS97] Uwe Nestmann and Martin Steffen. Typing Confluence. In *Proceedings of ERCIM (Formal Methods in Industrial Critical Systems)*, 1997.

[NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[Old87] Ernst-Rüdiger Olderog. Operational Petri Net Semantics for CCSP. In *Advances in Petri Nets*, volume 266 of *Lecture Notes in Computer Science*, pages 196–223. Springer, 1987.

[Pal03] Catuscia Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[Par08] Joachim Parrow. Expressiveness of Process Algebras. *Electronic Notes in Theoretical Computer Science*, 209:173–186, 2008.

[Pet62] Carl A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[Plo04] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60:17–140, 2004. [An earlier version of this paper was published as technical report at Aarhus University in 1981.].

[PN10a] Kirstin Peters and Uwe Nestmann. Breaking Symmetries. In *Proceedings of EXPRESS (Expressiveness in Concurrency)*, volume 41 of *Electronic Proceedings in Theoretical Computer Science*, pages 136–150, 2010.

[PN10b] Kirstin Peters and Uwe Nestmann. Breaking Symmetries. Technical report, Technische Universität Berlin, Germany, 2010. http://arxiv.org/abs/1007.4172v1.

[PN12a] Kirstin Peters and Uwe Nestmann. Breaking Symmetries. *Mathematical Structures in Computer Science*, 2012. To appear.

[PN12b] Kirstin Peters and Uwe Nestmann. Is It a "Good" Encoding of Mixed Choice? In *Proceedings of FoSSaCS (Foundations of Software Science and Computational Structures)*, volume 7213 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 2012.

[PN12c] Kirstin Peters and Uwe Nestmann. Is it a "Good" Encoding of Mixed Choice? Technical report, Technische Universität Berlin, Germany, 2012. http://arxiv.org/abs/1201.1410v1.

[Pra86] Vaughan Pratt. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15:33–71, 1986.

[Pri96] Corrado Priami. *Enhanced Operational Semantics for Concurrency.* PhD thesis, Universita' di Pisa-Genova-Udine, 1996.

[PS92] Joachim Parrow and Peter Sjödin. Multiway Synchronization Verified with Coupled Simulation. In *Proceedings of CONCUR (Concurrency Theory)*, volume 630 of *Lecture Notes in Computer Science*, pages 518–533. Springer, 1992.

[PS94] Joachim Parrow and Peter Sjödin. The Complete Axiomatization of Cs-congruence. In *Proceedings STACS (Symposium on Theoretical Aspects of Computer Science)*, volume 775 of *Lecture Notes in Computer Science*, pages 555–568. Springer, 1994.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. [A preliminary version appeared in proceedings of LICS '93.].

[PSN11] Kirstin Peters, Jens-Wolfhard Schicke, and Uwe Nestmann. Synchrony vs Causality in the Asynchronous Pi-Calculus. In *Proceedings of EXPRESS (Expressiveness in Concurrency)*, volume 64 of *Electronic Proceedings in Theoretical Computer Science*, pages 89–103, 2011. http://arxiv.org/abs/1108.4469v1.

[PT97] Benjamin C. Pierce and David N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 1997.

[QW00] Paola Quaglia and David Walker. On Synchronous and Asynchronous Mobile Processes. In *Proceedings of FoSSaCS (Foundations of Software Science and Computation Structures)*, volume 1784 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2000.

[QW05] Paola Quaglia and David Walker. Types and full abstraction for polyadic π-calculus. *Information and Computation*, 200(2):215–246, 2005.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras—First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[San94] Davide Sangiorgi. An investigation into functions as processes. In *Proceedings of MFPS (Mathematical Foundations of Programming Semantics)*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 1994.

[San97] Davide Sangiorgi. The name discipline of uniform receptiveness. In *Proceedings of ICALP (International Colloquium on Automata, Languages and Programming)*, volume 1256 of *Lecture Notes in Computer Science*, pages 303–313. Springer, 1997.

[San99] Davide Sangiorgi. The name discipline of uniform receptiveness. *Theoretical Computer Science*, 221(1–2):457–493, 1999.

[San09] Davide Sangiorgi. On the Origins of Bisimulation and Coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):Article 15, 2009.

[SPG11] Jens-Wolfhard Schicke, Kirstin Peters, and Ursula Goltz. Synchrony vs. Causality in Asynchronous Petri Nets. In *Proceedings of EXPRESS (Expressiveness in Concurrency)*, volume 64 of *Electronic Proceedings in Theoretical Computer Science*, pages 119–131, 2011. http://arxiv.org/abs/1108.4471v1.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press New York, NY, USA, October 16 2001.

[Tur96] David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Department of Computer Science, Universitiy of Edinburgh, 1996.

[vG93] Rob J. van Glabbeek. The Linear Time – Branching Time Spectrum II. In *Proceedings of CONCUR (Concurrency Theory)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[vG01] Rob J. van Glabbeek. The Linear Time – Branching Time Spectrum I: The Semantics of Conrete, Sequential Processes. *Handbook of Process Algebra*, pages 3–99, 2001.

[vGG01] Rob J. van Glabbeek and Ursula Goltz. Refinement of Actions and Equivalence Notions for Concurrent Systems. *Acta Informatica*, 37(4-5):229–327, 2001.

[vGGS08] Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke. On Synchronous and Asynchronous Interaction in Distributed Systems. In *Proceedings of MFCS (Mathematical Foundations of Computer Science)*, volume 5162 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2008.

[vGGS09] Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke. Symmetric and Asymmetric Asynchronous Interaction. In *Proceedings of ICE (Interaction and Concurrency Experiences)*, volume 229 of *Electronic Notes in Theoretical Computer Science*, pages 77–95. Elsevier, 2009.

[vGGS12] Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann. On Distributability of Petri Nets. In *Proceedings of FoSSaCS (Foundations of Software Science and Computational Structures)*, volume 7213 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 2012.

[VP96] Björn Victor and Joachim Parrow. Constraints as Processes. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR (Concurrency Theory)*, volume 1119 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 1996.

[VPP07] Maria Grazia Vigliotti, Iain Phillips, and Catuscia Palamidessi. Tutorial on separation results in process calculi via leader election problems. *Theoretical Computer Science*, 388(1–3):267–289, December 5 2007.

[Yos96] Nobuko Yoshida. Graph Types for Monadic Mobile Processes. In *Proceedings of FST&TCS (Foundations of Software Technology and Theoretical Computer Science)*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer, 1996.

A. Appendix

A.1. Typed Encoding Functions

A.1.1. Well-Typedness in the Basic Type System

Within this section we present the proofs of Lemma 6.2.19 and Lemma 6.2.18 as well as the missing cases of the proof of Lemma 6.2.17, i.e., we prove that the encodings $\mathcal{T}_{\mathrm{B}}^{1}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, $\mathcal{T}_{\mathrm{B}}^{2}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$, and $\mathcal{T}_{\mathrm{B}}^{3}[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ are well-typed. Note that all three encodings use sum locks and type them by the same type. Because of that, instantiations on sum locks are typed equivalently in the encoding functions.

Lemma A.1.1. *Positive and negative instantiations of a sum lock l that are typed by $\mathcal{T}_{\mathrm{B}}^{1}$, $\mathcal{T}_{\mathrm{B}}^{2}$, or $\mathcal{T}_{\mathrm{B}}^{3}$ are well-typed with respect to $l\colon\mathfrak{l}$.*

Proof. By Figure 6.3, Figure 6.2, and Figure 6.1, sum locks, booleans, and the auxiliary values used in instantiations of sum locks are typed equivalently in $\mathcal{T}_{\mathrm{B}}^{1}$, $\mathcal{T}_{\mathrm{B}}^{2}$, and $\mathcal{T}_{\mathrm{B}}^{3}$. More precisely, for $\mathcal{T} \in \{\, \mathcal{T}_{\mathrm{B}}^{1}, \mathcal{T}_{\mathrm{B}}^{2}, \mathcal{T}_{\mathrm{B}}^{3} \,\}$, we have

$$\mathcal{T}\big(\overline{l}\langle\top\rangle\big) \overset{\text{Def. 5.1.1}}{=} \mathcal{T}\big(l(t,f)\,.\,\overline{t}\big) \overset{\text{Def. 5.4.1}}{=} \mathcal{T}\big(l(t,f)\,.\,(\nu v_t)\,\overline{t}\langle v_t\rangle\big) = l(t,f)\,.\,(\nu v_t\colon\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle$$

for positive and

$$\mathcal{T}\big(\overline{l}\langle\bot\rangle\big) \overset{\text{Def. 5.1.1}}{=} \mathcal{T}\big(l(t,f)\,.\,\overline{f}\big) \overset{\text{Def. 5.4.1}}{=} \mathcal{T}\big(l(t,f)\,.\,(\nu v_f)\,\overline{f}\langle v_f\rangle\big) = l(t,f)\,.\,(\nu v_f\colon\mathfrak{v}_\bot)\,\overline{f}\langle v_f\rangle$$

for negative instantiations of l. Remember that $\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))$ for all three sets of type assignments. Let $\Gamma = l\colon\mathfrak{l}, t\colon\sharp(\mathfrak{v}_\top)\,, f\colon\sharp(\mathfrak{v}_\bot)$. Then,

$$\dfrac{\dfrac{}{l\colon\mathfrak{l}\vdash l\colon\mathfrak{l}}N \quad \dfrac{\dfrac{\dfrac{}{\Gamma, v_t\colon\mathfrak{v}_\top\vdash t\colon\sharp(\mathfrak{v}_\top)}N \quad \dfrac{}{\Gamma, v_t\colon\mathfrak{v}_\top\vdash v_t\colon\mathfrak{v}_\top}N}{l\colon\mathfrak{l}, t\colon\sharp(\mathfrak{v}_\top)\,, f\colon\sharp(\mathfrak{v}_\bot)\,, v_t\colon\mathfrak{v}_\top\vdash \overline{t}\langle v_t\rangle}\text{T-Out}_{\mathrm{B}}}{l\colon\mathfrak{l}, t\colon\sharp(\mathfrak{v}_\top)\,, f\colon\sharp(\mathfrak{v}_\bot)\vdash (\nu v_t\colon\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle}\text{T-Res}_{\mathrm{B}}}{l\colon\sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))\vdash l(t,f)\,.\,(\nu v_t\colon\mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle}\text{T-In}_{\mathrm{B}}$$

and

$$\dfrac{\dfrac{}{l\colon\mathfrak{l}\vdash l\colon\mathfrak{l}}N \quad \dfrac{\dfrac{\dfrac{}{\Gamma, v_f\colon\mathfrak{v}_\bot\vdash f\colon\sharp(\mathfrak{v}_\bot)}N \quad \dfrac{}{\Gamma, v_f\colon\mathfrak{v}_\bot\vdash v_f\colon\mathfrak{v}_\bot}N}{l\colon\mathfrak{l}, t\colon\sharp(\mathfrak{v}_\top)\,, f\colon\sharp(\mathfrak{v}_\bot)\,, v_f\colon\mathfrak{v}_\bot\vdash \overline{f}\langle v_f\rangle}\text{T-Out}_{\mathrm{B}}}{l\colon\mathfrak{l}, t\colon\sharp(\mathfrak{v}_\top)\,, f\colon\sharp(\mathfrak{v}_\bot)\vdash (\nu v_f\colon\mathfrak{v}_\bot)\,\overline{f}\langle v_f\rangle}\text{T-Res}_{\mathrm{B}}}{l\colon\sharp(\sharp(\mathfrak{v}_\top)\,,\sharp(\mathfrak{v}_\bot))\vdash l(t,f)\,.\,(\nu v_f\colon\mathfrak{v}_\bot)\,\overline{f}\langle v_f\rangle}\text{T-In}_{\mathrm{B}}$$

where $N = \text{T-Name}_{\mathrm{B}}$. □

Lemma 6.2.19 states that for all source terms $S \in \mathcal{P}_{\mathrm{s}}$ that are well-structured with respect to $\mathbb{T}_{\mathrm{s}}$ and $\mathcal{T}_S$, the encoding $\mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ is well-typed with respect to $\Gamma_{[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}} = \widetilde{\mathcal{T}_S}$.

Proof of Lemma 6.2.19. An encoding is well-typed if each encoded term is well-typed. Hence, we perform an induction over the structure of (well-structured) source terms $S \in \pi_{\mathrm{s}}$. Note that no source term can contain infinitely many free names. We conclude that $\mathcal{T}_S$ is finite and, thus, also $\Gamma_{[\![\, . \,]\!]^{\mathrm{s}}_{\mathrm{a}}} = \widetilde{\mathcal{T}_S}$ is finite for each source term S.

Base Case: In $\mathcal{P}_{\mathrm{s}}$ there are two terms without subterms, namely 0 and $\checkmark$. Note that $\mathcal{T}_S(0) = 0$ and $\mathcal{T}_S(\checkmark) = \checkmark$. $\mathcal{T}^1_{\mathrm{B}}[\![\, 0 \,]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu l{:}\mathfrak{l})\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right)\right)$ and $\mathcal{T}^1_{\mathrm{B}}[\![\, \checkmark \,]\!]^{\mathrm{s}}_{\mathrm{a}} = \checkmark$. The second case, i.e., the type judgement $\widetilde{\mathcal{T}_S} \vdash \checkmark$, follows directly from T-Succ$_{\mathrm{B}}$. For the first case, we have

$$\dfrac{\dfrac{}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l} \vdash \mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right)}\text{Lemma A.1.1 and Lemma 6.2.11}}{\widetilde{\mathcal{T}_S} \vdash (\nu l{:}\mathfrak{l})\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right)\right)}\text{T-Res}_{\mathrm{B}}$$

Induction Hypothesis:

$$\begin{array}{c}\forall S \in \mathcal{P}_{\mathrm{s}} \,.\, S \text{ is well-structured with respect to } \mathbb{T}_{\mathrm{s}} \text{ and } \mathcal{T}_S \\ \text{implies } \widetilde{\mathcal{T}_S} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}}\end{array} \tag{IH}$$

Induction Step: We perform a case split over the structure of S.

Case $S = (\nu x)\, S'$**:** By Definition 6.2.5 and because $x \in \mathsf{n}(S)$, there is some type assignment $x : T_S \in \mathcal{T}_S$ for some source type $T_S \in \mathbb{T}_{\mathrm{s}}$. Then $\mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} = \left(\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(x){:}\widetilde{T_S}\right)\mathcal{T}^1_{\mathrm{B}}[\![\, S' \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ and

$$\dfrac{\dfrac{}{\widetilde{\mathcal{T}_S} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, S' \,]\!]^{\mathrm{s}}_{\mathrm{a}}}(IH)}{\widetilde{\mathcal{T}_S} \vdash \left(\nu \varphi^{\mathrm{s}}_{\mathrm{a}}(x){:}\widetilde{T_S}\right)\mathcal{T}^1_{\mathrm{B}}[\![\, S' \,]\!]^{\mathrm{s}}_{\mathrm{a}}}\text{T-Res}_{\mathrm{B}}$$

because $\widetilde{\mathcal{T}_S}, \varphi^{\mathrm{s}}_{\mathrm{a}}(x){:}\widetilde{T_S} = \widetilde{\mathcal{T}_S}$.

Case $S = S_1 \mid S_2$**:** Then $\mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} = \mathcal{T}^1_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \mathcal{T}^1_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ and

$$\dfrac{\dfrac{}{\widetilde{\mathcal{T}_S} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}}}IH* \quad \dfrac{}{\widetilde{\mathcal{T}_S} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}}}IH*}{\widetilde{\mathcal{T}_S} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \mathcal{T}^1_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{s}}_{\mathrm{a}}}\text{T-Par}_{\mathrm{B}}$$

where $IH* = (IH)$ and Lemma 6.2.11.

Case $S = \sum_{i \in I} \pi_i.S_i$**:** Then $\mathcal{T}^1_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu l{:}\mathfrak{l})\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right) \mid \prod_{i \in I} \mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)$ and

$$\dfrac{\dfrac{\dfrac{}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l} \vdash \mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right)}\text{Lemma A.1.1 and Lemma 6.2.11} \quad D}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l} \vdash \mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right) \mid \prod_{i \in I} \mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}}\text{T-Par}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S} \vdash (\nu l{:}\mathfrak{l})\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\top\rangle\right) \mid \prod_{i \in I} \mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)}\text{T-Res}_{\mathrm{B}}$$

To prove D, we have to show that $\widetilde{\mathcal{T}_S}, l : \mathfrak{l} \vdash \prod_{i\in I} \mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}$. With the typing Rule T-Par$_{\mathrm{B}}$ we decompose this subgoal into several subgoals of the form $\widetilde{\mathcal{T}_S}, l:\mathfrak{l} \vdash \mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}$, where each π_i is either a τ, an output or an input prefix.

Case $\pi_i = \tau$**:** Then $\mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)$, where the subterms $T_1 = \overline{l}\langle t, f\rangle$, $T_2 = t(v_t).\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right) \mid \mathcal{T}^1_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)$ and $T_3 = f(v_f).\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right)\right)$. Let $\Gamma = \widetilde{\mathcal{T}_S}, l:\mathfrak{l}, t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot)$. Then

$$\dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma \vdash T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l}, t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot) \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l}, t:\sharp(\mathfrak{v}_\top) \vdash (\nu f:\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l} \vdash (\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}$$

Remember that $\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_\top), \sharp(\mathfrak{v}_\bot))$. Hence,

$$D_1 = \dfrac{\dfrac{}{\Gamma \vdash l:\mathfrak{l}}N \quad \dfrac{}{\Gamma \vdash t:\sharp(\mathfrak{v}_\top)}N \quad \dfrac{}{\Gamma \vdash f:\sharp(\mathfrak{v}_\bot)}N}{\Gamma \vdash \overline{l}\langle t, f\rangle}\text{T-Out}_{\mathrm{B}}$$

where $N =$ T-Name$_{\mathrm{B}}$.

$$D_2 = \dfrac{\dfrac{}{\Gamma \vdash t:\sharp(\mathfrak{v}_\top)}\text{T-Name}_{\mathrm{B}} \quad D_2'}{\Gamma \vdash t(v_t).\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right) \mid \mathcal{T}^1_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)}\text{T-In}_{\mathrm{B}}$$

To prove $\Gamma, v_t:\mathfrak{v}_\top \vdash \mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right) \mid \mathcal{T}^1_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}$ for D_2' apply T-Par$_{\mathrm{B}}$ and then Lemma A.1.1 and Lemma 6.2.11 at the left hand side and the induction hypothesis and Lemma 6.2.11 at the right hand side.

$$D_3 = \dfrac{\dfrac{}{\Gamma \vdash f:\sharp(\mathfrak{v}_\bot)}\text{T-Name}_{\mathrm{B}} \quad \dfrac{}{\Gamma, v_f:\mathfrak{v}_\bot \vdash \mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right)}R}{\Gamma \vdash f(v_f).\left(\mathcal{T}^1_{\mathrm{B}}\left(\overline{l}\langle\bot\rangle\right)\right)}\text{T-In}_{\mathrm{B}}$$

where $R =$ Lemma A.1.1 and Lemma 6.2.11.

Case $\pi_i = \overline{y}\langle z\rangle$**:** Then $\mathcal{T}^1_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu s:\mathfrak{s})\,(T_1 \mid T_2)$, where the subterms $T_1 = \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle$ and $T_2 = s(v_s).\mathcal{T}^1_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}$.

$$\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l}, s:\mathfrak{s} \vdash s(v_s).\mathcal{T}^1_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}}\text{T-In}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l}, s:\mathfrak{s} \vdash T_1 \mid T_2}\text{T-Par}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l:\mathfrak{l} \vdash (\nu s:\mathfrak{s})\,(T_1 \mid T_2)}\text{T-Res}_{\mathrm{B}}$$

By Definition 6.2.5 and because $y, z \in \mathsf{fn}(S)$, there exists $T_S \in \mathbb{T}_{\mathrm{s}}$ such that $y : \sharp(T_S), z : T_S \in \mathcal{T}_S$. Then, by Definition 6.2.6, $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$:

$\sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T_S}\right), \varphi_{\mathrm{a}}^{\mathrm{s}}(z) : \widetilde{T_S} \in \widetilde{\mathcal{T}_S}$. Hence, $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s} \vdash \overline{\varphi_{\mathrm{a}}^{\mathrm{s}}(y)}\langle l, s, \varphi_{\mathrm{a}}^{\mathrm{s}}(z)\rangle$ for D_1 follows from T-Out$_{\mathrm{B}}$ and T-Name$_{\mathrm{B}}$ for all remaining subgoals. $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s} \vdash s : \mathfrak{s}$ for D_2 follows from T-Name$_{\mathrm{B}}$ and to prove $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s}, v_s : \mathfrak{v}_{\mathfrak{s}} \vdash \mathcal{T}_{\mathrm{B}}^1[\![S_i]\!]_{\mathrm{a}}^{\mathrm{s}}$ for D_3 apply the induction hypothesis and Lemma 6.2.11.

Case $\pi_i = y(x)$**:** Then:

$$
\begin{aligned}
\mathcal{T}_{\mathrm{B}}^1[\![\pi_i.S_i]\!]_{\mathrm{a}}^{\mathrm{s}} &= (\nu r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\,((\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle \mid T_1)\\
T_1 &= r^{\star}(v_{s,r})\,.\varphi_{\mathrm{a}}^{\mathrm{s}}(y)\left(l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x)\right).T_2\\
T_2 &= (\nu t{:}\sharp(\mathfrak{v}_{\top}), f{:}\sharp(\mathfrak{v}_{\bot}))\,(T_3 \mid T_4 \mid T_8)\\
T_3 &= \overline{l}\langle t, f\rangle\\
T_4 &= t(v_t)\,.\,(\nu t{:}\sharp(\mathfrak{v}_{\top}), f{:}\sharp(\mathfrak{v}_{\bot}))\,(T_5 \mid T_6 \mid T_7)\\
T_5 &= \overline{l'}\langle t, f\rangle\\
T_6 &= t(v_t)\,.\mathcal{T}_{\mathrm{B}}^1\left(\overline{l}\langle\bot\rangle\right) \mid \mathcal{T}_{\mathrm{B}}^1\left(\overline{l'}\langle\bot\rangle\right) \mid (\nu v_s{:}\mathfrak{v}_{\mathfrak{s}})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}_{\mathrm{B}}^1[\![S_i]\!]_{\mathrm{a}}^{\mathrm{s}}\\
T_7 &= f(v_f)\,.\mathcal{T}_{\mathrm{B}}^1\left(\overline{l}\langle\top\rangle\right) \mid \mathcal{T}_{\mathrm{B}}^1\left(\overline{l'}\langle\bot\rangle\right) \mid (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle\\
T_8 &= f(v_f)\,.\mathcal{T}_{\mathrm{B}}^1\left(\overline{l}\langle\bot\rangle\right) \mid \overline{\varphi_{\mathrm{a}}^{\mathrm{s}}(y)}\langle l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x)\rangle
\end{aligned}
$$

Let $\Gamma_1 = \widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. Then

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})}N \quad \dfrac{}{\Gamma_1 \vdash v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}}N}{\Gamma_1 \vdash \overline{r}\langle v_{s,r}\rangle}\text{T-Out}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle}R \quad D_1}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle \mid T_1}\text{T-Par}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l} \vdash (\nu r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}))\,((\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle \mid T_1)}R
$$

where $N =$ T-Name$_{\mathrm{B}}$ and $R =$ T-Res$_{\mathrm{B}}$.

Let $\Gamma_2 = \widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}$. Then

$$
D_1 = \dfrac{D_1' \quad \dfrac{D_1'' \quad D_2}{\Gamma_2 \vdash \varphi_{\mathrm{a}}^{\mathrm{s}}(y)(l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x))\,.T_2}\text{T-In}_{\mathrm{B}}}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, r{:}\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash r^{\star}(v_{s,r})\,.\varphi_{\mathrm{a}}^{\mathrm{s}}(y)(l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x))\,.T_2}\text{T-Rep}_{\mathrm{B}}
$$

where $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, r : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash r : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$ for D_1' follows from T-Name$_{\mathrm{B}}$. By Definition 6.2.5 and because $x, y \in \mathsf{n}(S)$, there exists some source type $T_S \in \mathbb{T}_{\mathrm{s}}$ such that $y : \sharp(T_S), x : T_S \in \mathcal{T}_S$. Then, by Definition 6.2.6, $\varphi_{\mathrm{a}}^{\mathrm{s}}(y) : \sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T_S}\right), \varphi_{\mathrm{a}}^{\mathrm{s}}(x) : \widetilde{T_S} \in \widetilde{\mathcal{T}_S}$. Hence, $\Gamma_2 \vdash \varphi_{\mathrm{a}}^{\mathrm{s}}(y) : \sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T_S}\right)$ for D_1'' follows from T-Name$_{\mathrm{B}}$ and $\Gamma_2, l' : \mathfrak{l}, s : \mathfrak{s}, \varphi_{\mathrm{a}}^{\mathrm{s}}(x) : \widetilde{T_S} = \Gamma_2, l' : \mathfrak{l}, s : \mathfrak{s}$. Let $\Gamma_3 = \Gamma_2, l' : \mathfrak{l}, s : \mathfrak{s} = \widetilde{\mathcal{T}_S}, l, l' : \mathfrak{l}, r : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}}, s : \mathfrak{s}$ and $\Gamma_4 = \Gamma_3, t :$

$\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot)$.

$$D_2 = \dfrac{\dfrac{\dfrac{D_3 \quad \dfrac{D_4 \quad D_8}{\Gamma_4 \vdash T_4 \mid T_8}\text{T-Par}_\text{B}}{\Gamma_3, t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot) \vdash T_3 \mid T_4 \mid T_8}\text{T-Par}_\text{B}}{\Gamma_3, t{:}\sharp(\mathfrak{v}_\top) \vdash (\nu f{:}\sharp(\mathfrak{v}_\bot))\,(T_3 \mid T_4 \mid T_8)}\text{T-Res}_\text{B}}{\Gamma_3 \vdash (\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot))\,(T_3 \mid T_4 \mid T_8)}\text{T-Res}_\text{B}$$

$\Gamma_4 \vdash \overline{l}\langle t, f\rangle$ for D_3 follows from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals. Let $\Gamma_5 = \Gamma_4, v_t{:}\mathfrak{v}_\top$. Then

$$D_4 = \dfrac{D'_4 \quad \dfrac{\dfrac{\dfrac{D_5 \quad \dfrac{D_6 \quad D_7}{\Gamma_5 \vdash T_6 \mid T_7}\text{T-Par}_\text{B}}{\Gamma_5 \vdash T_5 \mid T_6 \mid T_7}\text{T-Par}_\text{B}}{\Gamma_5 \vdash (\nu f{:}\sharp(\mathfrak{v}_\bot))\,(T_5 \mid T_6 \mid T_7)}\text{T-Res}_\text{B}}{\Gamma_5 \vdash (\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot))\,(T_5 \mid T_6 \mid T_7)}\text{T-Res}_\text{B}}{\Gamma_4 \vdash t(v_t)\,.\,(\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot))\,(T_5 \mid T_6 \mid T_7)}\text{T-In}_\text{B}$$

where $\Gamma_4 \vdash t{:}\sharp(\mathfrak{v}_\top)$ for D'_4 follows from T-Name$_\text{B}$. Again, $\Gamma_5 \vdash \overline{l'}\langle t, f\rangle$ for D_5 follows from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals.

$$D_6 = \dfrac{D'_6 \quad \dfrac{D_{6,1} \quad \dfrac{D_{6,2} \quad \dfrac{\dfrac{D_{6,3}}{\Gamma_5 \vdash (\nu v_s{:}\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle}\text{T-Res}_\text{B} \quad D_{6,4}}{\Gamma_5 \vdash (\nu v_s{:}\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\text{B}[\![\, S_i \,]\!]^\text{s}_\text{a}}P}{\Gamma_5 \vdash \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_s{:}\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\text{B}[\![\, S_i \,]\!]^\text{s}_\text{a}}P}{\Gamma_5 \vdash \mathcal{T}^1_\text{B}\big(\overline{l}\langle\bot\rangle\big) \mid \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_s{:}\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\text{B}[\![\, S_i \,]\!]^\text{s}_\text{a}}P}{\Gamma_5 \vdash t(v_t)\,.\,\big(\mathcal{T}^1_\text{B}\big(\overline{l}\langle\bot\rangle\big) \mid \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_s{:}\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\text{B}[\![\, S_i \,]\!]^\text{s}_\text{a}\big)}I$$

where $P = \text{T-Par}_\text{B}$ and $I = \text{T-In}_\text{B}$. $\Gamma_5 \vdash t{:}\sharp(\mathfrak{v}_\top)$ for D'_6 follows from T-Name$_\text{B}$. $\Gamma_5 \vdash \mathcal{T}^1_\text{B}\big(\overline{l}\langle\bot\rangle\big)$ for $D_{6,1}$ and $\Gamma_5 \vdash \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big)$ for $D_{6,2}$ follow from Lemma A.1.1 and Lemma 6.2.11. $\Gamma_5, v_s{:}\mathfrak{v}_\mathfrak{s} \vdash \overline{s}\langle v_s\rangle$ for $D_{6,3}$ follows from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals. $\Gamma_5 \vdash \mathcal{T}^1_\text{B}[\![\, S_i \,]\!]^\text{s}_\text{a}$ for $D_{6,4}$ follows from the induction hypothesis and Lemma 6.2.11. Let $\Gamma_6 = \Gamma_5, v_f{:}\mathfrak{v}_\bot$. Then

$$D_7 = \dfrac{D_{7,1} \quad \dfrac{D_{7,2} \quad \dfrac{D_{7,3} \quad \dfrac{D_{7,4}}{\Gamma_6 \vdash (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\langle v_{s,r}\rangle}\text{T-Res}_\text{B}}{\Gamma_6 \vdash \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}}P}{\Gamma_6 \vdash \mathcal{T}^1_\text{B}\big(\overline{l}\langle\top\rangle\big) \mid \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}}P}{\Gamma_5 \vdash f(v_f)\,.\,\big(\mathcal{T}^1_\text{B}\big(\overline{l}\langle\top\rangle\big) \mid \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big) \mid (\nu v_{s,r}{:}\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{r}\big)}\text{T-In}_\text{B}$$

where $P = \text{T-Par}_\text{B}$. $\Gamma_5 \vdash f{:}\sharp(\mathfrak{v}_\bot)$ follows from T-Name$_\text{B}$. Again, $\Gamma_6 \vdash \mathcal{T}^1_\text{B}\big(\overline{l}\langle\top\rangle\big)$ for $D_{7,2}$ and $\Gamma_6 \vdash \mathcal{T}^1_\text{B}\big(\overline{l'}\langle\bot\rangle\big)$ for $D_{7,3}$ follow from Lemma A.1.1

and Lemma 6.2.11. $\Gamma_6, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{r}\langle v_{s,r}\rangle$ for $D_{7,4}$ follows from T-Out$_\mathrm{B}$ and T-Name$_\mathrm{B}$ for all remaining subgoals. Let $\Gamma_7 = \Gamma_4, v_f : \mathfrak{v}_\perp$ Finally,

$$D_8 = \dfrac{D_{8,1} \quad \dfrac{\dfrac{}{\Gamma_7 \vdash \mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right)} R \quad D_{8,2}}{\Gamma_7 \vdash \mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right) \mid \overline{\varphi^\mathrm{s}_\mathrm{a}(y)}\langle l', s, \varphi^\mathrm{s}_\mathrm{a}(x)\rangle}\text{T-Par}_\mathrm{B}}{\Gamma_4 \vdash f(v_f).\mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right) \mid \overline{\varphi^\mathrm{s}_\mathrm{a}(y)}\langle l', s, \varphi^\mathrm{s}_\mathrm{a}(x)\rangle}\text{T-In}_\mathrm{B}$$

where R = Lemma A.1.1 and Lemma 6.2.11. $\Gamma_4 \vdash f : \sharp(\mathfrak{v}_\perp)$ for $D_{8,1}$ follows from T-Name$_\mathrm{B}$. $\Gamma_7 \vdash \overline{\varphi^\mathrm{s}_\mathrm{a}(y)}\langle l', s, \varphi^\mathrm{s}_\mathrm{a}(x)\rangle$ for $D_{8,2}$ follows from T-Out$_\mathrm{B}$ and T-Name$_\mathrm{B}$ for all remaining subgoals.

Case $S = y^\star(x).S'$**:** Then

$$\begin{aligned}
\mathcal{T}^1_\mathrm{B}[\![\, S \,]\!]^\mathrm{s}_\mathrm{a} &= \varphi^\mathrm{s}_\mathrm{a}(y)^\star(l, s, \varphi^\mathrm{s}_\mathrm{a}(x)).(\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\perp))\,(T_1 \mid T_2 \mid T_3)\\
T_1 &= \overline{l}\langle t, f\rangle\\
T_2 &= t(v_t).\left(\mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\mathrm{B}[\![\, S' \,]\!]^\mathrm{s}_\mathrm{a}\right)\\
T_3 &= f(v_f).\mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right)
\end{aligned}$$

Let $\Gamma = \widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, s{:}\mathfrak{s}, t{:}\sharp(\mathfrak{v}_\top)$

$$D_0 \quad \dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma, f{:}\sharp(\mathfrak{v}_\perp) \vdash T_2 \mid T_3}\text{T-Par}_\mathrm{B}}{\Gamma, f{:}\sharp(\mathfrak{v}_\perp) \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_\mathrm{B}}{\Gamma \vdash (\nu f{:}\sharp(\mathfrak{v}_\perp))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_\mathrm{B}}{\widetilde{\mathcal{T}_S}, l{:}\mathfrak{l}, s{:}\mathfrak{s} \vdash (\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\perp))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_\mathrm{B}}{\widetilde{\mathcal{T}_S} \vdash \varphi^\mathrm{s}_\mathrm{a}(y)^\star(l, s, \varphi^\mathrm{s}_\mathrm{a}(x)).(\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\perp))\,(T_1 \mid T_2 \mid T_3)}\text{T-Rep}_\mathrm{B}$$

By Definition 6.2.5 and because $x, y \in \mathsf{n}(S)$, there exists some source type $T_S \in \mathbb{T}_\mathrm{s}$ such that $y : \sharp(T_S), x : T_S \in \mathcal{T}_S$. Then, by Definition 6.2.6, we have $\varphi^\mathrm{s}_\mathrm{a}(y) : \sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T_S}\right), \varphi^\mathrm{s}_\mathrm{a}(x) : \widetilde{T_S} \in \widetilde{\mathcal{T}_S}$. Hence, $\widetilde{\mathcal{T}_S} \vdash \varphi^\mathrm{s}_\mathrm{a}(y) : \sharp\left(\mathfrak{l}, \mathfrak{s}, \widetilde{T_S}\right)$ for D_0 follows from T-Name$_\mathrm{B}$ and $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s}, \varphi^\mathrm{s}_\mathrm{a}(x) : \widetilde{T_S} = \widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s}$. $\widetilde{\mathcal{T}_S}, l : \mathfrak{l}, s : \mathfrak{s}, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\perp) \vdash \overline{l}\langle t, f\rangle$ for D_1 follows from T-Out$_\mathrm{B}$ and T-Name$_\mathrm{B}$ for all remaining subgoals. Let $\Gamma_1 = \Gamma, f{:}\sharp(\mathfrak{v}_\perp)$ and $\Gamma_2 = \Gamma_1, v_t : \mathfrak{v}_\top$. $D_2 =$

$$D_{2,1} \quad \dfrac{D_{2,2} \quad \dfrac{\dfrac{D_{2,3}}{\Gamma_2 \vdash (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle}\text{T-Res}_\mathrm{B} \quad D_{2,4}}{\Gamma_2 \vdash (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\mathrm{B}[\![\, S' \,]\!]^\mathrm{s}_\mathrm{a}}\text{T-Par}_\mathrm{B}}{\Gamma_2 \vdash \mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\mathrm{B}[\![\, S' \,]\!]^\mathrm{s}_\mathrm{a}}\text{T-Par}_\mathrm{B}}{\Gamma_1 \vdash t(v_t).\left(\mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^1_\mathrm{B}[\![\, S' \,]\!]^\mathrm{s}_\mathrm{a}\right)}\text{T-In}_\mathrm{B}$$

$\Gamma_1 \vdash t{:}\sharp(\mathfrak{v}_\top)$ for $D_{2,1}$ follows from T-Name$_\mathrm{B}$. $\Gamma_2 \vdash \mathcal{T}^1_\mathrm{B}\left(\overline{l}\langle\perp\rangle\right)$ for $D_{2,2}$ follows from Lemma A.1.1 and Lemma 6.2.11. $\Gamma_2, v_s : \mathfrak{v}_\mathfrak{s} \vdash \overline{s}\langle v_s\rangle$ for $D_{2,3}$ follows from

T-Out$_\mathrm{B}$ and T-Name$_\mathrm{B}$ for all remaining subgoals. $\Gamma_2 \vdash \mathcal{T}^1_\mathrm{B}[\![\, S' \,]\!]^\mathrm{s}_\mathrm{a}$ for $D_{2,4}$ follows from the induction hypothesis and Lemma 6.2.11.

$$D_3 = \dfrac{D'_3 \quad \dfrac{}{\Gamma_1, v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^1_\mathrm{B}\big(\overline{l}\langle\bot\rangle\big)}\text{Lemma A.1.1 and Lemma 6.2.11}}{\Gamma_1 \vdash f(v_f).\mathcal{T}^1_\mathrm{B}\big(\overline{l}\langle\bot\rangle\big)}\text{T-In}_\mathrm{B}$$

where $\Gamma_1 \vdash f : \sharp(\mathfrak{v}_\bot)$ for D'_3 follows from T-Name$_\mathrm{B}$.

□

Lemma 6.2.18 states that the encoding $\mathcal{T}^2_\mathrm{B}[\![\,\cdot\,]\!]^\mathrm{m}_\mathrm{p}$ is well-typed with respect to $\Gamma_{[\![\,\cdot\,]\!]^\mathrm{m}_\mathrm{p}}$, where $\Gamma_{[\![\,S\,]\!]^\mathrm{m}_\mathrm{p}} = \{\, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \,\} \cup \{\, \varphi^\mathrm{m}_\mathrm{a}(x) : \mathfrak{v}_\mathfrak{n} \mid x \in \mathsf{fn}(S) \,\}$ for all source terms $S \in \mathcal{P}_\mathrm{m}$.

Proof of Lemma 6.2.18. Again, we perform an induction over the structure of source terms. Note that no source term can contain infinitely many free names. We conclude that $\Gamma_{[\![\,\cdot\,]\!]^\mathrm{m}_\mathrm{p}}$ is finite for each source term $S \in \mathcal{P}_\mathrm{m}$. Let $\Gamma = \Gamma_{[\![\,S\,]\!]^\mathrm{m}_\mathrm{p}}$.

Base Case: In $\mathcal{P}_\mathrm{m}$ there are two terms without subterms, namely $\mathsf{0}$ and $\checkmark$. $\mathcal{T}^2_\mathrm{B}[\![\,\mathsf{0}\,]\!]^\mathrm{m}_\mathrm{p} = (\nu l : \mathfrak{l})\, l(t, f).\big((\nu v_t : \mathfrak{v}_\top)\, \overline{t}\langle v_t\rangle\big)$ and $\mathcal{T}^2_\mathrm{B}[\![\,\checkmark\,]\!]^\mathrm{m}_\mathrm{p} = \checkmark$. The second case, i.e., the type judgement $\Gamma \vdash \checkmark$, follows directly from T-Succ$_\mathrm{B}$. For the first case, we have

$$\dfrac{\dfrac{}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}^2_\mathrm{B}\big(\overline{l}\langle\top\rangle\big)}\text{Lemma A.1.1 and Lemma 6.2.11}}{\Gamma \vdash (\nu l : \mathfrak{l})\,\big(\mathcal{T}^2_\mathrm{B}\big(\overline{l}\langle\top\rangle\big)\big)}\text{T-Res}_\mathrm{B}$$

Induction Hypothesis:

$$\forall S \in \mathcal{P}_\mathrm{m}\,.\,\Gamma_{[\![\,S\,]\!]^\mathrm{m}_\mathrm{p}} \vdash \mathcal{T}^2_\mathrm{B}[\![\,S\,]\!]^\mathrm{m}_\mathrm{p} \tag{IH}$$

Induction Step: We perform a case split over the structure of S.

Case $S = (\nu x)\, S'$**:** Then $\mathcal{T}^2_\mathrm{B}[\![\,S\,]\!]^\mathrm{m}_\mathrm{p} = \big(\nu\varphi^\mathrm{m}_\mathrm{p}(x) : \mathfrak{v}_\mathfrak{n}\big)\, \mathcal{T}^2_\mathrm{B}[\![\,S'\,]\!]^\mathrm{m}_\mathrm{p}$ and

$$\dfrac{\dfrac{}{\Gamma, \varphi^\mathrm{m}_\mathrm{p}(x) : \mathfrak{v}_\mathfrak{n} \vdash \mathcal{T}^2_\mathrm{B}[\![\,S'\,]\!]^\mathrm{m}_\mathrm{p}}(IH)\text{and Lemma 6.2.11}}{\Gamma \vdash \big(\nu\varphi^\mathrm{m}_\mathrm{p}(x) : \mathfrak{v}_\mathfrak{n}\big)\, \mathcal{T}^2_\mathrm{B}[\![\,S'\,]\!]^\mathrm{m}_\mathrm{p}}\text{T-Res}_\mathrm{B}$$

Case $S = S_1 \mid S_2$**:** Then

$$\begin{aligned}
\mathcal{T}^2_\mathrm{B}[\![\,S\,]\!]^\mathrm{m}_\mathrm{p} &= \big(\nu p_{o,up} : \mathfrak{o}', p_{i,up} : \mathfrak{i}', o : \mathfrak{t}_\mathfrak{o}, i : \mathfrak{t}_\mathfrak{i}\big)\,(T_1 \mid T_4 \mid T_7)\\
T_1 &= \big(\nu p_o : \mathfrak{o}', p_i : \mathfrak{i}'\big)\,\Big(\mathcal{T}^2_\mathrm{B}[\![\,S_1\,]\!]^\mathrm{m}_\mathrm{p} \mid T_2 \mid T_3\Big)\\
T_2 &= \mathsf{procLeftOutReq}\\
T_3 &= \mathsf{procLeftInReq}\\
T_4 &= \big(\nu p_o : \mathfrak{o}', p_i : \mathfrak{i}'\big)\,\Big(\mathcal{T}^2_\mathrm{B}[\![\,S_2\,]\!]^\mathrm{m}_\mathrm{p} \mid T_5 \mid T_6\Big)\\
T_5 &= \mathsf{procRightInReq}\\
T_6 &= \mathsf{procRightOutReq}\\
T_7 &= \mathsf{pushReq}
\end{aligned}$$

We have to show that $\Gamma \vdash \mathcal{T}^2_{\mathrm{B}}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{p}}$. After applying four times Rule T-Res$_{\mathrm{B}}$ it remains to show that $\Gamma_1 \vdash T_1 \mid T_4 \mid T_7$, where $\Gamma_1 = \Gamma, p_{o,up} : \mathfrak{o}', p_{i,up} : \mathfrak{i}', o : \mathfrak{t}_{\mathfrak{o}}, i : \mathfrak{t}_{\mathfrak{i}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash T_2 \mid T_3}P}{\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash \mathcal{T}^2_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_2 \mid T_3}P}{\Gamma_1, p_o : \mathfrak{o}' \vdash (\nu p_i : \mathfrak{i}') \left(\mathcal{T}^2_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_2 \mid T_3\right)}R}{\Gamma_1 \vdash (\nu p_o : \mathfrak{o}', p_i : \mathfrak{i}') \left(\mathcal{T}^2_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_2 \mid T_3\right)}R \quad \dfrac{D_4 \quad D_7}{\Gamma_1 \vdash T_4 \mid T_7}}{\Gamma_1 \vdash T_1 \mid T_4 \mid T_7}P$$

where $P =$ T-Par$_{\mathrm{B}}$ and $R =$ T-Res$_{\mathrm{B}}$. $\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash \mathcal{T}^2_{\mathrm{B}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{p}}$ for D_1 follows from the induction hypothesis and by Lemma 6.2.11. Let $\Gamma_2 = \Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}'$ and $\Gamma_3 = \Gamma_2, y, z : \mathfrak{v}_{\mathfrak{n}}, l : \mathfrak{l}, s_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), s_2 : \mathfrak{s}$.

$$D_2 = \dfrac{D_{2,1} \quad \dfrac{D_{2,2} \quad D_{2,3}}{\Gamma_3 \vdash \overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle}\text{T-Par}_{\mathrm{B}}}{\Gamma_2 \vdash p_o{}^{\star}(y, l, s_1, s_2, z) \,.\, (\overline{y \cdot o}\langle l, s_1, s_2, z\rangle \mid \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle)}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_2 \vdash p_o : \mathfrak{o}'$ for $D_{2,1}$ follows from T-Name$_{\mathrm{B}}$. $\Gamma_3 \vdash \overline{y \cdot o}\langle l, s_1, s_2, z\rangle$ for $D_{2,2}$ follows from T-OutPS$_{\mathrm{B}}$ and T-Name$_{\mathrm{B}}$ and $\Gamma_3 \vdash \overline{p_{o,up}}\langle y, l, s_1, s_2, z\rangle$ for $D_{2,3}$ follows from T-Out$_{\mathrm{B}}$ and T-Name$_{\mathrm{B}}$ for all remaining subgoals. Let $\Gamma_4 = \Gamma_2, y : \mathfrak{v}_{\mathfrak{n}}, l : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), r_2 : \mathfrak{r}'$.

$$D_3 = \dfrac{D_{3,1} \quad \dfrac{D_{3,2} \quad D_{3,3}}{\Gamma_4 \vdash \overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle}\text{T-Par}_{\mathrm{B}}}{\Gamma_2 \vdash p_i{}^{\star}(y, l, r_1, r_2) \,.\, (\overline{y \cdot i}\langle l, r_1, r_2\rangle \mid \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle)}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_2 \vdash p_i : \mathfrak{i}'$ for $D_{3,1}$ follows from T-Name$_{\mathrm{B}}$. $\Gamma_4 \vdash \overline{y \cdot i}\langle l, r_1, r_2\rangle$ for $D_{3,2}$ follows from T-OutPS$_{\mathrm{B}}$ and T-Name$_{\mathrm{B}}$ and $\Gamma_4 \vdash \overline{p_{i,up}}\langle y, l, r_1, r_2\rangle$ for $D_{3,3}$ follows from T-Out$_{\mathrm{B}}$ and T-Name$_{\mathrm{B}}$ for all remaining subgoals.

$$D_4 = \dfrac{\dfrac{\dfrac{D'_4 \quad \dfrac{D_5 \quad D_6}{\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash T_5 \mid T_6}\text{T-Par}_{\mathrm{B}}}{\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash \mathcal{T}^2_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_5 \mid T_6}\text{T-Par}_{\mathrm{B}}}{\Gamma_1, p_o : \mathfrak{o}' \vdash (\nu p_i : \mathfrak{i}') \left(\mathcal{T}^2_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_5 \mid T_6\right)}\text{T-Res}_{\mathrm{B}}}{\Gamma_1 \vdash (\nu p_o : \mathfrak{o}', p_i : \mathfrak{i}') \left(\mathcal{T}^2_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{p}} \mid T_5 \mid T_6\right)}\text{T-Res}_{\mathrm{B}}$$

where $\Gamma_1, p_o : \mathfrak{o}', p_i : \mathfrak{i}' \vdash \mathcal{T}^2_{\mathrm{B}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{p}}$ for D'_4 follows from the induction hypothesis and by Lemma 6.2.11. Let $T'_5 = \overline{r_2}\langle l_r, l_s, l_s, s_2, z, r_1, s_1\rangle$, $T''_5 = \overline{p_{o,up}}\langle y, l_s, s_1, s_2, z\rangle$, $\Gamma_5 = \Gamma_2, y, z : \mathfrak{v}_{\mathfrak{n}}, l_s : \mathfrak{l}, s_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}), s_2 : \mathfrak{s}$, and $\Gamma_6 = \Gamma_5, l_r :$

$\mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) , r_2 : \mathfrak{r}'$.

$$D_5 = \dfrac{D_{5,1} \quad \dfrac{\dfrac{D_{5,2} \quad D_{5,3} \quad D_{5,4}}{\Gamma_5 \vdash y \cdot i(l_r, r_1, r_2) \,.T_5'}\text{T-InPS}_\text{B} \quad D_{5,5}}{\Gamma_5 \vdash y \cdot i(l_r, r_1, r_2) \,.T_5' \mid T_5''}\text{T-Par}_\text{B}}{\Gamma_2 \vdash p_o\star(y, l_s, s_1, s_2, z) \,. (y \cdot i(l_r, r_1, r_2) \,.T_5' \mid T_5'')}\text{T-Rep}_\text{B}$$

$\Gamma_2 \vdash p_o : \mathfrak{o}'$ for $D_{5,1}$, $\Gamma_5 \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{5,2}$, and $\Gamma_5 \vdash i : \mathfrak{t}_\mathfrak{i}$ for $D_{5,3}$ follow from T-Name$_\text{B}$. $\Gamma_6 \vdash T_5'$ for $D_{5,4}$ and $\Gamma_5 \vdash T_5''$ for $D_{5,5}$ follow from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals. Let $T_6' = \overline{r_2}\langle l_s, l_r, l_s, s_2, z, s_1, r_1\rangle$, $T_6'' = \overline{p_{i,up}}\langle y, l_r, r_1, r_2\rangle$, $\Gamma_7 = \Gamma_2, y : \mathfrak{v}_\mathfrak{n}, l_r : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) , r_2 : \mathfrak{r}'$, and $\Gamma_8 = \Gamma_7, l_s : \mathfrak{l}, s_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) , s_2 : \mathfrak{s}, z : \mathfrak{v}_\mathfrak{n}$.

$$D_6 = \dfrac{D_{6,1} \quad \dfrac{\dfrac{D_{6,2} \quad D_{6,3} \quad D_{6,4}}{\Gamma_7 \vdash y \cdot o(l_s, s_1, s_2, z) \,.T_6'}\text{T-InPS}_\text{B} \quad D_{6,5}}{\Gamma_7 \vdash y \cdot o(l_s, s_1, s_2, z) \,.T_6' \mid T_6''}\text{T-Par}_\text{B}}{\Gamma_2 \vdash p_i\star(y, l_r, r_1, r_2) \,. (y \cdot o(l_s, s_1, s_2, z) \,.T_6' \mid T_6'')}\text{T-Rep}_\text{B}$$

$\Gamma_2 \vdash p_i : \mathfrak{i}'$ for $D_{6,1}$, $\Gamma_7 \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{6,2}$, and $\Gamma_7 \vdash o : \mathfrak{t}_\mathfrak{o}$ for $D_{6,3}$ follow from T-Name$_\text{B}$. $\Gamma_8 \vdash T_6'$ for $D_{6,4}$ and $\Gamma_8 \vdash T_6''$ for $D_{6,5}$ follow from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals. Let $T_7' = \overline{p_o}\langle y, l, s_1, s_2, z\rangle$ and $T_7'' = \overline{p_i}\langle y, l, r_1, r_2\rangle$.

$$D_7 = \dfrac{\dfrac{D_{7,1} \quad D_{7,2}}{\Gamma_1 \vdash p_{o,up}\star(y, l, s_1, s_2, z) \,.T_7'}R \quad \dfrac{D_{7,3} \quad D_{7,4}}{\Gamma_1 \vdash p_{i,up}\star(y, l, r_1, r_2) \,.T_7''}R}{\Gamma_1 \vdash p_{o,up}\star(y, l, s_1, s_2, z) \,.T_7' \mid p_{i,up}\star(y, l, r_1, r_2) \,.T_7''}\text{T-Par}_\text{B}$$

where $R = \text{T-Rep}_\text{B}$. $\Gamma_1 \vdash p_{o,up} : \mathfrak{o}'$ for $D_{7,1}$ and $\Gamma_1 \vdash p_{i,up} : \mathfrak{i}'$ for $D_{7,3}$ follow from T-Name$_\text{B}$. Finally, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) , s_2 : \mathfrak{s} \vdash T_7'$ for $D_{7,2}$ and $\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) , r_2 : \mathfrak{r}' \vdash T_7''$ for $D_{7,4}$ follow from T-Out$_\text{B}$ and T-Name$_\text{B}$ for all remaining subgoals.

Case $S = \sum_{i\in I} \pi_i.S_i$**:** Then $\mathcal{T}^2_\text{B}[\![\, S \,]\!]^\text{m}_\text{p} = (\nu l : \mathfrak{l}) \left(\mathcal{T}^2_\text{B}\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i\in I} \mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p}\right)$.

$$\dfrac{\dfrac{}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}^2_\text{B}\big(\overline{l}\langle\top\rangle\big)}\text{Lemma A.1.1 and Lemma 6.2.11} \quad D}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}^2_\text{B}\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i\in I} \mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p}}\text{T-Par}_\text{B}}{\Gamma \vdash (\nu l : \mathfrak{l}) \left(\mathcal{T}^2_\text{B}\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i\in I} \mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p}\right)}\text{T-Res}_\text{B}$$

To prove D, we have to show that $\Gamma, l : \mathfrak{l} \vdash \prod_{i\in I} \mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p}$. With T-Par$_\text{B}$ we decompose this goal into several subgoals of the form $\Gamma, l : \mathfrak{l} \vdash \mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p}$, where each π_i is either a τ, an output or an input prefix.

Case $\pi_i = \tau$**:** Then $\mathcal{T}^2_\text{B}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{p} = (\nu t : \sharp(\mathfrak{v}_\top) , f : \sharp(\mathfrak{v}_\bot)) \,(T_1 \mid T_2 \mid T_3)$, where the subterms $T_1 = \overline{l}\langle t, f\rangle$, $T_2 = t(v_t) \,. \left(\mathcal{T}^2_\text{B}\big(\overline{l}\langle\bot\rangle\big) \mid \mathcal{T}^2_\text{B}[\![\, S_i \,]\!]^\text{m}_\text{p}\right)$ and $T_3 =$

$f(v_f)\,.\,\big(\mathcal{T}^2_{\mathrm{B}}\big(\overline{l}\langle\bot\rangle\big)\big)$. Let $\Gamma' = \Gamma, l\!:\!\mathfrak{l}, t\!:\!\sharp(\mathfrak{v}_\top)\,, f\!:\!\sharp(\mathfrak{v}_\bot)$. Then

$$\dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma' \vdash T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l}, t\!:\!\sharp(\mathfrak{v}_\top)\,, f\!:\!\sharp(\mathfrak{v}_\bot) \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l}, t\!:\!\sharp(\mathfrak{v}_\top) \vdash (\nu f\!:\!\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l} \vdash (\nu t\!:\!\sharp(\mathfrak{v}_\top)\,, f\!:\!\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}$$

Remember that $\mathfrak{l} = \sharp(\sharp(\mathfrak{v}_\top)\,, \sharp(\mathfrak{v}_\bot))$. Hence,

$$D_1 = \dfrac{\dfrac{}{\Gamma' \vdash l\!:\!\mathfrak{l}}N \quad \dfrac{}{\Gamma' \vdash t\!:\!\sharp(\mathfrak{v}_\top)}N \quad \dfrac{}{\Gamma' \vdash f\!:\!\sharp(\mathfrak{v}_\bot)}N}{\Gamma' \vdash \overline{l}\langle t, f\rangle}\text{T-Out}_{\mathrm{B}}$$

where $N = \text{T-Name}_{\mathrm{B}}$.

$$D_2 = \dfrac{\dfrac{}{\Gamma' \vdash t\!:\!\sharp(\mathfrak{v}_\top)}\text{T-Name}_{\mathrm{B}} \quad D_2'}{\Gamma' \vdash t(v_t)\,.\,\Big(\mathcal{T}^2_{\mathrm{B}}\big(\overline{l}\langle\bot\rangle\big) \mid \mathcal{T}^2_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{p}}\Big)}\text{T-In}_{\mathrm{B}}$$

To prove $\Gamma', v_t\!:\!\mathfrak{v}_\top \vdash \mathcal{T}^2_{\mathrm{B}}\big(\overline{l}\langle\bot\rangle\big) \mid \mathcal{T}^2_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{p}}$ for D_2' apply T-Par$_{\mathrm{B}}$ and then Lemma A.1.1 and Lemma 6.2.11 at the left hand side and the induction hypothesis and Lemma 6.2.11 at the right hand side.

$$D_3 = \dfrac{\dfrac{}{\Gamma' \vdash f\!:\!\sharp(\mathfrak{v}_\bot)}\text{T-Name}_{\mathrm{B}} \quad \dfrac{}{\Gamma', v_f\!:\!\mathfrak{v}_\bot \vdash \mathcal{T}^2_{\mathrm{B}}\big(\overline{l}\langle\bot\rangle\big)}R}{\Gamma' \vdash f(v_f)\,.\,\big(\mathcal{T}^2_{\mathrm{B}}\big(\overline{l}\langle\bot\rangle\big)\big)}\text{T-In}_{\mathrm{B}}$$

where $R =$ Lemma A.1.1 and Lemma 6.2.11.

Case $\pi_i = \overline{y}\langle z\rangle$**:** Then

$$\begin{aligned}
\mathcal{T}^2_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{p}} &= (\nu s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, s_2\!:\!\mathfrak{s})\,(T_1 \mid T_2 \mid T_3)\\
T_1 &= (\nu v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{s_1}\langle v_{s,r}\rangle\\
T_2 &= s_1^\star(v_{s,r})\,.\overline{p_o}\big\langle \varphi^{\mathrm{m}}_{\mathrm{p}}(y)\,, l, s_1, s_2, \varphi^{\mathrm{m}}_{\mathrm{p}}(z)\big\rangle\\
T_3 &= s_2(v_s)\,.\mathcal{T}^2_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{p}}
\end{aligned}$$

Let $\Gamma_1 = \Gamma, l\!:\!\mathfrak{l}, s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, s_2\!:\!\mathfrak{s}$.

$$\dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma_1 \vdash T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l}, s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, s_2\!:\!\mathfrak{s} \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l}, s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash (\nu s_2\!:\!\mathfrak{s})\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}}{\Gamma, l\!:\!\mathfrak{l} \vdash (\nu s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,, s_2\!:\!\mathfrak{s})\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}$$

The proof of the first subgoal is given by the derivation

$$D_1 = \dfrac{\dfrac{}{\Gamma_1, v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash s_1\!:\!\sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})}N \quad \dfrac{}{\Gamma_1, v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}}N}{\Gamma_1, v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{s_1}\langle v_{s,r}\rangle}O}{\Gamma_1 \vdash (\nu v_{s,r}\!:\!\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\,\overline{s_1}\langle v_{s,r}\rangle}R$$

where $N = \text{T-NAME}_\text{B}$, $O = \text{T-OUT}_\text{B}$, and $R = \text{T-RES}_\text{B}$.

$$D_2 = \frac{D_{2,1} \quad D_{2,2}}{\Gamma_1 \vdash s_1^\star(v_{s,r}) \,.\, \overline{p_o}\langle \varphi_\text{p}^\text{m}(y) \,,\, l, s_1, s_2, \varphi_\text{p}^\text{m}(z) \rangle}\,\text{T-REP}_\text{B}$$

where $\Gamma_1 \vdash s_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$ for $D_{2,1}$ follows from T-NAME_B. Since $y, z \in \mathsf{fn}(S)$, we have $y, z : \mathfrak{v}_\mathfrak{n} \in \Gamma$. Hence, $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{p_o}\langle \varphi_\text{p}^\text{m}(y) \,,\, l, s_1, s_2, \varphi_\text{p}^\text{m}(z) \rangle$ for $D_{2,2}$ follows from T-OUT_B and T-NAME_B for all remaining subgoals.

$$D_3 = \frac{\dfrac{}{\Gamma_1 \vdash s_2 : \mathfrak{s}}\,\text{T-NAME}_\text{B} \quad \dfrac{}{\Gamma_1, v_s : \mathfrak{v}_\mathfrak{s} \vdash \mathcal{T}_\text{B}^2[\![\, S_i \,]\!]_\text{p}^\text{m}}\,R}{\Gamma_1 \vdash s_2(v_s) \,.\, \mathcal{T}_\text{B}^2[\![\, S_i \,]\!]_\text{p}^\text{m}}\,\text{T-IN}_\text{B}$$

where $R = (IH)$ and Lemma 6.2.11.

Case $\pi_i = y(x)$**:** Then:

$$\begin{aligned}
\mathcal{T}_\text{B}^2[\![\, \pi_i.S_i \,]\!]_\text{p}^\text{m} &= \left(\nu r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \,,\, r_2 : \mathfrak{r}'\right)(T_1 \mid T_2 \mid T_3) \\
T_1 &= (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{r_1}\langle v_{s,r} \rangle \\
T_2 &= r_1^\star(v_{s,r}) \,.\, \overline{p_i}\langle \varphi_\text{p}^\text{m}(y) \,,\, l, r_1, r_2 \rangle \\
T_3 &= r_2^\star\left(l_1, l_2, l_s, s_2, \varphi_\text{p}^\text{m}(x) \,,\, v, w\right) . \\
&\qquad ((\nu t : \sharp(\mathfrak{v}_\top) \,,\, f : \sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_9)) \\
T_4 &= \overline{l_1}\langle t, f \rangle \\
T_5 &= t(v_t) \,.\, ((\nu t : \sharp(\mathfrak{v}_\top) \,,\, f : \sharp(\mathfrak{v}_\bot))\,(T_6 \mid T_7 \mid T_8)) \\
T_6 &= \overline{l_2}\langle t, f \rangle \\
T_7 &= t(v_t) \,.\, \left(\mathcal{T}_\text{B}^2\left(\overline{l_1}\langle \bot \rangle\right) \mid \mathcal{T}_\text{B}^2\left(\overline{l_2}\langle \bot \rangle\right)\right. \\
&\qquad \left. \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\, \overline{s}\langle v_s \rangle \mid \mathcal{T}_\text{B}^2[\![\, S_i \,]\!]_\text{p}^\text{m}\right) \\
T_8 &= f(v_f) \,.\, \left(\mathcal{T}_\text{B}^2\left(\overline{l_1}\langle \bot \rangle\right) \mid \mathcal{T}_\text{B}^2\left(\overline{l_2}\langle \top \rangle\right) \mid (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{v}\langle v_{s,r} \rangle\right) \\
T_9 &= f(v_f) \,.\, \left(\mathcal{T}_\text{B}^2\left(\overline{l_1}\langle \bot \rangle\right) \mid (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{w}\langle v_{s,r} \rangle\right)
\end{aligned}$$

Let $\Gamma_1 = \Gamma, l : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \,,\, r_2 : \mathfrak{r}'$.

$$\frac{\dfrac{\dfrac{\dfrac{D_1}{\Gamma_1 \vdash T_1}\,\text{T-RES}_\text{B} \quad \dfrac{D_2 \quad D_3}{\Gamma_1 \vdash T_2 \mid T_3}\,\text{T-PAR}_\text{B}}{\Gamma, l : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \,,\, r_2 : \mathfrak{r}' \vdash T_1 \mid T_2 \mid T_3}\,\text{T-PAR}_\text{B}}{\Gamma, l : \mathfrak{l}, r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}}) \vdash (\nu r_2 : \mathfrak{r}')\,(T_1 \mid T_2 \mid T_3)}\,\text{T-RES}_\text{B}}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}}\,\text{T-RES}_\text{B}$$

where the subgoal $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{r_1}\langle v_{s,r} \rangle$ for D_1 follows from T-OUT_B and then T-NAME_B on all subgoals.

$$D_2 = \frac{D_{2,1} \quad \dfrac{D_{2,2} \quad D_{2,3} \quad D_{2,4} \quad D_{2,5} \quad D_{2,6}}{\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{p_i}\langle \varphi_\text{p}^\text{m}(y) \,,\, l, r_1, r_2 \rangle}\,\text{T-OUT}_\text{B}}{\Gamma_1 \vdash r_1^\star(v_{s,r}) \,.\, \overline{p_i}\langle \varphi_\text{p}^\text{m}(y) \,,\, l, r_1, r_2 \rangle}\,\text{T-REP}_\text{B}$$

where $\Gamma_1 \vdash r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$ for $D_{2,1}$, $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash p_i : \mathfrak{i}'$ for $D_{2,2}$, $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash l : \mathfrak{l}$ for $D_{2,4}$, $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash r_1 : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$ for $D_{2,5}$ and $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash r_2 : \mathfrak{r}'$ for $D_{2,6}$ follow from T-Name$_\mathrm{B}$. Moreover, since $y \in \mathsf{fn}(S)$, we have $y : \mathfrak{v}_\mathfrak{n} \in \Gamma$. Hence, $\Gamma_1, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \varphi^\mathrm{m}_\mathrm{p}(y) : \mathfrak{v}_\mathfrak{n}$ for $D_{2,3}$ follows from T-Name$_\mathrm{B}$. Let $\Gamma_2 = \Gamma_1, l_1, l_2, l_s : \mathfrak{l}, s_2 : \mathfrak{s}, \varphi^\mathrm{m}_\mathrm{p}(x) : \mathfrak{v}_\mathfrak{n}, v, w : \sharp(\mathfrak{v}_{\mathfrak{s},\mathfrak{r}})$.

$$D_3 = \dfrac{D'_3 \quad \dfrac{\dfrac{D_4 \quad \dfrac{D_5 \quad D_9}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash T_5 \mid T_9} P}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash T_4 \mid T_5 \mid T_9} P}{\Gamma_2, t : \sharp(\mathfrak{v}_\top) \vdash (\nu f : \sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_9)} R}{\Gamma_2 \vdash (\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_9)} R}{\Gamma_1 \vdash T_3}\,\text{T-Rep}_\mathrm{B}$$

where $R = \text{T-Res}_\mathrm{B}$ and $P = \text{T-Par}_\mathrm{B}$. $\Gamma_1 \vdash r_2 : \mathfrak{r}'$ follows from T-Name$_\mathrm{B}$. $\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash \overline{l_1}\langle t, f\rangle$ for D_4 follows from T-Out$_\mathrm{B}$ and T-Name$_\mathrm{B}$ for all remaining subgoals. Let $\Gamma_3 = \Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot)$ and $\Gamma_4 = \Gamma_3, v_t : \mathfrak{v}_\top$.

$$D_5 = \dfrac{D'_5 \quad \dfrac{\dfrac{D_6 \quad \dfrac{D_7 \quad D_8}{\Gamma_4 \vdash T_7 \mid T_8}\,\text{T-Par}_\mathrm{B}}{\Gamma_4 \vdash T_6 \mid T_7 \mid T_8}\,\text{T-Par}_\mathrm{B}}{\Gamma_4 \vdash (\nu f : \sharp(\mathfrak{v}_\bot))\,(T_6 \mid T_7 \mid T_8)} R}{\Gamma_3, v_t : \mathfrak{v}_\top \vdash (\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,(T_6 \mid T_7 \mid T_8)} R}{\Gamma_3 \vdash t(v_t)\,.\,((\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,(T_6 \mid T_7 \mid T_8))}\,\text{T-In}_\mathrm{B}$$

where $R = \text{T-Res}_\mathrm{B}$. $\Gamma_3 \vdash t : \sharp(\mathfrak{v}_\top)$ for D'_5 follows from T-Name$_\mathrm{B}$. Moreover, apply T-Out$_\mathrm{B}$ and then T-Name$_\mathrm{B}$ on all subgoals to show $\Gamma_4 \vdash \overline{l_2}\langle t, f\rangle$ for D_6. Let $T'_7 = (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle$.

$$D_7 = \dfrac{D_{7,1} \quad \dfrac{D_{7,2} \quad \dfrac{D_{7,3} \quad \dfrac{\dfrac{D_{7,4}}{\Gamma_4 \vdash T'_7} R \quad D_{7,5}}{\Gamma_4 \vdash T'_7 \mid \mathcal{T}^2_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{p}} P}{\Gamma_4 \vdash \mathcal{T}^2_\mathrm{B}\big(\overline{l_2}\langle\bot\rangle\big) \mid T'_7 \mid \mathcal{T}^2_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{p}} P}{\Gamma_4 \vdash \mathcal{T}^2_\mathrm{B}\big(\overline{l_1}\langle\bot\rangle\big) \mid \mathcal{T}^2_\mathrm{B}\big(\overline{l_2}\langle\bot\rangle\big) \mid T'_7 \mid \mathcal{T}^2_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{p}} P}{\Gamma_4 \vdash T_7}\,\text{T-In}_\mathrm{B}$$

where $P = \text{T-Par}_\mathrm{B}$ and $R = \text{T-Res}_\mathrm{B}$. Again, $\Gamma_4 \vdash t : \sharp(\mathfrak{v}_\top)$ for $D_{7,1}$ follows from T-Name$_\mathrm{B}$. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_4 \vdash \mathcal{T}^2_\mathrm{B}\big(\overline{l_1}\langle\bot\rangle\big)$ for $D_{7,2}$ and $\Gamma_4 \vdash \mathcal{T}^2_\mathrm{B}\big(\overline{l_2}\langle\bot\rangle\big)$ for $D_{7,3}$. To show $\Gamma_4, v_s : \mathfrak{v}_\mathfrak{s} \vdash \overline{s}\langle v_s\rangle$ for $D_{7,4}$ apply T-Out$_\mathrm{B}$ and then T-Name$_\mathrm{B}$ for both subgoals. $\Gamma_4 \vdash \mathcal{T}^2_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{p}$ for $D_{7,5}$ follows from the induction hypothesis

and Lemma 6.2.11. Let $T'_8 = (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{v}\langle v_{s,r}\rangle$.

$$D_8 = \dfrac{D_{8,1} \quad \dfrac{D_{8,2} \quad \dfrac{D_{8,3} \quad \dfrac{D_{8,4}}{\Gamma_4, v_f : \mathfrak{v}_\perp \vdash T'_8}\text{T-Res}_\text{B}}{\Gamma_4, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_2}\langle\top\rangle\big) \mid T'_8} P}{\Gamma_4, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big) \mid \mathcal{T}^2_\text{B}\big(\overline{l_2}\langle\top\rangle\big) \mid T'_8} P}{\Gamma_4 \vdash f(v_f)\,.\,\big(\mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big) \mid \mathcal{T}^2_\text{B}\big(\overline{l_2}\langle\top\rangle\big) \mid T'_8\big)}\text{T-In}_\text{B}$$

where $P = \text{T-Par}_\text{B}$. $\Gamma_4 \vdash f : \sharp(\mathfrak{v}_\perp)$ for $D_{8,1}$ follows from T-Name_B. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_4, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big)$ for $D_{8,2}$ and $\Gamma_4, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_2}\langle\top\rangle\big)$ for $D_{8,3}$. To show $\Gamma_4, v_f : \mathfrak{v}_\perp, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{v}\langle v_{s,r}\rangle$ for $D_{8,4}$ apply T-Out_B and then T-Name_B for both subgoals.

$$D_9 = \dfrac{D_{9,1} \quad \dfrac{D_{9,2} \quad \dfrac{D_{9,3}}{\Gamma_3, v_f : \mathfrak{v}_\perp \vdash (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{w}\langle v_{s,r}\rangle}\text{T-Res}_\text{B}}{\Gamma_3, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big) \mid (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{w}\langle v_{s,r}\rangle}\text{T-Par}_\text{B}}{\Gamma_3 \vdash f(v_f)\,.\,\big(\mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big) \mid (\nu v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}})\, \overline{w}\langle v_{s,r}\rangle\big)} I$$

where $I = \text{T-In}_\text{B}$. $\Gamma_3 \vdash f : \sharp(\mathfrak{v}_\perp)$ for $D_{9,1}$ follows from T-Name_B. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_3, v_f : \mathfrak{v}_\perp \vdash \mathcal{T}^2_\text{B}\big(\overline{l_1}\langle\perp\rangle\big)$ for $D_{9,2}$. Finally, apply T-Out_B and then T-Name_B for both subgoals to show $\Gamma_3, v_f : \mathfrak{v}_\perp, v_{s,r} : \mathfrak{v}_{\mathfrak{s},\mathfrak{r}} \vdash \overline{w}\langle v_{s,r}\rangle$ for $D_{9,3}$.

□

In the following we present the missing cases of Lemma 6.2.17, i.e., show that also parallel composition, sum, and replicated inputs are well-typed in the basic type system. Remember that $\Gamma = \Gamma_{[\![\,\cdot\,]\!]^\text{m}_\text{a}}$, where $\Gamma_{[\![\,S\,]\!]^\text{m}_\text{a}} = \{\, p_o : \mathfrak{o}, p_i : \mathfrak{i} \,\} \cup \{\, \varphi^\text{m}_\text{a}(x) : \mathfrak{v}_\mathfrak{n} \mid x \in \mathsf{fn}(S) \,\}$ for all source terms $S \in \mathcal{P}_\text{m}$, and the induction hypothesis is given by:

$$\forall S \in \mathcal{P}_\text{m}\,.\,\Gamma_{[\![\,S\,]\!]^\text{m}_\text{a}} \vdash \mathcal{T}^3_\text{B}[\![\,S\,]\!]^\text{m}_\text{a} \tag{IH}$$

1. If $S = S_1 \mid S_2$ for some $S_1, S_2 \in \mathcal{P}_\text{m}$ then:

$$\begin{aligned}\mathcal{T}^3_\text{B}[\![\,S\,]\!]^\text{m}_\text{a} = (\nu m_o, p_{o,up} : \mathfrak{o}, m_i, p_{i,up} : \mathsf{i}, c_o : \sharp(\mathsf{i})\,, c_i : \sharp(\mathfrak{o}))\,\big(&\\ (\nu p_o : \mathfrak{o}, p_i : \mathsf{i})\,\big(\mathcal{T}^3_\text{B}[\![\,S_1\,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftInReq})\big)&\\ \mid (\nu p_o : \mathfrak{o}, p_i : \mathsf{i})\,\big(\mathcal{T}^3_\text{B}[\![\,S_2\,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procRightInReq})\big)&\\ \mid \mathcal{T}^3_\text{B}(\mathsf{pushReq})\,\big)&\end{aligned}$$

By applying T-Res_B several times, $\Gamma \vdash \mathcal{T}^3_\text{B}[\![\,S\,]\!]^\text{m}_\text{a}$ becomes $\Gamma_1 \vdash T_1 \mid T_2 \mid T_3$, where $\Gamma_1 = \Gamma, m_o, p_{o,up} : \mathfrak{o}, m_i, p_{i,up} : \mathsf{i}, c_o : \sharp(\mathsf{i})\,, c_i : \sharp(\mathfrak{o})$ and

$$\begin{aligned}T_1 &= (\nu p_o : \mathfrak{o}, p_i : \mathsf{i})\,\big(\mathcal{T}^3_\text{B}[\![\,S_1\,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftInReq})\big)\\ T_2 &= (\nu p_o : \mathfrak{o}, p_i : \mathsf{i})\,\big(\mathcal{T}^3_\text{B}[\![\,S_2\,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procRightInReq})\big)\\ T_3 &= \mathcal{T}^3_\text{B}(\mathsf{pushReq})\end{aligned}$$

Then

$$\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma_1 \vdash T_2 \mid T_3}\text{T-Par}_\text{B}}{\Gamma_1 \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_\text{B}$$

Let $\Gamma_2 = \Gamma_1, p_o : \mathfrak{o}, p_i : \mathfrak{i}$. By applying again T-Res$_\text{B}$ two times, respectively, D_1 becomes $\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_1 \,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftInReq})$ and D_2 becomes $\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_2 \,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procRightInReq})$. Let $T_4 = \mathcal{T}^3_\text{B}(\mathsf{procLeftOutReq})$, $T_5 = \mathcal{T}^3_\text{B}(\mathsf{procLeftInReq})$, $T_6 = \mathcal{T}^3_\text{B}(\mathsf{procRightOutReq})$, and $T_7 = \mathcal{T}^3_\text{B}(\mathsf{procRightInReq})$. Then

$$\dfrac{\dfrac{}{\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_1 \,]\!]^\text{m}_\text{a}}\text{(IH) and Lemma 6.2.11} \quad \dfrac{D_4 \quad D_5}{\Gamma_2 \vdash T_4 \mid T_5}\text{T-Par}_\text{B}}{\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_1 \,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procLeftInReq})}\text{T-Par}_\text{B}$$

and

$$\dfrac{\dfrac{}{\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_2 \,]\!]^\text{m}_\text{a}}\text{(IH) and Lemma 6.2.11} \quad \dfrac{D_6 \quad D_7}{\Gamma_2 \vdash T_6 \mid T_7}\text{T-Par}_\text{B}}{\Gamma_2 \vdash \mathcal{T}^3_\text{B}[\![\, S_2 \,]\!]^\text{m}_\text{a} \mid \mathcal{T}^3_\text{B}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_\text{B}(\mathsf{procRightInReq})}\text{T-Par}_\text{B}$$

Let $\Gamma_3 = \Gamma_2, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s}$.

$$D_4 = \dfrac{D_{4,1} \quad \dfrac{D_{4,2} \quad D_{4,3}}{\Gamma_3 \vdash \overline{m_o}\langle y, l, s, z\rangle \mid \overline{p_{o,up}}\langle y, l, s, z\rangle}\text{T-Par}_\text{B}}{\Gamma_2 \vdash p_o\star(y, l, s, z)\,.\,(\overline{m_o}\langle y, l, s, z\rangle \mid \overline{p_{o,up}}\langle y, l, s, z\rangle)}\text{T-Rep}_\text{B}$$

$\Gamma_2 \vdash p_o : \mathfrak{o}$ for $D_{4,1}$ follows from T-Name$_\text{B}$. Apply T-Out$_\text{B}$ and then T-Name$_\text{B}$ on all subgoals to show $\Gamma_3 \vdash \overline{m_o}\langle y, l, s, z\rangle$ for $D_{4,2}$ and $\Gamma_3 \vdash \overline{p_{o,up}}\langle y, l, s, z\rangle$ for $D_{4,3}$. Let $\Gamma_4 = \Gamma_2, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r}$.

$$D_5 = \dfrac{D_{5,1} \quad \dfrac{D_{5,2} \quad D_{5,3}}{\Gamma_4 \vdash \overline{m_i}\langle y, l, r\rangle \mid \overline{p_{i,up}}\langle y, l, r\rangle}\text{T-Par}_\text{B}}{\Gamma_2 \vdash p_i\star(y, l, r)\,.\,(\overline{m_i}\langle y, l, r\rangle \mid \overline{p_{i,up}}\langle y, l, r\rangle)}\text{T-Rep}_\text{B}$$

$\Gamma_2 \vdash p_i : \mathfrak{i}$ for $D_{5,1}$ follows from T-Name$_\text{B}$. Apply T-Out$_\text{B}$ and then T-Name$_\text{B}$ on all subgoals to show $\Gamma_4 \vdash \overline{m_i}\langle y, l, r\rangle$ for $D_{5,2}$ and $\Gamma_4 \vdash \overline{p_{i,up}}\langle y, l, r\rangle$ for $D_{5,3}$. Let

$$\begin{aligned}
T_6 &= \overline{c_o}\langle m_i\rangle \mid c_o\star(m_i)\,.p_o(y, l_s, s, z)\,.\,\big(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6'\big)\\
T_6' &= (\nu m_{i,up} : \mathfrak{i})\,\big(T_6'' \mid (\nu m_i : \mathfrak{i})\,\big(m_{i,up}\star\big(y', l_r, r\big)\,.\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle\big)\big)\\
T_6'' &= m_i\star\big(y', l_r, r\big)\,.\,\big([y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle\big)
\end{aligned}$$

Let $\Gamma_5 = \Gamma_2, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \mathfrak{s}$.

$$D_6 = \dfrac{D_{6,1} \quad \dfrac{D_{6,2} \quad \dfrac{D_{6,3} \quad \dfrac{D_{6,4} \quad D_{6,5}}{\Gamma_5 \vdash \overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6'}P}{\Gamma_2 \vdash p_o(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')}\text{T-In}_\text{B}}{\Gamma_2 \vdash c_o\star(m_i)\,.p_o(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')}\text{T-Rep}_\text{B}}{\Gamma_2 \vdash \overline{c_o}\langle m_i\rangle \mid c_o\star(m_i)\,.p_o(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')}P$$

where $P = \text{T-PAR}_\text{B}$. Apply T-OUT_B and then T-NAME_B on all subgoals to show $\Gamma_2 \vdash \overline{c_o}\langle m_i\rangle$ for $D_{6,1}$ and $\Gamma_5 \vdash \overline{p_{o,up}}\langle y, l_s, s, z\rangle$ for $D_{6,4}$. $\Gamma_2 \vdash c_o : \mathfrak{o}$ for $D_{6,2}$ and $\Gamma_2 \vdash p_o : \mathfrak{o}$ for $D_{6,3}$ follow from T-NAME_B.

$$D_{6,5} = \dfrac{\dfrac{D_{6,6} \quad \dfrac{\dfrac{D_{6,7} \quad D_{6,8}}{\Gamma_5, m_{i,up} : \mathfrak{i} \vdash m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle}\text{T-REP}_\text{B} \quad D_{6,9}}{\Gamma_5, m_{i,up} : \mathfrak{i} \vdash m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle}P}{\Gamma_5, m_{i,up} : \mathfrak{i} \vdash (\nu m_i : \mathfrak{i})\,(m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle)}R}{\Gamma_5, m_{i,up} : \mathfrak{i} \vdash T_6'' \mid (\nu m_i : \mathfrak{i})\,(m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle)}P}{\Gamma_5 \vdash T_6'}R$$

where $R = \text{T-RES}_\text{B}$ and $P = \text{T-PAR}_\text{B}$. $\Gamma_5, m_{i,up} : \mathfrak{i} \vdash m_{i,up} : \mathfrak{i}$ for $D_{6,7}$ follows from T-NAME_B. Apply T-OUT_B and then T-NAME_B on all subgoals to show $\Gamma_6 \vdash \overline{m_i}\langle y', l_r, r\rangle$ for $D_{6,8}$ and $\Gamma_5, m_{i,up} : \mathfrak{i} \vdash \overline{c_o}\langle m_i\rangle$ for $D_{6,9}$, where $\Gamma_6 = \Gamma_5, m_{i,up} : \mathfrak{i}, y' : \mathfrak{v}_\mathfrak{n}, l_r : \mathfrak{l}, r : \mathfrak{r}$.

$$D_{6,6} = \dfrac{D_{6,10} \quad \dfrac{\dfrac{D_{6,11} \quad D_{6,12} \quad D_{6,13}}{\Gamma_6 \vdash [y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle}\text{T-MAT}_\text{B} \quad D_{6,14}}{\Gamma_6 \vdash [y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle}\text{T-PAR}_\text{B}}{\Gamma_5, m_{i,up} : \mathfrak{i} \vdash m_i{}^\star(y', l_r, r)\,.\,([y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle)}\text{T-REP}_\text{B}$$

$\Gamma_5, m_{i,up} : \mathfrak{i} \vdash m_i : \mathfrak{i}$ for $D_{6,10}$, $\Gamma_6 \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{6,11}$, and $\Gamma_6 \vdash y' : \mathfrak{v}_\mathfrak{n}$ for $D_{6,12}$ follow from T-NAME_B. To show $\Gamma_6 \vdash \overline{r}\langle l_r, l_s, l_s, s, z\rangle$ for $D_{6,13}$ and $\Gamma_6 \vdash \overline{m_{i,up}}\langle y', l_r, r\rangle$ for $D_{6,14}$ apply T-OUT_B and then T-NAME_B. Let

$$\begin{aligned}
T_7 &= \overline{c_i}\langle m_o\rangle \mid c_i{}^\star(m_o)\,.p_i(y, l_r, r)\,.\,\big(\overline{p_{i,up}}\langle y, l_r, r\rangle \mid T_7'\big)\\
T_7' &= (\nu m_{o,up} : \mathfrak{o})\,\big(T_7'' \mid (\nu m_o : \mathfrak{o})\,\big(m_{o,up}{}^\star\big(y', l_s, s, z\big)\,.\overline{m_o}\big\langle y', l_s, s, z\big\rangle \mid \overline{c_i}\langle m_o\rangle\big)\big)\\
T_7'' &= m_o{}^\star\big(y', l_s, s, z\big)\,.\,\big(\big[y' = y\big]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\big\langle y', l_s, s, z\big\rangle\big)
\end{aligned}$$

Let $\Gamma_7 = \Gamma_2, y : \mathfrak{v}_\mathfrak{n}, l_r : \mathfrak{l}, r : \mathfrak{r}$.

$$D_7 = \dfrac{D_{7,1} \quad \dfrac{D_{7,2} \quad \dfrac{D_{7,3} \quad \dfrac{D_{7,4} \quad D_{7,5}}{\Gamma_7 \vdash \overline{p_{i,up}}\langle y, l_r, r\rangle \mid T_7'}P}{\Gamma_2 \vdash p_i(y, l_r, r)\,.\,(\overline{p_{i,up}}\langle y, l_r, r\rangle \mid T_7')}\text{T-IN}_\text{B}}{\Gamma_2 \vdash c_i{}^\star(m_o)\,.p_i(y, l_r, r)\,.\,(\overline{p_{i,up}}\langle y, l_r, r\rangle \mid T_7')}\text{T-REP}_\text{B}}{\Gamma_2 \vdash \overline{c_i}\langle m_o\rangle \mid c_i{}^\star(m_o)\,.p_i(y, l_r, r)\,.\,(\overline{p_{i,up}}\langle y, l_r, r\rangle \mid T_7')}P$$

where $P = \text{T-PAR}_\text{B}$. Apply T-OUT_B and then T-NAME_B on all subgoals to show $\Gamma_2 \vdash \overline{c_i}\langle m_o\rangle$ for $D_{7,1}$ and $\Gamma_7 \vdash \overline{p_{i,up}}\langle y, l_r, r\rangle :$ for $D_{7,4}$. $\Gamma_2 \vdash c_i : \mathfrak{i}$ for $D_{7,2}$ and $\Gamma_2 \vdash p_i : \mathfrak{i}$ for $D_{7,3}$ follows from T-NAME_B. $D_{7,5} =$

$$\dfrac{D_{7,6} \quad \dfrac{\dfrac{\dfrac{D_{7,7} \quad D_{7,8}}{\Gamma_7, m_{o,up} : \mathfrak{o} \vdash m_{o,up}{}^\star(y', l_s, s, z)\,.\,\overline{m_o}\langle y', l_s, s, z\rangle}\text{T-REP}_\text{B} \quad D_{7,9}}{\Gamma_7, m_{o,up} : \mathfrak{o} \vdash m_{o,up}{}^\star(y', l_s, s, z)\,.\,\overline{m_o}\langle y', l_s, s, z\rangle \mid \overline{c_i}\langle m_o\rangle}P}{\Gamma_7, m_{o,up} : \mathfrak{o} \vdash (\nu m_o : \mathfrak{o})\,(m_{o,up}{}^\star(y', l_s, s, z)\,.\,\overline{m_o}\langle y', l_s, s, z\rangle \mid \overline{c_i}\langle m_o\rangle)}R}{\Gamma_7, m_{o,up} : \mathfrak{o} \vdash T_7'' \mid (\nu m_o : \mathfrak{o})\,(m_{o,up}{}^\star(y', l_s, s, z)\,.\,\overline{m_o}\langle y', l_s, s, z\rangle \mid \overline{c_i}\langle m_o\rangle)}P}{\Gamma_7 \vdash T_7'}R$$

where $R = \text{T-Res}_\text{B}$ and $P = \text{T-Par}_\text{B}$. $\Gamma_7, m_{o,up} : \mathfrak{o} \vdash m_{o,up} : \mathfrak{o}$ for $D_{7,7}$ follows from T-Name_B. Apply T-Out_B and then T-Name_B on all subgoals to show $\Gamma_8 \vdash \overline{m_o}\langle y', l_s, s, z\rangle$ for $D_{7,8}$ and $\Gamma_7, m_{o,up} : \mathfrak{o} \vdash \overline{c_i}\langle m_o\rangle$ for $D_{7,9}$, where $\Gamma_8 = \Gamma_7, m_{o,up} : \mathfrak{o}, y', z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \mathfrak{s}$. $D_{7,6} =$

$$\dfrac{D_{7,10} \quad \dfrac{\dfrac{D_{7,11} \quad D_{7,12} \quad D_{7,13}}{\Gamma_8 \vdash [y' = y]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle}\text{T-Mat}_\text{B} \quad D_{7,14}}{\Gamma_8 \vdash [y' = y]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\langle y', l_s, s, z\rangle}\text{T-Par}_\text{B}}{\Gamma_7, m_{o,up} : \mathfrak{o} \vdash m_o^\star(y', l_s, s, z)\,.\,([y' = y]\,\overline{r}\langle l_s, l_r, l_s, s, z\rangle \mid \overline{m_{o,up}}\langle y', l_s, s, z\rangle)}\text{T-Rep}_\text{B}$$

$\Gamma_7, m_{o,up} : \mathfrak{o} \vdash m_o : \mathfrak{o}$ for $D_{7,10}$, $\Gamma_8 \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{7,11}$, and $\Gamma_8 \vdash y' : \mathfrak{v}_\mathfrak{n}$ for $D_{7,12}$ follow from T-Name_B. To show $\Gamma_8 \vdash \overline{r}\langle l_s, l_r, l_s, s, z\rangle$ for $D_{7,13}$ and $\Gamma_8 \vdash \overline{m_{o,up}}\langle y', l_s, s, z\rangle$ for $D_{7,14}$ apply T-Out_B and then T-Name_B. Finally,

$$D_3 = \dfrac{D_{3,1} \quad D_{3,2}}{\Gamma_1 \vdash p_{o,up}^\star(y, l, s, z)\,.\,\overline{p_o}\langle y, l, s, z\rangle \mid p_{i,up}^\star(y, l, r)\,.\,\overline{p_i}\langle y, l, r\rangle}\text{T-Par}_\text{B}$$

where

$$D_{3,1} = \dfrac{D_{3,3} \quad \dfrac{D_{3,4} \quad D_{3,5} \quad D_{3,6} \quad D_{3,7} \quad D_{3,8}}{\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash \overline{p_o}\langle y, l, s, z\rangle}\text{T-Out}_\text{B}}{\Gamma_1 \vdash p_{o,up}^\star(y, l, s, z)\,.\,\overline{p_o}\langle y, l, s, z\rangle}\text{T-Rep}_\text{B}$$

and

$$D_{3,2} = \dfrac{D_{3,9} \quad \dfrac{D_{3,10} \quad D_{3,11} \quad D_{3,12} \quad D_{3,13}}{\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r} \vdash \overline{p_i}\langle y, l, r\rangle}\text{T-Out}_\text{B}}{\Gamma_1 \vdash p_{i,up}^\star(y, l, r)\,.\,\overline{p_i}\langle y, l, r\rangle}\text{T-Rep}_\text{B}$$

and $\Gamma_1 \vdash p_{o,up} : \mathfrak{o}$ for $D_{3,3}$, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash p_o : \mathfrak{o}$ for $D_{3,4}$, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{3,5}$, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash l : \mathfrak{l}$ for $D_{3,6}$, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash s : \mathfrak{s}$ for $D_{3,7}$, $\Gamma_1, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s} \vdash z : \mathfrak{v}_\mathfrak{n}$ for $D_{3,8}$, $\Gamma_1 \vdash p_{i,up} : \sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{r})$ for $D_{3,9}$, $\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r} \vdash p_i : \sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{r})$ for $D_{3,10}$, $\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r} \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{3,11}$, $\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r} \vdash l : \mathfrak{l}$ for $D_{3,12}$, and $\Gamma_1, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r} \vdash r : \mathfrak{r}$ for $D_{3,13}$ follow from T-Name_B.

2. If $S = \sum_{i \in I} \pi_i.S_i$ for some $\pi_i.S_i \in \mathcal{P}_\text{m}$ then

$$\mathcal{T}_\text{B}^3[\![\, S \,]\!]_\text{a}^\text{m} = (\nu l : \mathfrak{l})\left(\mathcal{T}_\text{B}^3\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i \in I} \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}\right).$$

We have:

$$\dfrac{\dfrac{}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}_\text{B}^3\big(\overline{l}\langle\top\rangle\big)}\text{Lemma A.1.1 and Lemma 6.2.11} \quad D}{\Gamma, l : \mathfrak{l} \vdash \mathcal{T}_\text{B}^3\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i \in I} \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}}\text{T-Par}_\text{B}}{\Gamma \vdash (\nu l : \mathfrak{l})\left(\mathcal{T}_\text{B}^3\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i \in I} \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}\right)}\text{T-Res}_\text{B}$$

To prove D, we have to show that $\Gamma, l : \mathfrak{l} \vdash \prod_{i \in I} \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}$. With T-Par_B we decompose this goal into several subgoals of the form $\Gamma, l : \mathfrak{l} \vdash \mathcal{T}_\text{B}^3[\![\, \pi_i.S_i \,]\!]_\text{a}^\text{m}$, where each π_i is either a τ, an output or an input prefix.

a) If $\pi_i = \tau$ then:

$$\mathcal{T}^3_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot))\,\big(\bar{l}\langle t, f\rangle \mid t(v_t).\,\big(\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\top\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big) \mid f(v_f).\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\bot\rangle\big)\big)$$

Let T_1, T_2, T_3 be such that $T_1 = \bar{l}\langle t, f\rangle$, $T_2 = t(v_t).\,\big(\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\top\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)$, and $T_3 = f(v_f).\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\bot\rangle\big)$.

$$\dfrac{\dfrac{\dfrac{D_1 \quad \dfrac{D_2 \quad D_3}{\Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot) \vdash T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot) \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_{\mathrm{B}}}{\Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top) \vdash (\nu f{:}\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}}{\Gamma, l{:}\mathfrak{l} \vdash (\nu t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot))\,(T_1 \mid T_2 \mid T_3)}\text{T-Res}_{\mathrm{B}}$$

where $\Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot) \vdash \bar{l}\langle t, f\rangle$ for D_1 follows from T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ for all subgoals. Let $\Gamma_1 = \Gamma, l{:}\mathfrak{l}, t{:}\sharp(\mathfrak{v}_\top), f{:}\sharp(\mathfrak{v}_\bot)$.

$$D_2 = \dfrac{D_{2,1} \quad \dfrac{D_{2,2} \quad D_{2,3}}{\Gamma_1, v_t{:}\mathfrak{v}_\top \vdash \mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\top\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-Par}_{\mathrm{B}}}{\Gamma_1 \vdash t(v_t).\,\big(\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\top\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)}\text{T-In}_{\mathrm{B}}$$

$\Gamma_1 \vdash t : \sharp(\mathfrak{v}_\top)$ for $D_{2,1}$ follows from T-Name$_{\mathrm{B}}$. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_1, v_t{:}\mathfrak{v}_\top \vdash \mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\top\rangle\big)$ for $D_{2,2}$. $\Gamma_1, v_t{:}\mathfrak{v}_\top \vdash \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ for $D_{2,3}$ follows from the induction hypothesis and Lemma 6.2.11.

$$D_3 = \dfrac{D_{3,1} \quad D_{3,2}}{\Gamma_1 \vdash f(v_f).\mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\bot\rangle\big)}\text{T-In}_{\mathrm{B}}$$

$\Gamma_1 \vdash f{:}\sharp(\mathfrak{v}_\bot)$ for $D_{3,1}$ follows from T-Name$_{\mathrm{B}}$. Finally, by Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_1, v_f{:}\mathfrak{v}_\bot \vdash \mathcal{T}^3_{\mathrm{B}}\big(\bar{l}\langle\bot\rangle\big)$ for $D_{3,2}$.

b) If $\pi_i = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$ then

$$\mathcal{T}^3_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}))\,\big(\overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s).\mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big).$$

Because $\Gamma(p_o) = \sharp(\mathfrak{v}_{\mathfrak{n}}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_{\mathfrak{n}})$ and $\Gamma(\varphi^{\mathrm{m}}_{\mathrm{a}}(y)) = \Gamma(\varphi^{\mathrm{m}}_{\mathrm{a}}(z)) = \mathfrak{v}_{\mathfrak{n}}$, we have

$$\dfrac{\dfrac{D_1 \quad \dfrac{\overline{\Gamma', s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}) \vdash s{:}\sharp(\mathfrak{v}_{\mathfrak{s}})}\text{T-Name}_{\mathrm{B}} \quad D_2}{\Gamma', s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}) \vdash s(v_s).\mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-In}_{\mathrm{B}}}{\Gamma', s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}) \vdash \overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s).\mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-Par}_{\mathrm{B}}}{\Gamma' \vdash (\nu s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}))\,\big(\overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s).\mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)}\text{T-Res}_{\mathrm{B}}$$

for $\Gamma' = \Gamma, l{:}\mathfrak{l}$ where

$$D_1 = \dfrac{D_{1,1} \quad D_{1,2} \quad D_{1,3} \quad D_{1,4} \quad D_{1,5}}{\Gamma, l{:}\mathfrak{l}, s{:}\sharp(\mathfrak{v}_{\mathfrak{s}}) \vdash \overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle}\text{T-Out}_{\mathrm{B}}$$

and

$$D_2 = \frac{}{\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}), v_s:\mathfrak{v}_\mathfrak{s} \vdash \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{(IH) and Lemma 6.2.11}$$

and the remaining subgoals $\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}) \vdash p_o:\sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_\mathfrak{n})$ for $D_{1,1}$, $\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}) \vdash \varphi^{\mathrm{m}}_{\mathrm{a}}(y):\mathfrak{v}_\mathfrak{n}$ for $D_{1,2}$, $\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}) \vdash l:\mathfrak{l}$ for $D_{1,3}$, $\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}) \vdash s:\sharp(\mathfrak{v}_\mathfrak{s})$ for $D_{1,4}$, and $\Gamma, l:\mathfrak{l}, s:\sharp(\mathfrak{v}_\mathfrak{s}) \vdash \varphi^{\mathrm{m}}_{\mathrm{a}}(z):\mathfrak{v}_\mathfrak{n}$ for $D_{1,5}$ follow from T-Name$_{\mathrm{B}}$.

c) If $\pi_i = y(x)$ for some $x, y \in \mathcal{N}$ then

$$\begin{aligned}
\mathcal{T}^3_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} &= (\nu r:\mathfrak{r})\,(\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid T_1)\\
T_1 &= r^\star(l_1, l_2, l_\mathfrak{s}, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x)).\,((\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_2 \mid T_3 \mid T_7))\\
T_2 &= \overline{l_1}\langle t, f\rangle\\
T_3 &= t(v_t).\,((\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_6))\\
T_4 &= \overline{l_2}\langle t, f\rangle\\
T_5 &= t(v_t).\big(\mathcal{T}^3_{\mathrm{B}}\big(\overline{l_1}\langle\bot\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}\big(\overline{l_2}\langle\bot\rangle\big) \mid (\nu v_s:\mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \mathcal{T}^3_{\mathrm{B}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)\\
T_6 &= f(v_f).\,\big(\mathcal{T}^3_{\mathrm{B}}\big(\overline{l_1}\langle\top\rangle\big) \mid \mathcal{T}^3_{\mathrm{B}}\big(\overline{l_2}\langle\bot\rangle\big)\big)\\
T_7 &= f(v_f).\mathcal{T}^3_{\mathrm{B}}\big(\overline{l_1}\langle\bot\rangle\big)
\end{aligned}$$

Because $\Gamma(p_i) = \sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{r})$ and $\Gamma(\varphi^{\mathrm{m}}_{\mathrm{a}}(y)) = \mathfrak{v}_\mathfrak{n}$, we have

$$\dfrac{\dfrac{\dfrac{D_{1,1} \quad D_{1,2} \quad D_{1,3} \quad D_{1,4}}{\Gamma, l:\mathfrak{l}, r:\sharp(\mathfrak{l},\mathfrak{l},\mathfrak{l},\mathfrak{s},\mathfrak{v}_\mathfrak{n}) \vdash \overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle}\text{T-Out}_{\mathrm{B}} \quad D_1}{\Gamma, l:\mathfrak{l}, r:\sharp(\mathfrak{l},\mathfrak{l},\mathfrak{l},\mathfrak{s},\mathfrak{v}_\mathfrak{n}) \vdash \overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid T_1}\text{T-Par}_{\mathrm{B}}}{\Gamma, l:\mathfrak{l} \vdash \mathcal{T}^3_{\mathrm{B}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-Res}_{\mathrm{B}}$$

where the subgoals $\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash p_i:\sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{r})$ for $D_{1,1}$, $\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash \varphi^{\mathrm{m}}_{\mathrm{a}}(y):\mathfrak{v}_\mathfrak{n}$ for $D_{1,2}$, $\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash l:\mathfrak{l}$ for $D_{1,3}$, and $\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash r:\mathfrak{r}$ for $D_{1,4}$ follow from T-Name$_{\mathrm{B}}$. Let $\Gamma_1 = \Gamma, l, l_1, l_2, l_\mathfrak{s}:\mathfrak{l}, r:\mathfrak{r}, s:\mathfrak{s}$ and $\Gamma_2 = \Gamma_1, t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot)$.

$$D_1 = \dfrac{D'_1 \quad \dfrac{\dfrac{D_2 \quad \dfrac{D_3 \quad D_7}{\Gamma_2 \vdash T_3 \mid T_7}\text{T-Par}_{\mathrm{B}}}{\Gamma_2 \vdash T_2 \mid T_3 \mid T_7}\text{T-Par}_{\mathrm{B}}}{\Gamma_1, t:\sharp(\mathfrak{v}_\top) \vdash (\nu f:\sharp(\mathfrak{v}_\bot))\,(T_2 \mid T_3 \mid T_7)}\text{T-Res}_{\mathrm{B}}}{\Gamma_1 \vdash (\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_2 \mid T_3 \mid T_7)}\text{T-Res}_{\mathrm{B}}}{\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash T_1}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma, l:\mathfrak{l}, r:\mathfrak{r} \vdash r:\mathfrak{r}$ for D'_1 follows from T-Name$_{\mathrm{B}}$. Apply T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ on each subgoal to prove $\Gamma_2 \vdash \overline{l_1}\langle t, f\rangle$ for D_2. $D_3 =$

$$\dfrac{D'_3 \quad \dfrac{\dfrac{\dfrac{D_4 \quad \dfrac{D_5 \quad D_6}{\Gamma_2, v_t:\mathfrak{v}_\top \vdash T_5 \mid T_6}\text{T-Par}_{\mathrm{B}}}{\Gamma_2, v_t:\mathfrak{v}_\top \vdash T_4 \mid T_5 \mid T_6}\text{T-Par}_{\mathrm{B}}}{\Gamma_2, v_t:\mathfrak{v}_\top \vdash (\nu f:\sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_6)}\text{T-Res}_{\mathrm{B}}}{\Gamma_2, v_t:\mathfrak{v}_\top \vdash (\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_6)}\text{T-Res}_{\mathrm{B}}}{\Gamma_2 \vdash t(v_t).\,((\nu t:\sharp(\mathfrak{v}_\top), f:\sharp(\mathfrak{v}_\bot))\,(T_4 \mid T_5 \mid T_6))}\text{T-In}_{\mathrm{B}}$$

Again, $\Gamma_2 \vdash t : \sharp(\mathfrak{v}_\top)$ for D'_3 follows from T-Name$_\mathrm{B}$. Apply T-Out$_\mathrm{B}$ and then T-Name$_\mathrm{B}$ on each subgoal to prove $\Gamma_2, v_t : \mathfrak{v}_\top \vdash \overline{l_2}\langle t, f\rangle$ for D_4. Let $T'_5 = (\nu v_s : \mathfrak{v}_\mathfrak{s})\, \overline{s}\langle v_s\rangle$ and $\Gamma_3 = \Gamma_2, v_t : \mathfrak{v}_\top$.

$$D_5 = \dfrac{D_{5,1} \quad \dfrac{D_{5,2} \quad \dfrac{D_{5,3} \quad \dfrac{\dfrac{D_{5,4}}{\Gamma_3 \vdash T'_5} R \quad D_{5,5}}{\Gamma_3 \vdash T'_5 \mid \mathcal{T}^3_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{a}} P}{\Gamma_3 \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\bot\rangle) \mid T'_5 \mid \mathcal{T}^3_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{a}} P}{\Gamma_3 \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle) \mid \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\bot\rangle) \mid T'_5 \mid \mathcal{T}^3_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{a}} P}{\Gamma_3 \vdash T_5}\text{T-In}_\mathrm{B}$$

where $P =$ T-Par$_\mathrm{B}$ and $R =$ T-Res$_\mathrm{B}$. Again, $\Gamma_3 \vdash t : \sharp(\mathfrak{v}_\top)$ for $D_{5,1}$ follows from T-Name$_\mathrm{B}$. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_3 \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle)$ for $D_{5,2}$ and $\Gamma_3 \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\bot\rangle)$ for $D_{5,3}$. To show $\Gamma_3, v_s : \mathfrak{v}_\mathfrak{s} \vdash \overline{s}\langle v_s\rangle$ for $D_{5,4}$ apply T-Out$_\mathrm{B}$ and then T-Name$_\mathrm{B}$ for both subgoals. $\Gamma_3 \vdash \mathcal{T}^3_\mathrm{B}[\![\, S_i \,]\!]^\mathrm{m}_\mathrm{a}$ for $D_{5,5}$ follows from the induction hypothesis and Lemma 6.2.11.

$$D_6 = \dfrac{D_{6,1} \quad \dfrac{D_{6,2} \quad D_{6,3}}{\Gamma_3, v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle) \mid \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\top\rangle)}\text{T-Par}_\mathrm{B}}{\Gamma_3 \vdash f(v_f)\,.\,(\mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle) \mid \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\top\rangle))}\text{T-In}_\mathrm{B}$$

$\Gamma_3 \vdash f : \sharp(\mathfrak{v}_\bot)$ for $D_{6,1}$ follows from T-Name$_\mathrm{B}$. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_3, v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle)$ for $D_{6,2}$ and $\Gamma_3, v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_2}\langle\top\rangle)$ for $D_{6,3}$.

$$D_7 = \dfrac{D_{7,1} \quad D_{7,2}}{\Gamma_2 \vdash f(v_f)\,.\mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle)}\text{T-In}_\mathrm{B}$$

$\Gamma_2 \vdash f : \sharp(\mathfrak{v}_\bot)$ for $D_{7,1}$ follows from T-Name$_\mathrm{B}$. Finally, by Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_2, v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^3_\mathrm{B}(\overline{l_1}\langle\bot\rangle)$ for $D_{7,2}$.

3. If $S = y^\star(x)\,.S_2$ for some $x, y \in \mathcal{N}$ and $S_2 \in \mathcal{P}_\mathrm{m}$ then:

$$\begin{aligned}\mathcal{T}^3_\mathrm{B}[\![\, S \,]\!]^\mathrm{m}_\mathrm{a} = (\nu l : \mathfrak{l}, r : \mathfrak{r}, c_{r1} : \sharp(\mathfrak{v}_\mathfrak{n})\,, c_{r2} : \sharp(\mathfrak{o}, \mathfrak{i})\,, r_o : \mathfrak{o}, r_i : \mathfrak{i})\ &\big(\overline{p_i}\langle\varphi^\mathrm{m}_\mathrm{a}(y)\,, l, r\rangle \\ &\mid r^\star(l_1, l_2, l_s, s, z)\,.\,(\nu t : \sharp(\mathfrak{v}_\top)\,, f : \sharp(\mathfrak{v}_\bot))\ \big(\overline{l_s}\langle t, f\rangle \\ &\quad \mid t(v_t)\,.\,\big(\mathcal{T}^3_\mathrm{B}(\overline{l_s}\langle\bot\rangle) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle\big) \mid f(v_f)\,.\mathcal{T}^3_\mathrm{B}(\overline{l_s}\langle\bot\rangle)\big)\ \big) \\ &\mid \overline{r_i}\langle\varphi^\mathrm{m}_\mathrm{a}(y)\,, l, r\rangle \mid \mathcal{T}^3_\mathrm{B}(\overline{l}\langle\top\rangle) \mid \mathcal{T}^3_\mathrm{B}(\mathsf{encodedContinuations})\ \big)\end{aligned}$$

By applying T-Res$_\mathrm{B}$ several times, $\Gamma \vdash \mathcal{T}^3_\mathrm{B}[\![\, S \,]\!]^\mathrm{m}_\mathrm{a}$ becomes $\Gamma_1 \vdash T_1 \mid T_2 \mid T_3 \mid T_4 \mid$

T_5 where $\Gamma_1 = \Gamma, l : \mathfrak{l}, r : \mathfrak{r}, c_{r1} : \sharp(\mathfrak{v}_\mathfrak{n}), c_{r2} : \sharp(\mathfrak{o}, \mathfrak{i}), r_o : \mathfrak{o}, r_i : \mathfrak{i}$ and

$$
\begin{aligned}
T_1 &= \overline{p_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle \\
T_2 &= r^\star(l_1, l_2, l_s, s, z).\,(\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,\big(\overline{l_s}\langle t, f\rangle \\
&\qquad \mid t(v_t).\,\big(\mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle\big) \mid f(v_f).\,\mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big)\,\big) \\
T_3 &= \overline{r_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle \\
T_4 &= \mathcal{T}_{\mathrm{B}}^3\big(\overline{l}\langle\top\rangle\big) \\
T_5 &= \mathcal{T}_{\mathrm{B}}^3(\mathsf{encodedContinuations})
\end{aligned}
$$

Then

$$
\dfrac{D_1 \quad \dfrac{D_2 \quad \dfrac{D_3 \quad \dfrac{D_4 \quad D_5}{\Gamma_1 \vdash T_4 \mid T_5}\text{T-Par}_\text{B}}{\Gamma_1 \vdash T_3 \mid T_4 \mid T_5}\text{T-Par}_\text{B}}{\Gamma_1 \vdash T_2 \mid T_3 \mid T_4 \mid T_5}\text{T-Par}_\text{B}}{\Gamma_1 \vdash T_1 \mid T_2 \mid T_3 \mid T_4 \mid T_5}\text{T-Par}_\text{B}
$$

Since $y \in \mathsf{fn}(S)$, we have $\varphi_{\mathrm{a}}^{\mathrm{m}}(y) : \mathfrak{v}_\mathfrak{n} \in \Gamma$. Moreover, $\Gamma\,(p_i) = \sharp(\mathfrak{v}_\mathfrak{n}, \mathfrak{l}, \mathfrak{r})$. Hence, apply T-Out$_\text{B}$ and the T-Name$_\text{B}$ on all subgoals to show $\Gamma_1 \vdash \overline{p_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle$ for D_1 and $\Gamma_1 \vdash \overline{r_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y), l, r\rangle$ for D_3. $\Gamma_1 \vdash \mathcal{T}_{\mathrm{B}}^3\big(\overline{l}\langle\top\rangle\big)$ for D_4 follows from Lemma A.1.1 and Lemma 6.2.11. Let $T_2' = t(v_t).\,\big(\mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle\big)$, $T_2'' = f(v_f).\,\mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big)$, and $\Gamma_2 = \Gamma_1, l_1, l_2, l_s : \mathfrak{l}, s : \mathfrak{s}, z : \mathfrak{v}_\mathfrak{n}$.

$$
D_2 = \dfrac{D_{2,1} \quad \dfrac{D_{2,2} \quad \dfrac{\dfrac{D_{2,3} \quad D_{2,4}}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash T_2' \mid T_2''}P}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash \overline{l_s}\langle t, f\rangle \mid T_2' \mid T_2''}P}{\Gamma_2, t : \sharp(\mathfrak{v}_\top) \vdash (\nu f : \sharp(\mathfrak{v}_\bot))\,\big(\overline{l_s}\langle t, f\rangle \mid T_2' \mid T_2''\big)}R}{\Gamma_2 \vdash (\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,\big(\overline{l_s}\langle t, f\rangle \mid T_2' \mid T_2''\big)}R}{\Gamma_1 \vdash r^\star(l_1, l_2, l_s, s, z).\,(\nu t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot))\,\big(\overline{l_s}\langle t, f\rangle \mid T_2' \mid T_2''\big)}\text{T-Rep}_\text{B}
$$

where $P = \text{T-Par}_\text{B}$ and $R = \text{T-Res}_\text{B}$. $\Gamma_1 \vdash r : \sharp(\mathfrak{l}, \mathfrak{l}, \mathfrak{l}, \mathfrak{s}, \mathfrak{v}_\mathfrak{n})$ for $D_{2,1}$ follows from T-Name$_\text{B}$. To show $\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash \overline{l_s}\langle t, f\rangle$ for $D_{2,2}$ apply T-Out$_\text{B}$ and then T-Name$_\text{B}$ on all subgoals. Let $\Gamma_3 = \Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot), v_t : \mathfrak{v}_\top$.

$$
D_{2,3} = \dfrac{D_{2,5} \quad \dfrac{D_{2,6} \quad \dfrac{\dfrac{D_{2,7}}{\Gamma_3 \vdash (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle}\text{T-Res}_\text{B} \quad D_{2,8}}{\Gamma_3 \vdash (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle}\text{T-Par}_\text{B}}{\Gamma_3 \vdash \mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle}\text{T-Par}_\text{B}}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash t(v_t).\,\big(\mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big) \mid (\nu v_s : \mathfrak{v}_\mathfrak{s})\,\overline{s}\langle v_s\rangle \mid \overline{c_{r1}}\langle z\rangle\big)}\text{T-In}_\text{B}
$$

$\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash t : \sharp(\mathfrak{v}_\top)$ for $D_{2,5}$ follows from T-Name$_\text{B}$. By Lemma A.1.1 and Lemma 6.2.11, we have $\Gamma_3 \vdash \mathcal{T}_{\mathrm{B}}^3\big(\overline{l_s}\langle\bot\rangle\big)$ for $D_{2,6}$. Apply T-Out$_\text{B}$ and then

T-NAME$_\mathrm{B}$ on all subgoals to show $\Gamma_3, v_s : \mathfrak{v}_\mathfrak{s} \vdash \overline{s}\langle v_s \rangle$ for $D_{2,2}$ and $\Gamma_3 \vdash \overline{c_{r1}}\langle z \rangle$ for $D_{2,8}$.

$$D_{2,4} = \frac{D_{2,9} \quad D_{2,10}}{\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash f(v_f) . \mathcal{T}^3_\mathrm{B}\big(\overline{l_s}\langle \bot \rangle\big)}\,\text{T-IN}_\mathrm{B}$$

$\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot) \vdash f : \sharp(\mathfrak{v}_\bot)$ for $D_{2,9}$ follows from T-NAME$_\mathrm{B}$. $\Gamma_2, t : \sharp(\mathfrak{v}_\top), f : \sharp(\mathfrak{v}_\bot), v_f : \mathfrak{v}_\bot \vdash \mathcal{T}^3_\mathrm{B}\big(\overline{l_s}\langle \bot \rangle\big)$ for $D_{2,10}$ follows from Lemma A.1.1 and Lemma 6.2.11. We further decompose $T_5 = \mathcal{T}^3_\mathrm{B}(\mathsf{encodedContinuations})$ into:

$$\begin{aligned}
T_5 &= \overline{c_{r2}}\langle r_o, r_i \rangle \mid c_{r1}{}^\star(\varphi^\mathrm{m}_\mathrm{a}(x)) . c_{r2}(r_o, r_i) . T_6 \\
T_6 &= (\nu m_o, p_{o,up}, r_{o,up} : \mathfrak{o}, m_i, p_{i,up}, r_{i,up} : \mathfrak{i}, c_o : \sharp(\mathfrak{i}), c_i : \sharp(\mathfrak{o}))\, T_7 \\
T_7 &= \mathcal{T}^3_\mathrm{B}(\mathsf{pushReqIn}) \mid T_8 \mid T_9 \\
T_8 &= (\nu p_o : \mathfrak{o}, p_i : \mathfrak{i})\, \big(\mathcal{T}^3_\mathrm{B}[\![\, S_2 \,]\!]^\mathrm{m}_\mathrm{a} \mid \mathcal{T}^3_\mathrm{B}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_\mathrm{B}(\mathsf{procRightInReq})\big) \\
T_9 &= (\nu r_o : \mathfrak{o}, r_i : \mathfrak{i})\, \big(\overline{c_{r2}}\langle r_o, r_i \rangle \mid \mathcal{T}^3_\mathrm{B}(\mathsf{pushReqOut})\big)
\end{aligned}$$

Then

$$D_5 = \frac{D_{5,1} \quad \dfrac{D_{5,2} \quad \dfrac{D_{5,3} \quad D_6}{\Gamma_1, \varphi^\mathrm{m}_\mathrm{a}(x) : \mathfrak{v}_\mathfrak{n} \vdash c_{r2}(r_o, r_i) . T_6}\,\text{T-IN}_\mathrm{B}}{\Gamma_1 \vdash c_{r1}{}^\star(\varphi^\mathrm{m}_\mathrm{a}(x)) . c_{r2}(r_o, r_i) . T_6}\,\text{T-REP}_\mathrm{B}}{\Gamma_1 \vdash \overline{c_{r2}}\langle r_o, r_i \rangle \mid c_{r1}{}^\star(\varphi^\mathrm{m}_\mathrm{a}(x)) . c_{r2}(r_o, r_i) . T_6}\,\text{T-PAR}_\mathrm{B}$$

To show $\Gamma_1 \vdash \overline{c_{r2}}\langle r_o, r_i \rangle$ for $D_{5,1}$ apply T-OUT$_\mathrm{B}$ and then T-NAME$_\mathrm{B}$ on all subgoals. $\Gamma_1 \vdash c_{r1} : \sharp(\mathfrak{v}_\mathfrak{n})$ for $D_{5,2}$ and $\Gamma_1, \varphi^\mathrm{m}_\mathrm{a}(x) : \mathfrak{v}_\mathfrak{n} \vdash c_{r2} : \sharp(\mathfrak{o}, \mathfrak{i})$ for $D_{5,3}$ follow from T-NAME$_\mathrm{B}$. Applying several times T-RES$_\mathrm{B}$ on the subgoal $\Gamma_1, \varphi^\mathrm{m}_\mathrm{a}(x) : \mathfrak{v}_\mathfrak{n} \vdash T_6$ for D_6 leads to $\Gamma_4 \vdash T_7$, where $\Gamma_4 = \Gamma_1, \varphi^\mathrm{m}_\mathrm{a}(x) : \mathfrak{v}_\mathfrak{n}, m_o, p_{o,up}, r_{o,up} : \mathfrak{o}, m_i, p_{i,up}, r_{i,up} : \mathfrak{i}, c_o : \sharp(\mathfrak{i}), c_i : \sharp(\mathfrak{o})$. Note that $\mathcal{T}^3_\mathrm{B}(\mathsf{pushReqIn}) = T'_7 \mid T''_7$, where $T'_7 = r_o{}^\star(y, l, s, z) . (T_{7,1} \mid T_{7,2})$, $T''_7 = r_i{}^\star(y, l, r) . (T_{7,3} \mid T_{7,4})$, $T_{7,1} = \overline{m_o}\langle y, l, s, z \rangle$, $T_{7,2} = \overline{r_{o,up}}\langle y, l, s, z \rangle$, $T_{7,3} = \overline{m_i}\langle y, l, r \rangle$, and $T_{7,4} = \overline{r_{i,up}}\langle y, l, r \rangle$. Let $\Gamma_5 = \Gamma_4, y, z : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, s : \mathfrak{s}$ and $\Gamma_6 = \Gamma_4, y : \mathfrak{v}_\mathfrak{n}, l : \mathfrak{l}, r : \mathfrak{r}$. Then

$$\frac{\dfrac{D_{7,1} \quad \dfrac{D_{7,2} \quad D_{7,3}}{\Gamma_5 \vdash T_{7,1} \mid T_{7,2}}\,P}{\Gamma_4 \vdash T'_7}\,R \quad \dfrac{D_{7,4} \quad \dfrac{D_{7,5} \quad D_{7,6}}{\Gamma_6 \vdash T_{7,3} \mid T_{7,4}}\,P}{\Gamma_4 \vdash T''_7}\,R}{\Gamma_4 \vdash r_o{}^\star(y, l, s, z) . (T_{7,1} \mid T_{7,2}) \mid r_i{}^\star(y, l, r) . (T_{7,3} \mid T_{7,4})}\,P \quad \frac{D_8 \quad D_9}{\Gamma_4 \vdash T_8 \mid T_9}\,P}{\Gamma_4 \vdash T_7}\,P$$

where $P =$ T-PAR$_\mathrm{B}$ and $R =$ T-REP$_\mathrm{B}$. $\Gamma_4 \vdash r_o : \mathfrak{o}$ for $D_{7,1}$ and $\Gamma_4 \vdash r_i : \mathfrak{i}$ for $D_{7,4}$ follow from T-NAME$_\mathrm{B}$. Apply T-OUT$_\mathrm{B}$ and then T-NAME$_\mathrm{B}$ on all subgoals to show $\Gamma_5 \vdash \overline{m_o}\langle y, l, s, z \rangle$ for $D_{7,2}$, $\Gamma_5 \vdash \overline{r_{o,up}}\langle y, l, s, z \rangle$ for $D_{7,3}$, $\Gamma_6 \vdash \overline{m_i}\langle y, l, r \rangle$ for $D_{7,5}$, and $\Gamma_6 \vdash \overline{r_{i,up}}\langle y, l, r \rangle$ for $D_{7,6}$.

Apart from the type environment $\Gamma_4 \vdash T_8$ for D_8 is equal to the goal of D_2 in the first case. Hence, it can be proved by the same argumentation and the Lemmata 6.2.10 and 6.2.11.

Finally, note that $\mathcal{T}^3_{\mathrm{B}}(\mathsf{pushReqOut}) = T_{9,1} \mid T_{9,2} \mid T_{9,3} \mid T_{9,4}$, where

$$\begin{aligned}
T_{9,1} &= p_{o,up}{}^{\star}(y,l,s,z)\,.\,(\overline{p_o}\langle y,l,s,z\rangle \mid \overline{r_o}\langle y,l,s,z\rangle)\\
T_{9,2} &= r_{o,up}{}^{\star}(y,l,s,z)\,.\overline{r_o}\langle y,l,s,z\rangle\\
T_{9,3} &= p_{i,up}{}^{\star}(y,l,r)\,.\,(\overline{p_i}\langle y,l,r\rangle \mid \overline{r_i}\langle y,l,r\rangle)\\
T_{9,4} &= p_{i,up}{}^{\star}(y,l,r)\,.\overline{r_i}\langle y,l,r\rangle
\end{aligned}$$

Then

$$D_9 = \dfrac{\dfrac{\dfrac{\dfrac{D'_9 \quad \dfrac{D_{9,1} \quad \dfrac{D_{9,2} \quad \dfrac{D_{9,3} \quad D_{9,4}}{\Gamma_4 \vdash T_{9,3} \mid T_{9,4}}P}{\Gamma_4 \vdash T_{9,2} \mid T_{9,3} \mid T_{9,4}}P}{\Gamma_4 \vdash T_{9,1} \mid T_{9,2} \mid T_{9,3} \mid T_{9,4}}P}{\Gamma_4 \vdash \overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathcal{T}^3_{\mathrm{B}}(\mathsf{pushReqOut})}P}{\Gamma_4 \vdash (\nu r_i : \mathtt{i})\,\big(\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathcal{T}^3_{\mathrm{B}}(\mathsf{pushReqOut})\big)}\text{T-Res}_{\mathrm{B}}}{\Gamma_4 \vdash (\nu r_o : \mathtt{o}, r_i : \mathtt{i})\,\big(\overline{c_{r2}}\langle r_o, r_i\rangle \mid \mathcal{T}^3_{\mathrm{B}}(\mathsf{pushReqOut})\big)}\text{T-Res}_{\mathrm{B}}$$

where $P = \text{T-Par}_{\mathrm{B}}$. Apply T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ on all subgoals to show $\Gamma_4 \vdash \overline{c_{r2}}\langle r_o, r_i\rangle$ for D'_9.

$$D_{9,1} = \dfrac{D_{10,1} \quad \dfrac{D_{10,2} \quad D_{10,3}}{\Gamma_5 \vdash \overline{p_o}\langle y,l,s,z\rangle \mid \overline{r_o}\langle y,l,s,z\rangle}\text{T-Par}_{\mathrm{B}}}{\Gamma_4 \vdash p_{o,up}{}^{\star}(y,l,s,z)\,.\,(\overline{p_o}\langle y,l,s,z\rangle \mid \overline{r_o}\langle y,l,s,z\rangle)}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_4 \vdash p_{o,up} : \mathtt{o}$ for $D_{10,1}$ follows from T-Name$_{\mathrm{B}}$. Apply T-Out$_{\mathrm{B}}$ and afterwards T-Name$_{\mathrm{B}}$ on all subgoals to show $\Gamma_5 \vdash \overline{p_o}\langle y,l,s,z\rangle$ for $D_{10,2}$ and $\Gamma_5 \vdash \overline{r_o}\langle y,l,s,z\rangle$ for $D_{10,3}$.

$$D_{9,2} = \dfrac{D_{10,4} \quad D_{10,5}}{\Gamma_4 \vdash r_{o,up}{}^{\star}(y,l,s,z)\,.\overline{r_o}\langle y,l,s,z\rangle}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_4 \vdash r_{o,up} : \mathtt{o}$ for $D_{10,4}$ follows from T-Name$_{\mathrm{B}}$. Again, apply T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ on all subgoals to show $\Gamma_5 \vdash \overline{r_o}\langle y,l,s,z\rangle$ for $D_{10,5}$.

$$D_{9,3} = \dfrac{D_{10,6} \quad \dfrac{D_{10,7} \quad D_{10,8}}{\Gamma_6 \vdash \overline{p_i}\langle y,l,r\rangle \mid \overline{r_i}\langle y,l,r\rangle}\text{T-Par}_{\mathrm{B}}}{\Gamma_4 \vdash p_{i,up}{}^{\star}(y,l,r)\,.\,(\overline{p_i}\langle y,l,r\rangle \mid \overline{r_i}\langle y,l,r\rangle)}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_4 \vdash p_{i,up} : \mathtt{i}$ for $D_{10,6}$ follows from T-Name$_{\mathrm{B}}$. Apply T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ on all subgoals to show $\Gamma_6 \vdash \overline{p_i}\langle y,l,r\rangle$ for $D_{10,7}$ and $\Gamma_6 \vdash \overline{r_i}\langle y,l,r\rangle$ for $D_{10,8}$.

$$D_{9,4} = \dfrac{D_{10,9} \quad D_{10,10}}{\Gamma_4 \vdash r_{i,up}{}^{\star}(y,l,r)\,.\overline{r_i}\langle y,l,r\rangle}\text{T-Rep}_{\mathrm{B}}$$

$\Gamma_4 \vdash r_{i,up} : \mathtt{i}$ for $D_{10,9}$ follows from T-Name$_{\mathrm{B}}$. Again, apply T-Out$_{\mathrm{B}}$ and then T-Name$_{\mathrm{B}}$ on all subgoals to show $\Gamma_6 \vdash \overline{r_i}\langle y,l,r\rangle$ for $D_{10,10}$.

A.1.2. Properties of the Monadic Type System

Within this section we present the proofs of some properties of the monadic type system discussed in Section 6.2.3. In the following we show strengthening and weakening in the monadic type system.

Proof of Lemma 6.2.29. We perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \}$ be the set of processes of the target language of the considered encoding.

Base Case: If $\Gamma, x{:}T; \Delta; \Psi \vdash P$ can be derived from one of the axioms then either $P = \mathsf{0}$ or $P = \checkmark$ and $\Delta = \Psi = \emptyset$. In both cases $\Gamma; \Delta; \Psi \vdash P$ follows again directly by T-NIL$_{\mathrm{M}}$ or T-SUCC$_{\mathrm{M}}$.

Induction Hypothesis:

$$\forall P \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}}) \ . \Gamma, x{:}T; \Delta; \Psi \vdash P \wedge x \notin \mathsf{fn}(P) \quad \text{imply} \quad \Gamma; \Delta; \Psi \vdash P$$

Induction Step: We perform a case split on the inference rules in Figure 6.8. As in Lemma 6.2.10, each case is simple. Nonetheless, we present them in full detail, to show the influence of the additional sets in judgements.

Case of T-RES-B$_{\mathrm{M}}$ **:** In this case, $P = (\nu x'{:}T')\, P'$ for some $x' \in \mathcal{N}$, $T' \in \mathbb{T}_{\mathrm{M}}$, $T' \neq \circ \triangleright T''$ for all T'', $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Gamma, x{:}T, x'{:}T'; \Delta; \Psi \vdash P'$, and $\Delta = \emptyset$. Without loss of generality, let us assume that $x' \neq x$, else apply α-conversion first. Since $x' \neq x$ and $x \notin \mathsf{fn}(P)$, we have $x \notin \mathsf{fn}(P')$. Thus, by the induction hypothesis, $\Gamma, x'{:}T'; \Delta; \Psi \vdash P'$. Then $\Gamma; \Delta; \Psi \vdash P$ follows from T-RES-B$_{\mathrm{M}}$.

Case of T-RES-M$_{\mathrm{M}}$ **:** Again $P = (\nu x'{:}T')\, P'$ for some $x' \in \mathcal{N}$, $x' \neq x$, $T' \in \mathbb{T}_{\mathrm{M}}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, and $x \notin \mathsf{fn}(P')$ but here $T' = T_1 \triangleright T_2$ for some T_1, T_2 and $\Gamma, x : T, x' : T'; \Delta, \Delta'; \Psi \vdash P'$, where Δ' is either $x' : T_2$ or $\emptyset$. Thus, by the induction hypothesis, we have $\Gamma, x'{:}T'; \Delta, \Delta'; \Psi \vdash P'$, and conclude again by T-RES-M$_{\mathrm{M}}$.

Case of T-PAR$_{\mathrm{M}}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $x \notin \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$, $\Delta = \Delta_1, \Delta_2$, $\Psi = \Psi_1 \cdot \Psi_2$, $\Gamma, x : T; \Delta_1; \Psi_1 \vdash P_1$, and $\Gamma, x : T; \Delta_2; \Psi_2 \vdash P_2$. Thus, by the induction hypothesis, $\Gamma; \Delta_1; \Psi_1 \vdash P_1$ and $\Gamma; \Delta_2; \Psi_2 \vdash P_2$. We conclude by T-PAR$_{\mathrm{M}}$.

Case of T-MAT$_{\mathrm{M}}$ **:** In this case, $P = [a = b]\, P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $x \notin \mathsf{fn}(P')$, $\Gamma, x : T \vdash a : T'$, $\Gamma, x : T \vdash b : T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$, and $\Gamma, x : T; \Delta; \Psi \vdash P'$. Because $x \notin \mathsf{fn}(P)$, we have $a \neq x \neq b$. Thus, by Lemma 6.2.9 and Lemma 6.2.10, also $\Gamma \vdash a{:}T'$ and $\Gamma \vdash b{:}T'$. By the induction hypothesis, we have $\Gamma; \Delta; \Psi \vdash P'$. We conclude with T-MAT$_{\mathrm{M}}$.

Case of T-TAU$_{\mathrm{M}}$ **:** In this case, $P = \tau.P'$ for some $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $x \notin \mathsf{fn}(P')$, $\Delta = \Psi = \emptyset$, and $\Gamma, x : T; \Delta; \Psi \vdash P'$. By the induction hypothesis, we have $\Gamma; \Delta; \Psi \vdash P'$. We conclude with T-TAU$_{\mathrm{M}}$.

Case of T-Out-B$_{\mathrm{M}}$ **:** In this case, $P = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma, x{:}T \vdash y{:}\sharp(T')$, and $\Gamma, x{:}T \vdash z{:}T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$. Since $x \notin \mathsf{fn}(P)$, we have $y \neq x \neq z$. Thus, by Lemma 6.2.9 and Lemma 6.2.10, also $\Gamma \vdash y{:}\sharp(T')$ and $\Gamma \vdash z{:}T'$. We conclude with T-Out-B$_{\mathrm{M}}$.

Case of T-Out-M$_{\mathrm{M}}$ **:** Here $P = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$, $y \neq x \neq z$, $\Delta = \emptyset$, $\Gamma, x{:}T \vdash z{:}T'$, and $\Gamma \vdash z{:}T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$ but $\Psi = y{:}\sharp(T')$. Hence, again we conclude with T-Out-M$_{\mathrm{M}}$.

Case of T-OutPS$_{\mathrm{M}}$ **:** In this case, $P = \overline{y \cdot o}\langle z\rangle$ for some $o, y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma, x{:}T \vdash y{:}\mathfrak{v}_{\mathfrak{n}}$, $\Gamma, x{:}T \vdash o{:}\sharp(T')$, and $\Gamma, x{:}T \vdash z{:}T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$. Since $x \notin \mathsf{fn}(P)$, we have $x \notin \{ o, y, z \}$. Thus, by Lemma 6.2.9 and Lemma 6.2.10, also $\Gamma \vdash y{:}\mathfrak{v}_{\mathfrak{n}}$, $\Gamma \vdash o{:}\sharp(T')$, and $\Gamma \vdash z{:}T'$. We conclude with T-OutPS$_{\mathrm{M}}$.

Case of T-In-B$_{\mathrm{M}}$ **:** In this case, $P = y(x')\,.P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Delta = \emptyset$, $\Gamma, x : T \vdash y : \sharp(T')$, and $\Gamma, x : T, x' : T'; \Delta; \Psi \vdash P'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$ such that $T' \neq \circ \triangleright T''$ for all T''. Without loss of generality, let us assume that $x' \neq x$. Hence, $x \notin \mathsf{fn}(P')$. Moreover, since $x \notin \mathsf{fn}(P)$, we have $y \neq x$. Thus, by Lemma 6.2.9 and Lemma 6.2.10, also $\Gamma \vdash y{:}\sharp(T')$. By the induction hypothesis, we have $\Gamma, x'{:}T'; \Delta; \Psi \vdash P'$. We conclude with T-In-B$_{\mathrm{M}}$.

Case of T-In-M1$_{\mathrm{M}}$ **:** Again $P = y(x')\,.P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $x' \neq x \neq y$, $\Delta = \emptyset$, $\Gamma, x : T \vdash y : \sharp(T')$, and $\Gamma \vdash y : \sharp(T')$ for some $T' \in \mathbb{T}_{\mathrm{B}}$ but here $\Psi = \emptyset$, $T' = \circ \triangleright T''$ for some T'', and $\Gamma, x : T, x' : T'; \Delta; x' : T'' \vdash P'$. By the induction hypothesis, we have $\Gamma, x'{:}T'; \Delta; x' : T'' \vdash P'$. We conclude with T-In-M1$_{\mathrm{M}}$.

Case of T-In-M2$_{\mathrm{M}}$ **:** Here $P = y(x')\,.P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, and $x' \neq x \neq y$, but $\Delta = y : \sharp(T') \triangleright T''$, $\Psi = \emptyset$, and $\Gamma, x : T, x' : T'; \Delta'; \Psi \vdash P'$ for some T', T'' and either $T'' = \bullet$ and $\Delta' = \emptyset$ or $T'' \neq \bullet$ and $\Delta' = y : T''$. By the induction hypothesis, we have $\Gamma, x'{:}T'; \Delta'; \Psi \vdash P'$. We conclude with T-In-M2$_{\mathrm{M}}$.

Case of T-InPS$_{\mathrm{M}}$ **:** In this case, $P = y \cdot o(x')\,.P'$ for some $o, x', y \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Delta = \Psi = \emptyset$, $\Gamma, x : T \vdash y : \mathfrak{v}_{\mathfrak{n}}$, $\Gamma, x : T \vdash o : \sharp(T')$, and $\Gamma, x : T, x' : T'; \Delta; x' : T'' \vdash P'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$ such that $T' = \circ \triangleright T''$ for some T''. Without loss of generality, let us assume that $x' \neq x$. Hence, $x \notin \mathsf{fn}(P')$. Moreover, since $x \notin \mathsf{fn}(P)$, we have $y \neq x$. Thus, by Lemma 6.2.9 and Lemma 6.2.10, also $\Gamma \vdash y : \mathfrak{v}_{\mathfrak{n}}$ and $\Gamma \vdash o{:}\sharp(T')$. By the induction hypothesis, we have $\Gamma, x'{:}T'; \Delta; x' : T'' \vdash P'$. We conclude with T-InPS$_{\mathrm{M}}$.

Case of T-Rep-B$_{\mathrm{M}}$ **:** Similar to the case of T-In-B$_{\mathrm{M}}$.

Case of T-Rep-M$_{\mathrm{M}}$ **:** Similar to the case of T-In-M1$_{\mathrm{M}}$.

□

The proof of weakening is similar.

Proof of Lemma 6.2.30. Since type environments are sets, if $\Gamma(x) = T$ then $\Gamma, x{:}T = \Gamma$ and the lemma holds trivially.

For the other case, we perform an induction on the depth of the derivation. Let $\mathcal{P} \in \{ \mathcal{P}_{\mathrm{a}}, \mathcal{P}_{\mathrm{p}}, \mathcal{P}_{\mathrm{a}}^{=} \}$ be the set of processes of the target language of the considered encoding.

Base Case: If $\Gamma; \Delta; \Psi \vdash P$ can be derived from one of the axioms then either $P = \mathsf{0}$ or $P = \checkmark$ and $\Delta = \Psi = \emptyset$. In both cases $\Gamma, x{:}T; \Delta; \Psi \vdash P$ follows again directly by T-Nil$_{\mathrm{M}}$ or T-Succ$_{\mathrm{M}}$.

Induction Hypothesis: $\forall x \in \mathcal{N} \,.\, \forall T \in \mathbb{T}_{\mathrm{M}} \,.\, \forall P \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}}) \,.\, \Gamma; \Delta; \Psi \vdash P \wedge x \notin \mathsf{n}(\Gamma)$ imply $\Gamma, x{:}T; \Delta; \Psi \vdash P$

Induction Step: We perform a case split on the inference rules in Figure 6.8. As for Lemma 6.2.11, each case is simple. Nonetheless, we present them in full detail, to show the influence of the additional sets in judgements.

Case of T-Res-B$_{\mathrm{M}}$ **:** In this case, $P = (\nu x'{:}T')\, P'$ for some $x' \in \mathcal{N}$, $T' \in \mathbb{T}_{\mathrm{M}}$, $T' \neq \circ\triangleright T''$ for all T'', $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Gamma, x'{:}T'; \Delta; \Psi \vdash P'$. Without loss of generality, let us assume that $x' \neq x$, else apply α-conversion first. By assumption, $x \notin \mathsf{n}(\Gamma)$. Thus, by the induction hypothesis, $\Gamma, x'{:}T', x{:}T; \Delta; \Psi \vdash P'$. Then $\Gamma, x{:}T; \Delta; \Psi \vdash P$ follows from T-Res-B$_{\mathrm{M}}$.

Case of T-Res-M$_{\mathrm{M}}$ **:** Again $P = (\nu x'{:}T')\, P'$ for some $x' \in \mathcal{N}$, $x' \neq x$, $T' \in \mathbb{T}_{\mathrm{M}}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, and $x \notin \mathsf{n}(\Gamma)$ but here $T' = T_1 \triangleright T_2$ for some T_1, T_2, and $\Gamma, x'{:}T'; \Delta, \Delta'; \Psi \vdash P'$, where Δ' is either $x'{:}T_2$ or $\emptyset$. Thus, by the induction hypothesis, we have $\Gamma, x' : T', x : T; \Delta, \Delta'; \Psi \vdash P'$, and conclude again by T-Res-M$_{\mathrm{M}}$.

Case of T-Par$_{\mathrm{M}}$ **:** In this case, $P = P_1 \mid P_2$ for some $P_1, P_2 \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Delta = \Delta_1, \Delta_2$, $\Psi = \Psi_1 \cdot \Psi_2$, $\Gamma; \Delta_1; \Psi_1 \vdash P_1$, and $\Gamma; \Delta_2; \Psi_2 \vdash P_2$. Thus, by the induction hypothesis, $\Gamma, x : T; \Delta_1; \Psi_1 \vdash P_1$ and $\Gamma, x : T; \Delta_2; \Psi_2 \vdash P_2$. We conclude by T-Par$_{\mathrm{M}}$.

Case of T-Mat$_{\mathrm{M}}$ **:** In this case, $P = [a = b]\, P'$ for some $a, b \in \mathcal{N}$, $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $\Gamma \vdash a : T'$, $\Gamma \vdash b : T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$, and $\Gamma; \Delta; \Psi \vdash P'$. Because of Lemma 6.2.9, $\Gamma \vdash a : T'$, and $\Gamma \vdash b : T'$, we have $\Gamma(a) = T' = \Gamma(b)$ and, so, $a \neq x \neq b$. Thus, by Lemma 6.2.11, also $\Gamma, x{:}T \vdash a{:}T'$ and $\Gamma, x{:}T \vdash b{:}T'$. By the induction hypothesis, we have $\Gamma, x{:}T; \Delta; \Psi \vdash P'$. We conclude with T-Mat$_{\mathrm{M}}$.

Case of T-Tau$_{\mathrm{M}}$ **:** In this case, $P = \tau.P'$ for some $P' \in (\mathcal{P}{:}\mathbb{T}_{\mathrm{M}})$, $x \notin \mathsf{fn}(P')$, $\Delta = \Psi = \emptyset$, and $\Gamma; \Delta; \Psi \vdash P'$. By the induction hypothesis, $\Gamma, x : T; \Delta; \Psi \vdash P'$. We conclude with T-Tau$_{\mathrm{M}}$.

Case of T-Out-B$_{\mathrm{M}}$ **:** In this case, $P = \overline{y}\langle z \rangle$ for some $y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma \vdash y : \sharp(T')$, and $\Gamma \vdash z : T'$ for some $T' \in \mathbb{T}_{\mathrm{M}}$. Because of Lemma 6.2.9, $x \notin \mathsf{n}(\Gamma)$, $\Gamma \vdash y : \sharp(T')$, and $\Gamma \vdash z : T'$, we have $y \neq x \neq z$. Thus, by Lemma 6.2.11, also $\Gamma, x{:}T \vdash y{:}\sharp(T')$ and $\Gamma, x{:}T \vdash z{:}T'$. We conclude with T-Out-B$_{\mathrm{M}}$.

Case of T-Out-M$_\mathrm{M}$ **:** Here $P = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$, $\Delta = \emptyset$, $\Gamma \vdash z : T'$, $x \neq z$, and $\Gamma, x : T \vdash z : T'$ for some $T' \in \mathbb{T}_\mathrm{M}$ but $\Psi = y : \sharp(T')$. Hence, again we conclude with T-Out-M$_\mathrm{M}$.

Case of T-OutPS$_\mathrm{M}$ **:** In this case, $P = \overline{y \cdot o}\langle z\rangle$ for some $o, y, z \in \mathcal{N}$, $\Delta = \Psi = \emptyset$, $\Gamma \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma \vdash o : \sharp(T')$, and $\Gamma \vdash z : T'$ for some $T' \in \mathbb{T}_\mathrm{M}$. Since $x \notin \mathsf{fn}(P)$, we have $x \notin \{ o, y, z \}$. Thus, by Lemma 6.2.9 and Lemma 6.2.11, also $\Gamma, x : T \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma, x : T \vdash o : \sharp(T')$, and $\Gamma, x : T \vdash z : T'$. We conclude with T-OutPS$_\mathrm{M}$.

Case of T-In-B$_\mathrm{M}$ **:** In this case, $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_\mathrm{M})$, $\Delta = \emptyset$, $\Gamma \vdash y : \sharp(T')$, and $\Gamma, x' : T'; \Delta; \Psi \vdash P'$ for some $T' \in \mathbb{T}_\mathrm{M}$ such that $T' \neq \circ \triangleright T''$ for all T''. Without loss of generality, let us assume that $x' \neq x$. Moreover, because of Lemma 6.2.9, $x \notin \mathsf{n}(\Gamma)$, and $\Gamma \vdash y : \sharp(T')$, we have $y \neq x$. Thus, by Lemma 6.2.11, also $\Gamma, x : T \vdash y : \sharp(T')$. By the induction hypothesis, we have $\Gamma, x' : T', x : T; \Delta; \Psi \vdash P'$. We conclude with T-In-B$_\mathrm{M}$.

Case of T-In-M1$_\mathrm{M}$ **:** Again $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_\mathrm{M})$, $x' \neq x \neq y$, $\Delta = \emptyset$, $\Gamma \vdash y : \sharp(T')$, and $\Gamma, x : T \vdash y : \sharp(T')$ for some $T' \in \mathbb{T}_\mathrm{B}$ but here $\Psi = \emptyset$, $T' = \circ \triangleright T''$ for some T'', and $\Gamma, x' : T'; \Delta; x' : T'' \vdash P'$. By the induction hypothesis, we have $\Gamma, x' : T', x : T; \Delta; x' : T'' \vdash P'$. We conclude with T-In-M1$_\mathrm{M}$.

Case of T-In-M2$_\mathrm{M}$ **:** Here $P = y(x').P'$ for some $x', y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_\mathrm{M})$, and $x' \neq x \neq y$ but $\Delta = y : \sharp(T') \triangleright T''$, $\Psi = \emptyset$, and $\Gamma, x' : T'; \Delta'; \Psi \vdash P'$ for some T', T'' and either $T'' = \bullet$ and $\Delta' = \emptyset$ or $T'' \neq \bullet$ and $\Delta' = y : T''$. By the induction hypothesis, we have $\Gamma, x' : T', x : T; \Delta'; \Psi \vdash P'$. We conclude with T-In-M2$_\mathrm{M}$.

Case of T-InPS$_\mathrm{M}$ **:** In this case, $P = y \cdot o(x').P'$ for some $o, x', y \in \mathcal{N}$, $P' \in (\mathcal{P} : \mathbb{T}_\mathrm{M})$, $\Delta = \Psi = \emptyset$, $\Gamma \vdash y : \mathfrak{v}_\mathfrak{n}$, $\Gamma \vdash o : \sharp(T')$, and $\Gamma, x' : T'; \Delta; x' : T'' \vdash P'$ for some $T' \in \mathbb{T}_\mathrm{M}$ such that $T' = \circ \triangleright T''$ for some T''. Without loss of generality, let us assume that $x' \neq x$. Hence, $x \notin \mathsf{fn}(P')$. Moreover, since $x \notin \mathsf{fn}(P)$, we have $y \neq x$. Thus, by Lemma 6.2.9 and Lemma 6.2.11, also $\Gamma, x : T \vdash y : \mathfrak{v}_\mathfrak{n}$ and $\Gamma, x : T \vdash o : \sharp(T')$. By the induction hypothesis, we have $\Gamma, x' : T', x : T; \Delta; x' : T'' \vdash P'$. We conclude with T-InPS$_\mathrm{M}$.

Case of T-Rep-B$_\mathrm{M}$ **:** Similar to the case of T-In-B$_\mathrm{M}$.

Case of T-Rep-M$_\mathrm{M}$ **:** Similar to the case of T-In-M1$_\mathrm{M}$.

□

Lemma 6.2.31 states that if $\Gamma; \Delta; \Psi \vdash P$, $P \equiv Q$, and Γ is closed for Q then $\Gamma; \Delta; \Psi \vdash Q$.

Proof of Lemma 6.2.31. We show the condition for a single application of the rules of structural congruence in Figure 2.1 at Page 18. The lemma then follows by induction on the depth of the derivation of $P \equiv Q$. As described above, we equate typed processes that are equivalent modulo α-conversion.

Case of $P' \mid \mathsf{0} \equiv P'$**:** In this case, either $P = P' \mid \mathsf{0}$ and $Q = P'$ or $P = P'$ and $Q = P' \mid \mathsf{0}$. In the first case, the derivation of $\Gamma; \Delta; \Psi \vdash P' \mid \mathsf{0}$ starts with T-PAR$_\text{M}$ which leads to the subgoals $\Gamma; \Delta_1; \Psi_1 \vdash P'$ and $\Gamma; \Delta_2; \Psi_2 \vdash \mathsf{0}$ such that $\Delta = \Delta_1, \Delta_2$ and $\Psi = \Psi_1 \cdot \Psi_2$. $\Gamma; \Delta_2; \Psi_2 \vdash \mathsf{0}$ can only be proved by T-NIL$_\text{M}$ which requires $\Delta_2 = \Psi_2 = \emptyset$. Hence, $\Delta_1 = \Delta$ and, by Definition 6.2.27, $\Psi_1 = \Psi$. Hence, $\Gamma; \Delta; \Psi \vdash Q$ follows from $\Gamma; \Delta_1; \Psi_1 \vdash P'$.

In the other case, $P = P'$, i.e., we have $\Gamma; \Delta; \Psi \vdash P'$. Then $\Gamma; \Delta; \Psi \vdash Q$ with $Q = P' \mid \mathsf{0}$ holds, because

$$\dfrac{\dfrac{}{\Gamma; \Delta; \Psi \vdash P'}\text{by assumption} \quad \dfrac{}{\Gamma; \emptyset; \emptyset \vdash \mathsf{0}}\text{T-NIL}_\text{M}}{\Gamma; \Delta; \Psi \vdash P' \mid \mathsf{0}}\text{T-PAR}_\text{M}$$

Case of $P' \mid Q' \equiv Q' \mid P'$**:** This case follows by the symmetric definition of T-PAR$_\text{M}$, i.e., we can always exchange the left and the right hand side in parallel compositions. Note that also the preconditions $\Delta_P, \Delta_Q = \Delta_P \cup \Delta_Q$ and $\Psi_P \cdot \Psi_Q$ in Definition 6.2.27 are defined such that they always hold also in the symmetric case.

Case of $P' \mid (Q' \mid R) \equiv (P' \mid Q') \mid R$**:** In this case, either $P = P' \mid (Q' \mid R)$ and $Q = (P' \mid Q') \mid R$ or $P = (P' \mid Q') \mid R$ and $Q = P' \mid (Q' \mid R)$. In the first case the derivation of $\Gamma; \Delta; \Psi \vdash P$ starts with

$$\dfrac{\dfrac{}{\Gamma; \Delta_1; \Psi_1 \vdash P'} \cdots \quad \dfrac{\dfrac{}{\Gamma; \Delta_{2,1}; \Psi_{2,1} \vdash Q'} \cdots \quad \dfrac{}{\Gamma; \Delta_{2,2}; \Psi_{2,2} \vdash R} \cdots}{\Gamma; \Delta_2; \Psi_2 \vdash Q' \mid R}\text{T-PAR}_\text{M}}{\Gamma; \Delta; \Psi \vdash P' \mid (Q' \mid R)}\text{T-PAR}_\text{M}$$

such that $\Delta_2 = \Delta_{2,1}, \Delta_{2,2}$, $\Psi_2 = \Psi_{2,1} \cdot \Psi_{2,2}$, $\Delta = \Delta_1, (\Delta_{2,1}, \Delta_{2,2})$, and $\Psi = \Psi_1 \cdot (\Psi_{2,1} \cdot \Psi_{2,2})$. Obviously, $\Delta_1, (\Delta_{2,1}, \Delta_{2,2}) = (\Delta_1, \Delta_{2,1}), \Delta_{2,2}$. Moreover, by Definition 6.2.27, also $\Psi_1 \cdot (\Psi_{2,1} \cdot \Psi_{2,2}) = \Psi = (\Psi_1 \cdot \Psi_{2,1}) \cdot \Psi_{2,2}$. Hence, $\Gamma; \Delta; \Psi \vdash Q$ can be shown by the derivation

$$\dfrac{\dfrac{\dfrac{}{\Gamma; \Delta_1; \Psi_1 \vdash P'}A \quad \dfrac{}{\Gamma; \Delta_{2,1}; \Psi_{2,1} \vdash Q'}A}{\Gamma; \Delta_1, \Delta_{2,1}; \Psi_1 \cdot \Psi_{2,1} \vdash P' \mid Q'}\text{T-PAR}_\text{M} \quad \dfrac{}{\Gamma; \Delta_{2,2}; \Psi_{2,2} \vdash R}A}{\Gamma; \Delta; \Psi \vdash (P' \mid Q') \mid R}\text{T-PAR}_\text{M}$$

where A = by assumption, i.e., the remaining subgoals hold by the derivation above. The other case is similar.

Case of $[a = a]\, P' \equiv P'$**:** In this case either $P = [a = a]\, P'$ and $Q = P'$ or $P = P'$ and $Q = [a = a]\, P'$. In the first case, $\Gamma; \Delta; \Psi \vdash Q$ follows from $\Gamma; \Delta; \Psi \vdash [a = a]\, Q$ and T-MAT$_\text{M}$. In the second case, i.e., if $Q = [a = a]\, P'$, we apply T-MAT$_\text{M}$ which results in three subgoals. The first two are both $\Gamma \vdash a : T$ for some arbitrary type T. By assumption, there is some T' such that $\Gamma\,(a) = T'$. Thus, we choose $T = T'$. Then $\Gamma \vdash a : T$ follows by Lemma 6.2.9. The last subgoal $\Gamma; \Delta; \Psi \vdash P'$ holds by assumption.

Case of $(\nu n{:}T)\,\mathsf{0} \equiv \mathsf{0}$**:** If $P = (\nu n{:}T)\,\mathsf{0}$ and $Q = \mathsf{0}$ then the derivation of $\Gamma;\Delta;\Psi \vdash (\nu n{:}T)\,\mathsf{0}$ starts with T-Res-B$_\mathrm{M}$ followed by T-Nil$_\mathrm{M}$, because the latter requires $\Delta = \Psi = \emptyset$. Hence, $\Gamma;\emptyset;\emptyset \vdash \mathsf{0}$ follows from T-Nil$_\mathrm{M}$. Else, if $P = \mathsf{0}$ and $Q = (\nu n{:}T)\,\mathsf{0}$, $\Gamma;\Delta;\Psi \vdash \mathsf{0}$ can only be shown by T-Nil$_\mathrm{M}$ if $\Delta = \Psi = \emptyset$. Hence, $\Gamma;\emptyset;\emptyset \vdash (\nu n{:}T)\,\mathsf{0}$ follows from T-Res-B$_\mathrm{M}$ and then T-Nil$_\mathrm{M}$.

Case of $(\nu n{:}T_n)\,(\nu m{:}T_m)\,P' \equiv (\nu m{:}T_m)\,(\nu n{:}T_n)\,P'$**:** If $P = (\nu n{:}T_n)\,(\nu m{:}T_m)\,P'$ and $Q = (\nu m{:}T_m)\,(\nu n{:}T_n)\,P'$, the derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with T-Res-B$_\mathrm{M}$ or T-Res-M$_\mathrm{M}$ followed by another application of one of these two rules. (1) Applying two times T-Res-B$_\mathrm{M}$ leads to $\Gamma, n : T_n, m : T_m;\Delta;\Psi \vdash P'$, (2) applying first T-Res-B$_\mathrm{M}$ and then T-Res-M$_\mathrm{M}$ leads to $\Gamma, n : T_n, m : T_m;\Delta,\Delta_2;\Psi \vdash P'$, where Δ_2 is either $m : T'_m$ or $\emptyset$, (3) applying first T-Res-M$_\mathrm{M}$ and then T-Res-B$_\mathrm{M}$ leads to $\Gamma, n : T_n, m : T_m;\Delta,\Delta_3;\Psi \vdash P'$, where Δ_3 is either $n : T'_n$ or $\emptyset$, and (4) applying two times T-Res-M$_\mathrm{M}$ leads to $\Gamma, n{:}T_n, m{:}T_m;\Delta,\Delta_4;\Psi \vdash P'$, where $\Delta_4 \subseteq n{:}T'_n, m{:}T'_m$. Accordingly, $\Gamma;\Delta;\Psi \vdash Q$ can be shown by applying (1) two times T-Res-B$_\mathrm{M}$, (2) first T-Res-M$_\mathrm{M}$ and then T-Res-B$_\mathrm{M}$, (3) first T-Res-B$_\mathrm{M}$ and then T-Res-M$_\mathrm{M}$, or (4) two times T-Res-M$_\mathrm{M}$ and the assumption.

Case of $P' \mid (\nu n{:}T)\,Q' \equiv (\nu n{:}T)\,(P' \mid Q')$ **with** $n \notin \mathsf{fn}(P')$**:** In this case, we have $P = P' \mid (\nu n{:}T)\,Q'$ and $Q = (\nu n{:}T)\,(P' \mid Q')$ or $P = (\nu n{:}T)\,(P' \mid Q')$ and $Q = P' \mid (\nu n{:}T)\,Q'$. In the first case the derivation of $\Gamma;\Delta;\Psi \vdash P$ starts with

$$\dfrac{\dfrac{\dots}{\Gamma;\Delta_1;\Psi_1 \vdash P'} \cdots \quad \dfrac{\dfrac{\dots}{\Gamma, n{:}T;\Delta'_2;\Psi_2 \vdash Q'} \cdots}{\Gamma;\Delta_2;\Psi_2 \vdash (\nu n{:}T)\,Q'}R}{\Gamma;\Delta;\Psi \vdash P' \mid (\nu n{:}T)\,Q'}\text{T-Par}_\mathrm{M}$$

where R is either T-Res-B$_\mathrm{M}$ or T-Res-M$_\mathrm{M}$, $\Delta = \Delta_1,\Delta_2$, and $\Psi = \Psi_1 \cdot \Psi_2$. If $R =$ T-Res-B$_\mathrm{M}$ then $\Delta'_2 = \Delta_2$ and else, if $R =$ T-Res-M$_\mathrm{M}$, then $\Delta'_2 = \Delta_2, n{:}T'$ or $\Delta'_2 = \Delta_2$. In both cases, we have

$$\dfrac{\dfrac{\dfrac{}{\Gamma, n{:}T;\Delta_1;\Psi_1 \vdash P'}A \quad \dfrac{}{\Gamma, n{:}T;\Delta'_2;\Psi_2 \vdash Q'}\text{by assumption}}{\Gamma, n{:}T;\Delta';\Psi \vdash P' \mid Q'}\text{T-Par}_\mathrm{M}}{\Gamma;\Delta;\Psi \vdash (\nu n{:}T)\,(P' \mid Q')}R$$

where $A =$ (by Lemma 6.2.30 and assumption) and $\Delta' = \Delta$ if $R =$ T-Res-B$_\mathrm{M}$ and else $\Delta' = \Delta, n{:}T'$ or $\Delta' = \Delta$. The other case is similar.

□

A.1.3. Well-Typedness in the Linear Type System

Within this section we present the proofs of the Lemma 6.2.51, Lemma 6.2.52, and Lemma 6.2.53, i.e., we prove that the encodings $\mathcal{T}^1_\mathrm{L}[\![\cdot]\!]^\mathrm{s}_\mathrm{a}$, $\mathcal{T}^2_\mathrm{L}[\![\cdot]\!]^\mathrm{s}_\mathrm{a}$, and $\mathcal{T}^3_\mathrm{L}[\![\cdot]\!]^\mathrm{s}_\mathrm{a}$ are well-typed. Note that all three encodings use sum locks and type them by the same type. In fact, instantiations on sum locks are typed equivalently in the encoding functions.

Lemma A.1.2. *Positive and negative instantiations of a sum lock l that are typed by $\mathcal{T}_{\mathrm{L}}^1$, $\mathcal{T}_{\mathrm{L}}^2$, or $\mathcal{T}_{\mathrm{L}}^3$ are well-typed with respect to $l : \mathfrak{l}$ and $\Delta = \Psi = \emptyset$.*

Proof. By Figure 6.11, Figure 6.10, and Figure 6.9, sum locks, booleans, and the auxiliary values used in instantiations of sum locks are typed equivalently in $\mathcal{T}_{\mathrm{L}}^1$, $\mathcal{T}_{\mathrm{L}}^2$, and $\mathcal{T}_{\mathrm{L}}^3$. More precisely, for $\mathcal{T} \in \{ \mathcal{T}_{\mathrm{L}}^1, \mathcal{T}_{\mathrm{L}}^2, \mathcal{T}_{\mathrm{L}}^3 \}$, we have

$$
\begin{aligned}
&\mathcal{T}\big(\overline{l}\langle\top\rangle\big) \overset{\text{Def. 5.1.1}}{=} \mathcal{T}\big(l(t,f)\,.\,\overline{t}\big)\\
&\overset{\text{Def. 5.4.1}}{=} \mathcal{T}\big(l(u_{\sim,l})\,.\,\big((\nu u_t)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big((\nu u_f)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,(\nu v_t)\,\overline{t}\langle v_t\rangle\big)\big)\big)\big)\big)\\
&= l(u_{\sim,l})\,.\,\big(\,\big(\nu u_t : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))\big)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big(\,\big(\nu u_f : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))\big)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle\\
&\qquad\qquad \mid u_f(f)\,.\,(\nu v_t : \mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\,\big)\big)\big)\big)
\end{aligned}
$$

for positive and

$$
\begin{aligned}
&\mathcal{T}\big(\overline{l}\langle\bot\rangle\big) \overset{\text{Def. 5.1.1}}{=} \mathcal{T}\big(l(t,f)\,.\,\overline{f}\big)\\
&\overset{\text{Def. 5.4.1}}{=} \mathcal{T}\big(l(u_{\sim,l})\,.\,\big((\nu u_t)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big((\nu u_f)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid u_f(f)\,.\,(\nu v_f)\,\overline{f}\langle v_f\rangle\big)\big)\big)\big)\big)\\
&= l(u_{\sim,l})\,.\,\big(\,\big(\nu u_t : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))\big)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid u_t(t)\,.\,\big(\,\big(\nu u_f : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))\big)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle\\
&\qquad\qquad \mid u_f(f)\,.\,(\nu v_f : \mathfrak{v}_\bot)\,\overline{f}\langle v_f\rangle\,\big)\big)\big)\big)
\end{aligned}
$$

for negative instantiations of l. Remember that

$$\mathfrak{l} = \sharp(\mathfrak{l}_\circ) = \sharp\big(\circ \triangleright \sharp\big(\uparrow^1\big(\uparrow^+(\mathfrak{v}_\top)\big)\big) \triangleright \sharp\big(\uparrow^1\big(\uparrow^+(\mathfrak{v}_\bot)\big)\big) \triangleright \bullet\big)$$

for all three sets of type assignments. Let

$$
\begin{aligned}
\mathcal{T}\big(\overline{l}\langle b\rangle\big) &= l(u_{\sim,l})\,.\,\big(\big(\nu u_t : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))\big)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid T_{2,b}\big)\big)\\
T_{2,b} &= u_t(t)\,.\,\big(\big(\nu u_f : \updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))\big)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid T_{3,b}\big)\big)\\
T_{3,\top} &= u_f(f)\,.\,(\nu v_t : \mathfrak{v}_\top)\,\overline{t}\langle v_t\rangle\\
T_{3,\bot} &= u_f(f)\,.\,(\nu v_f : \mathfrak{v}_\bot)\,\overline{f}\langle v_f\rangle
\end{aligned}
$$

$T_1 = \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\big)$, and $T_2 = \sharp\big(\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\big)$.

$$
\dfrac{D_{1,1}\quad \dfrac{\dfrac{\dfrac{D_{1,2}}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, u_t:\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\,;\,\emptyset;\, u_{\sim,l}:T_1 \vdash \overline{u_{\sim,l}}\langle u_t\rangle}\,O \quad D_2}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, u_t:\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))\,;\,\emptyset;\, u_{\sim,l}:T_1 \triangleright T_2 \triangleright \bullet \vdash \overline{u_{\sim,l}}\langle u_t\rangle \mid T_{2,b}}\,\text{T-Par}_{\mathrm{L}}}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ;\emptyset;\, u_{\sim,l}:T_1 \triangleright T_2 \triangleright \bullet \vdash \big(\nu u_t:\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\top))\big)\,\big(\overline{u_{\sim,l}}\langle u_t\rangle \mid T_{2,b}\big)}\,R}{l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}\big(\overline{l}\langle b\rangle\big)}\,I
$$

where $O = \text{T-Out-M}_{\mathrm{L}}$, $R = \text{T-Res-B}_{\mathrm{L}}$, and $I = \text{T-In-M1}_{\mathrm{L}}$. $l:\mathfrak{l} \vdash l:\sharp_+(\mathfrak{l}_\circ)$ for $D_{1,1}$ and $l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, u_t:\uparrow^1(\uparrow^+(\mathfrak{v}_\top)) \vdash u_t:\uparrow^1(\uparrow^+(\mathfrak{v}_\top))$ for $D_{1,2}$ follow from $\text{T-Name}_{\mathrm{L}}$. $D_2 =$

$$
\dfrac{D_{2,1}\quad \dfrac{\dfrac{\dfrac{D_{2,2}}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, u_f:\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\,;\,\emptyset;\, u_{\sim,l}:T_2 \vdash \overline{u_{\sim,l}}\langle u_f\rangle}\,O \quad D_{3,b}}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, t:\uparrow^+(\mathfrak{v}_\top),\, u_f:\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))\,;\,\emptyset;\, u_{\sim,l}:T_2 \vdash \overline{u_{\sim,l}}\langle u_f\rangle \mid T_{3,b}}\,\text{T-Par}_{\mathrm{L}}}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, t:\uparrow^+(\mathfrak{v}_\top)\,;\,\emptyset;\, u_{\sim,l}:T_2 \vdash \big(\nu u_f:\updownarrow_1^1(\uparrow^+(\mathfrak{v}_\bot))\big)\,\big(\overline{u_{\sim,l}}\langle u_f\rangle \mid T_{3,b}\big)}\,R}{l:\mathfrak{l},\, u_{\sim,l}:\mathfrak{l}_\circ,\, u_t:\downarrow_1(\uparrow^+(\mathfrak{v}_\top))\,;\,\emptyset;\, u_{\sim,l}:T_2 \vdash T_{2,b}}\,I
$$

where $O = \text{T-Out-M}_\text{L}$, $R = \text{T-Res-B}_\text{L}$, and $I = \text{T-In-B}_\text{L}$. $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, u_t : \downarrow_1(\uparrow^+(\mathfrak{v}_\top)) \vdash u_t : \downarrow_1(\uparrow^+(\mathfrak{v}_\top))$ for $D_{2,1}$ and $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, u_f : \uparrow^1(\uparrow^+(\mathfrak{v}_\bot)) \vdash u_f : \uparrow^1(\uparrow^+(\mathfrak{v}_\bot))$ for $D_{2,2}$ follow from T-Name_L. Finally,

$$D_{3,\top} = \dfrac{D_{3,1} \quad \dfrac{\dfrac{D_{3,2} \quad D_{3,3}}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), v_t : \mathfrak{v}_\top ; \emptyset ; \emptyset \vdash \overline{t}\langle v_t \rangle} O}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot) ; \emptyset ; \emptyset \vdash (\nu v_t : \mathfrak{v}_\top)\, \overline{t}\langle v_t \rangle} \text{T-Res-B}_\text{L}}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), u_f : \downarrow_1(\uparrow^+(\mathfrak{v}_\bot)) ; \emptyset ; \emptyset \vdash T_{3,\top}} I$$

and

$$D_{3,\bot} = \dfrac{D_{3,1} \quad \dfrac{\dfrac{D_{3,4} \quad D_{3,5}}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), v_f : \mathfrak{v}_\bot ; \emptyset ; \emptyset \vdash \overline{f}\langle v_f \rangle} O}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot) ; \emptyset ; \emptyset \vdash (\nu v_f : \mathfrak{v}_\bot)\, \overline{f}\langle v_f \rangle} \text{T-Res-B}_\text{L}}{l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), u_f : \downarrow_1(\uparrow^+(\mathfrak{v}_\bot)) ; \emptyset ; \emptyset \vdash T_{3,\bot}} I$$

where $O = \text{T-Out-B}_\text{L}$ and $I = \text{T-In-B}_\text{L}$. $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, u_f : \downarrow_1(\uparrow^+(\mathfrak{v}_\bot)) \vdash u_f : \downarrow_1(\uparrow^+(\mathfrak{v}_\bot))$ for $D_{3,1}$, $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), v_t : \mathfrak{v}_\top \vdash t : \uparrow^+(\mathfrak{v}_\top)$ for $D_{3,2}$, $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, v_t : \mathfrak{v}_\top \vdash v_t : \mathfrak{v}_\top$ for $D_{3,3}$, $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), v_f : \mathfrak{v}_\bot \vdash f : \uparrow^+(\mathfrak{v}_\bot)$ for $D_{3,4}$, and $l : \mathfrak{l}, u_{\sim,l} : \mathfrak{l}_\circ, v_f : \mathfrak{v}_\bot \vdash v_f : \mathfrak{v}_\bot$ for $D_{3,5}$ follow from T-Name_L. □

Similarly, outputs on sum locks, i.e., test-constructs, are well-typed in the typed variants of all three encodings within the linear type system.

Lemma A.1.3. *All* test*-constructs* test l then P else Q *in the typed encodings* $\mathcal{T}^1_\text{L}[\![\cdot]\!]^\text{s}_\text{a}$, $\mathcal{T}^2_\text{L}[\![\cdot]\!]^\text{m}_\text{p}$, *or* $\mathcal{T}^3_\text{L}[\![\cdot]\!]^\text{m}_\text{a}$ *are well-typed with respect to* $\Delta = \Psi = \emptyset$ *and all* Γ *such that* $\Gamma ; \emptyset ; \emptyset \vdash P \mid Q$.

Proof. By Figure 6.11, Figure 6.10, and Figure 6.9, sum locks, booleans, and the auxiliary values used in outputs on sum locks are typed equivalently in $\mathcal{T}^1_\text{L}$, $\mathcal{T}^2_\text{L}$, and $\mathcal{T}^3_\text{L}$. More precisely, for $\mathcal{T} \in \{ \mathcal{T}^1_\text{L}, \mathcal{T}^2_\text{L}, \mathcal{T}^3_\text{L} \}$, we have

$$\begin{aligned}
&\mathcal{T}(\text{test } l \text{ then } P \text{ else } Q) \overset{\text{Def. 5.1.1}}{=} \mathcal{T}\big((\nu t, f)\, \big(\overline{l}\langle t, f \rangle \mid t.P \mid f.Q\big)\big) \\
&\overset{\text{Def. 5.4.1}}{=} \mathcal{T}\big((\nu t, f)\, \big((\nu u_{\sim,l})\, \big(\overline{l}\langle u_{\sim,l} \rangle \mid u_{\sim,l}(u_t).\, (\overline{u_t}\langle t \rangle \mid u_{\sim,l}(u_f).\, \overline{u_f}\langle f \rangle)\big) \mid t(v_t).P \\
&\qquad\qquad \mid f(v_f).Q\big)\big) \\
&= \big(\nu t : \updownarrow^+_1(\mathfrak{v}_\top), f : \updownarrow^+_1(\mathfrak{v}_\bot)\big)\, \big(\big(\nu u_{\sim,l} : \circ \triangleright \sharp\big(\uparrow^1\big(\uparrow^+(\mathfrak{v}_\top)\big)\big) \triangleright \sharp\big(\uparrow^1\big(\uparrow^+(\mathfrak{v}_\bot)\big)\big) \triangleright \bullet\big) \\
&\qquad\qquad \big(\overline{l}\langle u_{\sim,l} \rangle \mid u_{\sim,l}(u_t).\, (\overline{u_t}\langle t \rangle \mid u_{\sim,l}(u_f).\, \overline{u_f}\langle f \rangle)\big) \mid t(v_t).P \mid f(v_f).Q\big)
\end{aligned}$$

Moreover, we observe that, for all three encodings and all test-constructs, Γ contains at least $l : \mathfrak{l}$. By T-Par_L, $\Gamma ; \emptyset ; \emptyset \vdash P \mid Q$ implies that there exists two type environments Γ_P and Γ_Q such that

$$\Gamma_P ; \emptyset ; \emptyset \vdash P, \tag{A.1}$$
$$\Gamma_Q ; \emptyset ; \emptyset \vdash Q, \tag{A.2}$$

and $\Gamma = \Gamma_P + \Gamma_Q$. By Definition 6.2.42, $l : \mathfrak{l} \in \Gamma_P$ as well as $l : \mathfrak{l} \in \Gamma_Q$. Let

$$\begin{aligned}\mathcal{T}(\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q) &= \left(\nu t : \updownarrow_1^+(\mathfrak{v}_\top), f : \updownarrow_1^+(\mathfrak{v}_\bot)\right)(P_1 \mid t(v_t).P \mid f(v_f).Q)\\ P_1 &= (\nu u_{\sim,l} : \mathfrak{l}_\circ)\left(\overline{l}\langle u_{\sim,l}\rangle \mid P_2\right)\\ P_2 &= u_{\sim,l}(u_t).(\overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f).\overline{u_f}\langle f\rangle)\end{aligned}$$

$\mathfrak{l}_\circ = \circ \triangleright \sharp\left(\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\right) \triangleright \sharp\left(\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\right) \triangleright \bullet$, $T_1 = \sharp\left(\uparrow^1(\uparrow^+(\mathfrak{v}_\top))\right)$, and $T_2 = \sharp\left(\uparrow^1(\uparrow^+(\mathfrak{v}_\bot))\right)$. Moreover, let Γ' contain all type assignments of Γ that do not assign a linear type (inclusive $l : \mathfrak{l}$). Then, by Definition 6.2.42, $\Gamma = \Gamma' + \Gamma$. We use Γ' for D_1 in the following derivation.

$$\dfrac{\dfrac{D_1 \quad \dfrac{D_P \quad D_Q}{\Gamma, t : \downarrow_1(\mathfrak{v}_\top), f : \downarrow_1(\mathfrak{v}_\bot); \emptyset; \emptyset \vdash t(v_t).P \mid f(v_f).Q}\text{T-Par}_\text{L}}{\Gamma, t : \updownarrow_1^+(\mathfrak{v}_\top), f : \updownarrow_1^+(\mathfrak{v}_\bot); \emptyset; \emptyset \vdash P_1 \mid t(v_t).P \mid f(v_f).Q}\text{T-Par}_\text{L}}{\Gamma, t : \updownarrow_1^+(\mathfrak{v}_\top); \emptyset; \emptyset \vdash \left(\nu f : \updownarrow_1^+(\mathfrak{v}_\bot)\right)(P_1 \mid t(v_t).P \mid f(v_f).Q)}R}{\Gamma; \emptyset; \emptyset \vdash \mathcal{T}(\mathsf{test}\ l\ \mathsf{then}\ P\ \mathsf{else}\ Q)}R$$

where $R = \text{T-Res-B}_\text{L}$.

$$D_P = \dfrac{\dfrac{}{\Gamma_P, t : \downarrow_1(\mathfrak{v}_\top) \vdash t : \downarrow_1(\mathfrak{v}_\top)}N \quad \dfrac{}{\Gamma'_P; \emptyset; \emptyset \vdash P}(A.1)\text{ and Lemma 6.2.46}}{\Gamma_P, t : \downarrow_1(\mathfrak{v}_\top); \emptyset; \emptyset \vdash t(v_t).P}\text{T-In-B}_\text{L}$$

where $N = \text{T-Name}_\text{L}$ and $\Gamma'_P = \Gamma_P$ if $v_t \in \mathsf{n}(\Gamma_P)$ and else $\Gamma'_P = \Gamma_P, v_t : \mathfrak{v}_\top$. Note that the respective renaming policy ensures that for each encoding and each test-construct we always have $v_t \notin \mathsf{fn}(P)$. However, since $v_t : \mathfrak{v}_\top$ and $\mathfrak{v}_\top$ is a linear value type, the name v_t can never be used as link or be received on a channel that expects a link and so does no harm. Similarly,

$$D_Q = \dfrac{\dfrac{}{\Gamma_Q, f : \downarrow_1(\mathfrak{v}_\bot) \vdash f : \downarrow_1(\mathfrak{v}_\bot)}N \quad \dfrac{}{\Gamma'_Q; \emptyset; \emptyset \vdash Q}(A.2)\text{ and Lemma 6.2.46}}{\Gamma_Q, f : \downarrow_1(\mathfrak{v}_\bot); \emptyset; \emptyset \vdash f(v_f).Q}\text{T-In-B}_\text{L}$$

where $N = \text{T-Name}_\text{L}$ and $\Gamma'_Q = \Gamma_Q$ if $v_f \in \mathsf{n}(\Gamma_Q)$ and else $\Gamma'_Q = \Gamma_Q, v_f : \mathfrak{v}_\bot$.

$$D_1 = \dfrac{\dfrac{\dfrac{}{\Gamma', u_{\sim,l} : \mathfrak{l}_\circ; \emptyset; \emptyset \vdash \overline{l}\langle u_{\sim,l}\rangle}\text{T-Out-B}_\text{L} \quad D_2}{\Gamma', t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), u_{\sim,l} : \mathfrak{l}_\circ; u_{\sim,l} : T_1 \triangleright T_2 \triangleright \bullet; \emptyset \vdash \overline{l}\langle u_{\sim,l}\rangle \mid P_2}\text{T-Par}_\text{L}}{\Gamma', t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot); \emptyset; \emptyset \vdash (\nu u_{\sim,l} : \mathfrak{l}_\circ)\left(\overline{l}\langle u_{\sim,l}\rangle \mid P_2\right)}R$$

where $R = \text{T-Res-M}_\text{L}$. $\Gamma', u_{\sim,l} : \mathfrak{l}_\circ \vdash l : \sharp(\mathfrak{l}_\circ)$ for $D_{1,1}$ and $\Gamma', u_{\sim,l} : \mathfrak{l}_\circ \vdash u_{\sim,l} : \mathfrak{l}_\circ$ for $D_{1,2}$ follow from T-Name_L. Let $\Gamma_2 = \Gamma', t : \uparrow^+(\mathfrak{v}_\top), f : \uparrow^+(\mathfrak{v}_\bot), u_{\sim,l} : \mathfrak{l}_\circ$. $D_2 =$

$$\dfrac{\dfrac{\dfrac{D_{2,1} \quad D_{2,2}}{\Gamma', t : \uparrow^+(\mathfrak{v}_\top), u_{\sim,l} : \mathfrak{l}_\circ, u_t : \uparrow^1(\uparrow^+(\mathfrak{v}_\top)); \emptyset; \emptyset \vdash \overline{u_t}\langle t\rangle}\text{T-Out-B}_\text{L} \quad D_3}{\Gamma_2, u_t : \uparrow^1(\uparrow^+(\mathfrak{v}_\top)); u_{\sim,l} : T_2 \triangleright \bullet; \emptyset \vdash \overline{u_t}\langle t\rangle \mid u_{\sim,l}(u_f).\overline{u_f}\langle f\rangle}\text{T-Par}_\text{L}}{\Gamma_2; u_{\sim,l} : T_1 \triangleright T_2 \triangleright \bullet; \emptyset \vdash P_2}\text{T-In-M2}_\text{L}$$

$\Gamma', u_{\sim,l} : \mathfrak{l}_\circ, u_t : \uparrow^1(\uparrow^+(\mathfrak{v}_\top)) \vdash u_t : \uparrow^1(\uparrow^+(\mathfrak{v}_\top))$ for $D_{2,1}$ and $\Gamma', t : \uparrow^+(\mathfrak{v}_\top), u_{\sim,l} : \mathfrak{l}_\circ \vdash t : \uparrow^+(\mathfrak{v}_\top)$ for $D_{2,2}$ follow from T-NAME$_\mathrm{L}$. Finally,

$$D_3 = \dfrac{\dfrac{D_{3,1} \quad D_{3,2}}{\Gamma', f : \uparrow^+(\mathfrak{v}_\bot), u_{\sim,l} : \mathfrak{l}_\circ, u_f : \uparrow^1(\uparrow^+(\mathfrak{v}_\bot)) ; \emptyset; \emptyset \vdash \overline{u_f}\langle f \rangle}\text{T-OUT-B}_\mathrm{L}}{\Gamma', f : \uparrow^+(\mathfrak{v}_\bot), u_{\sim,l} : \mathfrak{l}_\circ; u_{\sim,l} : T_2 \triangleright \bullet; \emptyset \vdash u_{\sim,l}(u_f) . \overline{u_f}\langle f \rangle}\text{T-IN-M2}_\mathrm{L}$$

$\Gamma', u_{\sim,l} : \mathfrak{l}_\circ, u_f : \uparrow^1(\uparrow^+(\mathfrak{v}_\bot)) \vdash u_f : \uparrow^1(\uparrow^+(\mathfrak{v}_\bot))$ for $D_{3,1}$ and $\Gamma', f : \uparrow^+(\mathfrak{v}_\bot), u_{\sim,l} : \mathfrak{l}_\circ \vdash f : \uparrow^+(\mathfrak{v}_\bot)$ for $D_{3,2}$ follow from T-NAME$_\mathrm{L}$. □

Moreover, remember that the unfolding of polyadic communication follows a strict schema given in Definition 6.2.24 and proved correct in Lemma 6.2.39. Also the augmentation with linear types follows a strict line within this unfolding as described by Figures 6.9, 6.10, and 6.11. Hence, all such unfoldings of polyadic communication are well-typed in the linear type system.

Lemma A.1.4. *All unfolded polyadic communications in the typed encodings $\mathcal{T}^1_\mathrm{L}[\![\cdot]\!]^\mathrm{s}_\mathrm{a}$, $\mathcal{T}^2_\mathrm{L}[\![\cdot]\!]^\mathrm{m}_\mathrm{p}$, and $\mathcal{T}^3_\mathrm{L}[\![\cdot]\!]^\mathrm{m}_\mathrm{a}$ are well-typed with respect to $\Delta = \Psi = \emptyset$ and some type environment Γ if the respective continuation is well-typed with respect to Γ' and $\Delta = \Psi = \emptyset$, where in case of outputs $\Gamma' = \Gamma$ and else Γ' is the union of Γ and a type assignment for each received value in the polyadic communication.*

Proof. Analysing the types in Figures 6.9, 6.10, and 6.11, we observe that all auxiliary links introduced by the unfoldings of polyadic communications (Definition 5.4.1) are typed by m-sorts or linear types. Since all these unfoldings follow the same schema and have similar types also the derivations of the respective type judgements are similar. With Lemma A.1.2 and Lemma A.1.3 we present two derivations of such judgements; one for inputs and one for outputs of polyadic communications of multiplicity two. The other derivations are similar. □

Lemma 6.2.53 states that the encoding $\mathcal{T}^3_\mathrm{L}[\![\cdot]\!]^\mathrm{m}_\mathrm{a}$ is well-typed with respect to $\Gamma_{[\![\cdot]\!]^\mathrm{m}_\mathrm{a}}$, where $\Gamma_{[\![\cdot]\!]^\mathrm{m}_\mathrm{a}} = \{\, p_o : \uparrow^\omega(\mathfrak{o}_\circ), p_i : \uparrow^\omega(\mathfrak{i}_\circ) \mid p_o, p_i \in \mathsf{fn}([\![S]\!]^\mathrm{m}_\mathrm{a}) \,\} \cup \{\, x : \mathfrak{v}_\mathfrak{n} \mid x \in \mathsf{fn}(S) \,\}$ for all source terms $S \in \mathcal{P}_\mathrm{m}$.

Proof of Lemma 6.2.53. An encoding is well-typed if each encoded term is well-typed. Hence, we perform an induction over the structure of source terms. Note that no source term can contain infinitely many free names. We conclude that the type environment is finite for each translation of a source term. Let $\Gamma = \Gamma_{[\![\cdot]\!]^\mathrm{m}_\mathrm{a}}$.

Base Case: In $\mathcal{P}_\mathrm{m}$ there are two terms without subterms, namely $\mathsf{0}$ and $\checkmark$. $\mathcal{T}^3_\mathrm{L}[\![\mathsf{0}]\!]^\mathrm{m}_\mathrm{a} = (\nu l : \mathfrak{l})\,(\mathcal{T}^3_\mathrm{L}(\overline{l}\langle\top\rangle))$ and $\mathcal{T}^3_\mathrm{L}[\![\checkmark]\!]^\mathrm{m}_\mathrm{a} = \checkmark$. The second case, i.e., the type judgement $\emptyset; \emptyset; \emptyset \vdash \checkmark$, follows directly from T-SUCC$_\mathrm{L}$. For the first case, we have

$$\dfrac{\dfrac{}{l : \mathfrak{l}; \emptyset; \emptyset \vdash \mathcal{T}^3_\mathrm{L}(\overline{l}\langle\top\rangle)}\text{Lemma A.1.2}}{\emptyset; \emptyset; \emptyset \vdash (\nu l : \mathfrak{l})\,(\mathcal{T}^3_\mathrm{L}(\overline{l}\langle\top\rangle))}\text{T-RES-B}_\mathrm{L}$$

Induction Hypothesis:

$$\forall S \in \mathcal{P}_{\mathrm{m}} \,.\, \Gamma;\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \tag{IH}$$

Induction Step: We have to prove that $\Gamma;\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ for the cases that S is a restricted term, a parallel composition, a sum, or a replicated input. The corresponding derivations are huge but very simple. For each step in each case there applies exactly one rule of Figure 6.12.

1. If $S = (\nu x)\, S'$ for some $S' \in \mathcal{P}_{\mathrm{m}}$. Note that $\mathsf{fn}(S) = \mathsf{fn}(S') \setminus \{\, x \,\}$. Thus, $\Gamma_{[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}} = \Gamma_{[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}}, \varphi^{\mathrm{m}}_{\mathrm{a}}(x) : \mathfrak{v}_{\mathfrak{n}}$ if $x \in \mathsf{fn}(S')$ and else $\Gamma_{[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}} = \Gamma_{[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}}$. Then:

$$\frac{\dfrac{}{\Gamma, \varphi^{\mathrm{m}}_{\mathrm{a}}(x) : \mathfrak{v}_{\mathfrak{n}};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}}(IH) \text{ and Lemma 6.2.46}}{\Gamma;\emptyset;\emptyset \vdash (\nu \varphi^{\mathrm{m}}_{\mathrm{a}}(x) : \mathfrak{v}_{\mathfrak{n}})\, \mathcal{T}^3_{\mathrm{L}}[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-Res-B}_{\mathrm{L}}$$

2. If $S = S_1 \mid S_2$ for some $S_1, S_2 \in \mathcal{P}_{\mathrm{m}}$ then $\mathcal{T}^3_{\mathrm{L}}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} =$

$$\begin{aligned}
&(\nu m_o, p_{o,up} : \mathfrak{o}_*, m_i, p_{i,up} : \mathfrak{i}_*, c_o : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{i}_\circ)), c_i : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{o}_\circ)))\ (\\
&\quad (\nu p_o : \mathfrak{o}_*, p_i : \mathfrak{i}_*)\ (\mathcal{T}^3_{\mathrm{L}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathcal{T}^3_{\mathrm{L}}(\mathsf{procLeftOutReq}) \mid \mathcal{T}^3_{\mathrm{L}}(\mathsf{procLeftInReq}))\\
&\quad \mid (\nu p_o : \mathfrak{o}, p_i : \mathfrak{i})\ (\mathcal{T}^3_{\mathrm{L}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathcal{T}^3_{\mathrm{L}}(\mathsf{procRightOutReq}) \mid \mathcal{T}^3_{\mathrm{L}}(\mathsf{procRightInReq}))\\
&\quad \mid \mathcal{T}^3_{\mathrm{L}}(\mathsf{pushReq})\ ).
\end{aligned}$$

By applying α-conversion and T-Res-B$_{\mathrm{L}}$ several times, $\Gamma;\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ becomes $\Gamma_1;\emptyset;\emptyset \vdash T_1 \mid T_2 \mid T_3$, where $\Gamma_1 = \Gamma, m_o, p_{o,up} : \mathfrak{o}_*, m_i, p_{i,up} : \mathfrak{i}_*, c_o : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{i}_\circ)), c_i : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{o}_\circ))$, $\sigma_1 = \{\, {}^{p_o'}/_{p_o}, {}^{p_i'}/_{p_i} \,\}$ and

$$\begin{aligned}
T_1 &= (\nu p_o' : \mathfrak{o}_*, p_i' : \mathfrak{i}_*)\ (\sigma_1 \mathcal{T}^3_{\mathrm{L}}[\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid T_4 \mid T_5)\\
T_2 &= (\nu p_o' : \mathfrak{o}, p_i' : \mathfrak{i})\ (\sigma_1 \mathcal{T}^3_{\mathrm{L}}[\![\, S_2 \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid T_6 \mid T_7)\\
T_3 &= \mathcal{T}^3_{\mathrm{L}}(\mathsf{pushReq}),\ T_4 = \sigma_1 \mathcal{T}^3_{\mathrm{L}}(\mathsf{procLeftOutReq}),\ T_5 = \sigma_1 \mathcal{T}^3_{\mathrm{L}}(\mathsf{procLeftInReq})\\
T_6 &= \sigma_1 \mathcal{T}^3_{\mathrm{L}}(\mathsf{procRightOutReq}),\ T_7 = \sigma_1 \mathcal{T}^3_{\mathrm{L}}(\mathsf{procRightInReq})
\end{aligned}$$

Let $\Gamma_{1,1} = \Gamma, m_o, p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), m_i, p_{i,up} : \uparrow^{\omega}(\mathfrak{i}_\circ)$, $\Gamma_{1,2} = \Gamma, m_o : \downarrow_*(\mathfrak{o}_\circ), p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), m_i : \downarrow_*(\mathfrak{i}_\circ), p_{i,up} : \uparrow^{\omega}(\mathfrak{i}_\circ), c_o : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{i}_\circ)), c_i : \updownarrow^{\omega}_*(\downarrow_*(\mathfrak{o}_\circ))$, and $\Gamma_{1,3} = \Gamma, p_{o,up} : \downarrow_*(\mathfrak{o}_\circ), p_{i,up} : \downarrow_*(\mathfrak{i}_\circ)$. Then

$$\frac{D_1 \qquad \dfrac{D_2 \quad D_3}{\Gamma_{1,2} + \Gamma_{1,3};\emptyset;\emptyset \vdash T_2 \mid T_3}\text{T-Par}_{\mathrm{L}}}{\Gamma_1;\emptyset;\emptyset \vdash T_1 \mid T_2 \mid T_3}\text{T-Par}_{\mathrm{L}}$$

Let $\Gamma_{2,1} = \Gamma_{1,1}, p_o' : \mathfrak{o}_*, p_i' : \mathfrak{i}_*$, $\Gamma_{2,1,1} = \Gamma, p_o' : \uparrow^{\omega}(\mathfrak{o}_\circ), p_i' : \uparrow^{\omega}(\mathfrak{i}_\circ)$, $\Gamma_{2,1,2} = \Gamma_{1,1}, p_o' : \downarrow_*(\mathfrak{o}_\circ), p_i' : \downarrow_*(\mathfrak{i}_\circ)$, $\Gamma_{2,2} = \Gamma_{1,2}, p_o' : \mathfrak{o}, p_i' : \mathfrak{i}$, $\Gamma_{2,2,1} = \Gamma, p_o' : \uparrow^{\omega}(\mathfrak{o}_\circ), p_i' : \uparrow^{\omega}(\mathfrak{i}_\circ)$, and $\Gamma_{2,2,2} = \Gamma_{1,2}, p_o' : \mathfrak{o}, p_i' : \mathfrak{i}$. By applying T-Res-B$_{\mathrm{L}}$ two times,

respectively, D_1 becomes $\Gamma_{2,1}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_1\ ]\!]_{\mathrm{a}}^{\mathrm{m}} \mid T_4 \mid T_5$ and D_2 becomes $\Gamma_{2,2}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_2\ ]\!]_{\mathrm{a}}^{\mathrm{m}} \mid T_6 \mid T_7$. Then

$$\dfrac{\dfrac{}{\Gamma_{2,1,1}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_1\ ]\!]_{\mathrm{a}}^{\mathrm{m}}}\text{(IH) and Lemma 6.2.46} \quad \dfrac{D_4 \quad D_5}{\Gamma_{2,1,2}; \emptyset; \emptyset \vdash T_4 \mid T_5} P}{\Gamma_{2,1}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_1\ ]\!]_{\mathrm{a}}^{\mathrm{m}} \mid T_4 \mid T_5} P$$

and

$$\dfrac{\dfrac{}{\Gamma_{2,2,1}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_2\ ]\!]_{\mathrm{a}}^{\mathrm{m}}}\text{(IH) and Lemma 6.2.46} \quad \dfrac{D_6 \quad D_7}{\Gamma_{2,2,2}; \emptyset; \emptyset \vdash T_6 \mid T_7} P}{\Gamma_{2,2}; \emptyset; \emptyset \vdash \sigma_1 \mathcal{T}_{\mathrm{L}}^3[\![\ S_2\ ]\!]_{\mathrm{a}}^{\mathrm{m}} \mid T_6 \mid T_7} P$$

where $P = \text{T-Par}_{\mathrm{L}}$. Let $\Gamma_3 = \Gamma_{1,1}, y, z : \mathfrak{v}_{\mathfrak{n}}, l : \mathfrak{l}, s : \uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}})$.

$$D_4 = \dfrac{D_{4,1} \quad \dfrac{D_{4,2} \quad D_{4,3}}{\Gamma_3; \emptyset; \emptyset \vdash \overline{m_o}\langle y, l, s, z\rangle \mid \overline{p_{o,up}}\langle y, l, s, z\rangle}\text{T-Par}_{\mathrm{L}}}{\Gamma_{1,1}, {p_o}' : \downarrow_*(\mathfrak{o}_\circ)\,; \emptyset; \emptyset \vdash {p_o}'^{\star}(y, l, s, z)\,.\,(\overline{m_o}\langle y, l, s, z\rangle \mid \overline{p_{o,up}}\langle y, l, s, z\rangle)} R$$

where $R = \text{T-Rep-B}_{\mathrm{L}}$ and Lemma A.1.4. $\Gamma_{1,1}, {p_o}' : \downarrow_*(\mathfrak{o}_\circ) \vdash {p_o}' : \downarrow_*(\mathfrak{o}_\circ)$ for $D_{4,1}$ follows from $\text{T-Name}_{\mathrm{L}}$. Apply $\text{T-Out-B}_{\mathrm{L}}$ and Lemma A.1.4, and then $\text{T-Name}_{\mathrm{L}}$ on all subgoals to show $\Gamma_3; \emptyset; \emptyset \vdash \overline{m_o}\langle y, l, s, z\rangle$ for $D_{4,2}$ and $\Gamma_3; \emptyset; \emptyset \vdash \overline{p_{o,up}}\langle y, l, s, z\rangle$ for $D_{4,3}$. Let $\Gamma_4 = \Gamma_{1,1}, y : \mathfrak{v}_{\mathfrak{n}}, l : \mathfrak{l}, r : \uparrow^{\omega}(\mathfrak{r}_\circ)$.

$$D_5 = \dfrac{D_{5,1} \quad \dfrac{D_{5,2} \quad D_{5,3}}{\Gamma_4; \emptyset; \emptyset \vdash \overline{m_i}\langle y, l, r\rangle \mid \overline{p_{i,up}}\langle y, l, r\rangle}\text{T-Par}_{\mathrm{L}}}{\Gamma_{1,1}, {p_i}' : \downarrow_*(\mathfrak{i}_\circ)\,; \emptyset; \emptyset \vdash {p_i}'^{\star}(y, l, r)\,.\,(\overline{m_i}\langle y, l, r\rangle \mid \overline{p_{i,up}}\langle y, l, r\rangle)} R$$

where $R = \text{T-Rep-B}_{\mathrm{L}}$ and Lemma A.1.4. $\Gamma_{1,1}, {p_i}' : \downarrow_*(\mathfrak{i}_\circ) \vdash {p_i}' : \downarrow_*(\mathfrak{i}_\circ)$ for $D_{5,1}$ follows from $\text{T-Name}_{\mathrm{L}}$. Apply $\text{T-Out-B}_{\mathrm{L}}$ and Lemma A.1.4, and then $\text{T-Name}_{\mathrm{L}}$ on all subgoals to show $\Gamma_4; \emptyset; \emptyset \vdash \overline{m_i}\langle y, l, r\rangle$ for $D_{5,2}$ and $\Gamma_4; \emptyset; \emptyset \vdash \overline{p_{i,up}}\langle y, l, r\rangle$ for $D_{5,3}$. Let

$$\begin{aligned}
T_6 &= \overline{c_o}\langle m_i\rangle \mid c_o^{\star}(m_i)\,.p_o'(y, l_s, s, z)\,.\,\left(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6'\right)\\
T_6' &= (\nu m_{i,up} : \mathfrak{i}_*)\left(T_6'' \mid (\nu m_i : \mathfrak{i}_*)\left(m_{i,up}{}^{\star}(y', l_r, r)\,.\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle\right)\right)\\
T_6'' &= m_i{}^{\star}(y', l_r, r)\,.\left([y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle\right)
\end{aligned}$$

Let $\Gamma_{2,2,2,1} = \Gamma, p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), m_i : \downarrow_*(\mathfrak{i}_\circ), c_o : \updownarrow_*^{\omega}(\downarrow_*(\mathfrak{i}_\circ)), {p_o}' : \mathfrak{o}$, $\Gamma_{2,2,2,2} = \Gamma, m_o : \downarrow_*(\mathfrak{o}_\circ), p_{i,up} : \uparrow^{\omega}(\mathfrak{i}_\circ), c_i : \updownarrow_*^{\omega}(\downarrow_*(\mathfrak{o}_\circ)), {p_i}' : \mathfrak{i}$, $\Gamma_{2,2,2,1,2} = \Gamma, p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), c_o : \updownarrow_*^{\omega}(\downarrow_*(\mathfrak{i}_\circ)), {p_o}' : \mathfrak{o}$, $\Gamma'_{2,2,2,1,2} = \Gamma, p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), c_o : \uparrow^{\omega}(\downarrow_*(\mathfrak{i}_\circ)), {p_o}' : \mathfrak{o}, m_i : \downarrow_*(\mathfrak{i}_\circ)$, and $\Gamma_5 = \Gamma, p_{o,up} : \uparrow^{\omega}(\mathfrak{o}_\circ), c_o : \uparrow^{\omega}(\downarrow_*(\mathfrak{i}_\circ)), m_i : \downarrow_*(\mathfrak{i}_\circ), y, z : \mathfrak{v}_{\mathfrak{n}}, l_s : \mathfrak{l}, s : \uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}})$.

$$D_6 = \dfrac{D_{6,1} \quad \dfrac{D_{6,2} \quad \dfrac{D_{6,3} \quad \dfrac{D_{6,4} \quad D_{6,5}}{\Gamma_5 \vdash \overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6'} P}{\Gamma'_{2,2,2,1,2}; \emptyset; \emptyset \vdash {p_o}'(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')} I}{\Gamma_{2,2,2,1,2}; \emptyset; \emptyset \vdash c_o^{\star}(m_i)\,.p_o'(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')} R}{\Gamma_{2,2,2,1}; \emptyset; \emptyset \vdash \overline{c_o}\langle m_i\rangle \mid c_o^{\star}(m_i)\,.p_o'(y, l_s, s, z)\,.\,(\overline{p_{o,up}}\langle y, l_s, s, z\rangle \mid T_6')} P$$

where $P = \text{T-PAR}_\text{L}$, $R = \text{T-REP-B}_\text{L}$ and Lemma A.1.4, and $I = \text{T-IN-B}_\text{L}$ and Lemma A.1.4. Apply T-OUT-B_L and Lemma A.1.4, and then T-NAME_L on all subgoals to show $\Gamma, m_i : \downarrow_*(\mathfrak{i}_\circ)\,, c_o : \uparrow^\omega(\downarrow_*(\mathfrak{i}_\circ))\,; \emptyset; \emptyset \vdash \overline{c_o}\langle m_i\rangle$ for $D_{6,1}$ and $\Gamma, p_{o,up} : \uparrow^\omega(\mathfrak{o}_\circ)\,, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,; \emptyset; \emptyset \vdash \overline{p_{o,up}}\langle y, l_s, s, z\rangle$ for $D_{6,4}$. $\Gamma, c_o : \downarrow_*(\mathfrak{o}_\circ) \vdash c_o : \downarrow_*(\mathfrak{o}_\circ)$ for $D_{6,2}$ and $\Gamma, p_o{}' : \mathfrak{o} \vdash p_o{}' : \mathfrak{o}$ for $D_{6,3}$ follow from T-NAME_L. Let $\Gamma_{5,2} = \Gamma, c_o : \uparrow^\omega(\downarrow_*(\mathfrak{i}_\circ))\,, m_i : \downarrow_*(\mathfrak{i}_\circ)\,, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})$, $\Gamma'_{5,2} = \Gamma_{5,2}, m_{i,up} : \mathfrak{i}_*$, $\Gamma_{5,2,1} = \Gamma, m_i : \downarrow_*(\mathfrak{i}_\circ)\,, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,, m_{i,up} : \uparrow^\omega(\mathfrak{i}_\circ)$, $\Gamma_{5,2,2} = \Gamma, c_o : \uparrow^\omega(\downarrow_*(\mathfrak{i}_\circ))\,, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,, m_{i,up} : \downarrow_*(\mathfrak{i}_\circ)$, $\Gamma'_{5,2,2} = \Gamma_{5,2,2}, m_i : \mathfrak{i}_*$, $\Gamma_{5,2,2,1} = \Gamma, y, z : \mathfrak{v}_\mathfrak{n}, l_s : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,, m_{i,up} : \downarrow_*(\mathfrak{i}_\circ)\,, m_i : \uparrow^\omega(\mathfrak{i}_\circ)$, and $\Gamma_{5,2,2,2} = \Gamma, c_o : \uparrow^\omega(\downarrow_*(\mathfrak{i}_\circ))\,, m_i : \downarrow_*(\mathfrak{i}_\circ)$. $D_{6,5} =$

$$\dfrac{D_{6,6} \quad \dfrac{\dfrac{\dfrac{D_{6,7} \quad D_{6,8}}{\Gamma_{5,2,2,1}; \emptyset; \emptyset \vdash m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle}R' \quad D_{6,9}}{\Gamma'_{5,2,2}; \emptyset; \emptyset \vdash m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle}P}{\Gamma_{5,2,2}; \emptyset; \emptyset \vdash (\nu m_i : \mathfrak{i}_*)\,(m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle)}R}{\Gamma'_{5,2}; \emptyset; \emptyset \vdash T''_6 \mid (\nu m_i : \mathfrak{i}_*)\,(m_{i,up}{}^\star(y', l_r, r)\,.\,\overline{m_i}\langle y', l_r, r\rangle \mid \overline{c_o}\langle m_i\rangle)}P}{\Gamma_{5,2}; \emptyset; \emptyset \vdash T'_6}R$$

where $R = \text{T-RES-B}_\text{L}$, $P = \text{T-PAR}_\text{L}$, and $R' = \text{T-REP-B}_\text{L}$ and Lemma A.1.4. $\Gamma, m_{i,up} : \downarrow_*(\mathfrak{i}_\circ) \vdash m_{i,up} : \downarrow_*(\mathfrak{i}_\circ)$ for $D_{6,7}$ follows from T-NAME_L. Apply T-OUT-B_L and Lemma A.1.4, and then T-NAME_L on all subgoals to show $\Gamma, y, y', z : \mathfrak{v}_\mathfrak{n}, l_s, l_r : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,, m_i : \uparrow^\omega(\mathfrak{i}_\circ)\,, r : \uparrow^\omega(\mathfrak{r}_\circ)\,; \emptyset; \emptyset \vdash \overline{m_i}\langle y', l_r, r\rangle$ for $D_{6,8}$ and $\Gamma_{5,2,2,2}; \emptyset; \emptyset \vdash \overline{c_o}\langle m_i\rangle$ for $D_{6,9}$. Let $\Gamma_6 = \Gamma, y, y', z : \mathfrak{v}_\mathfrak{n}, l_s, l_r : \mathfrak{l}, s : \uparrow^\omega(\mathfrak{v}_\mathfrak{s})\,, m_{i,up} : \uparrow^\omega(\mathfrak{i}_\circ)\,, r : \uparrow^\omega(\mathfrak{r}_\circ)$.

$$D_{6,6} = \dfrac{D_{6,10} \quad \dfrac{\dfrac{D_{6,11} \quad D_{6,12} \quad D_{6,13}}{\Gamma_6; \emptyset; \emptyset \vdash [y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle}\text{T-MAT}_\text{L} \quad D_{6,14}}{\Gamma_6; \emptyset; \emptyset \vdash [y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle}\text{T-PAR}_\text{L}}{\Gamma_{5,2,1}; \emptyset; \emptyset \vdash m_i{}^\star(y', l_r, r)\,.\,([y' = y]\,\overline{r}\langle l_r, l_s, l_s, s, z\rangle \mid \overline{m_{i,up}}\langle y', l_r, r\rangle)}R$$

where $R = \text{T-REP-B}_\text{L}$ and Lemma A.1.4. $\Gamma, m_i : \downarrow_*(\mathfrak{i}_\circ) \vdash m_i : \downarrow_*(\mathfrak{i}_\circ)$ for $D_{6,10}$, $\Gamma_6 \vdash y' : \mathfrak{v}_\mathfrak{n}$ for $D_{6,11}$, and $\Gamma_6 \vdash y : \mathfrak{v}_\mathfrak{n}$ for $D_{6,12}$ follow from T-NAME_L. To show $\Gamma_6; \emptyset; \emptyset \vdash \overline{r}\langle l_r, l_s, l_s, s, z\rangle$ for $D_{6,13}$ and $\Gamma_6; \emptyset; \emptyset \vdash \overline{m_{i,up}}\langle y', l_r, r\rangle$ for $D_{6,14}$ apply T-OUT-B_L and Lemma A.1.4, and then T-NAME_L. The derivation of $\Gamma_{2,2,2,2}; \emptyset; \emptyset \vdash T_7$ for D_7 is similar to the derivation of D_6. Finally $\Gamma_{1,3}; \emptyset; \emptyset \vdash p_{o,up}(y, l, s, z)\,.\,\overline{p_o}\langle y, l, s, z\rangle \mid p_{i,up}(y, l, r)\,.\,\overline{p_i}\langle y, l, r\rangle$ for D_3 follows from T-PAR_L, then T-IN-B_L and Lemma A.1.4, T-OUT-B_L and Lemma A.1.4, and T-NAME_L on all leafs.

3. If $S = \sum_{i \in I} \pi_i.S_i$ for some $\pi_i.S_i \in \mathcal{P}_\text{m}$ then

$$\mathcal{T}^3_\text{L}[\![\, S \,]\!]^\text{m}_\text{a} = (\nu l : \mathfrak{l})\left(\mathcal{T}^3_\text{L}(\overline{l}\langle\top\rangle) \mid \prod_{i \in I} \mathcal{T}^3_\text{L}[\![\, \pi_i.S_i \,]\!]^\text{m}_\text{a}\right).$$

We have:

$$\dfrac{\dfrac{}{l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}\big(\overline{l}\langle\top\rangle\big)}\text{Lemma A.1.2 and Lemma 6.2.46}\quad D}{\dfrac{\Gamma, l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i\in I} \mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}{\Gamma;\emptyset;\emptyset \vdash (\nu l:\mathfrak{l})\,\big(\mathcal{T}^3_{\mathrm{L}}\big(\overline{l}\langle\top\rangle\big) \mid \prod_{i\in I} \mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)}\text{T-Res-B}_{\mathrm{L}}}\text{T-Par}_{\mathrm{L}}$$

To prove D, we have to show that $\Gamma, l:\mathfrak{l};\emptyset;\emptyset \vdash \prod_{i\in I} \mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. With T-Par$_{\mathrm{L}}$ we decompose this goal into several subgoals of the form $\Gamma, l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, where each π_i is either a τ, an output or an input prefix.

a) If $\pi_i = \tau$ then $\Gamma, l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, \tau.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ follows from Lemma A.1.2, Lemma A.1.3, and the induction hypothesis.

b) If $\pi_i = \overline{y}\langle z\rangle$ for some $y, z \in \mathcal{N}$ then

$$\mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} = \big(\nu s:\updownarrow^{\omega}_{+}(\mathfrak{v}_{\mathfrak{s}})\big)\,\big(\overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s)\,.\mathcal{T}^3_{\mathrm{L}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)\,.$$

Because $\Gamma\,(p_o) = \uparrow^{\omega}(\mathfrak{o}_\circ)$ and $\Gamma\,(\varphi^{\mathrm{m}}_{\mathrm{a}}(y)) = \Gamma\,(\varphi^{\mathrm{m}}_{\mathrm{a}}(z)) = \mathfrak{v}_{\mathfrak{n}}$, we have

$$\dfrac{\dfrac{D_1 \quad \dfrac{}{\Gamma', s:\downarrow_{+}(\mathfrak{v}_{\mathfrak{s}}) \vdash s:\downarrow_{+}(\mathfrak{v}_{\mathfrak{s}})}\text{T-Name}_{\mathrm{L}} \quad D_2}{\Gamma', s:\downarrow_{+}(\mathfrak{v}_{\mathfrak{s}})\,;\emptyset;\emptyset \vdash s(v_s)\,.\mathcal{T}^3_{\mathrm{L}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}I}{\dfrac{\Gamma', s:\updownarrow^{\omega}_{+}(\mathfrak{v}_{\mathfrak{s}})\,;\emptyset;\emptyset \vdash \overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s)\,.\mathcal{T}^3_{\mathrm{L}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}{\Gamma';\emptyset;\emptyset \vdash \big(\nu s:\updownarrow^{\omega}_{+}(\mathfrak{v}_{\mathfrak{s}})\big)\,\big(\overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s(v_s)\,.\mathcal{T}^3_{\mathrm{L}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)}R}\text{T-Par}_{\mathrm{L}}$$

where $I =$ T-In-B$_{\mathrm{L}}$ and Lemma A.1.4, $R =$ T-Res-B$_{\mathrm{L}}$, and $\Gamma' = \Gamma, l:\mathfrak{l}$. $\Gamma, l:\mathfrak{l}, s:\uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}})\,;\emptyset;\emptyset \vdash \overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle$ for D_1 follows from T-Out-B$_{\mathrm{L}}$ and Lemma A.1.4, and then T-Name$_{\mathrm{L}}$ for all subgoals.

$$D_2 = \dfrac{}{\Gamma, l:\mathfrak{l}, v_s:\mathfrak{v}_{\mathfrak{s}} \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{(IH) and Lemma 6.2.46}$$

c) If $\pi_i = y(x)$ for some $x, y \in \mathcal{N}$ then

$$\begin{aligned}
\mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} &= (\nu r:\updownarrow^{\omega}_{*}(\mathfrak{r}_\circ))\,(\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r\rangle \mid T_1)\\
T_1 &= r^{\star}(l_1, l_2, l_s, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.T_2\\
T_2 &= \mathcal{T}^3_{\mathrm{L}}\Big(\mathsf{test}\ l_1\ \ \mathsf{then\ test}\ l_2\ \ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle\\
&\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle\ \Big)
\end{aligned}$$

Because $\Gamma\,(p_i) = \uparrow^{\omega}(\mathfrak{i}_\circ)$ and $\Gamma\,(\varphi^{\mathrm{m}}_{\mathrm{a}}(y)) = \mathfrak{v}_{\mathfrak{n}}$, we have

$$\dfrac{\dfrac{D \quad D_1}{\Gamma, l:\mathfrak{l}, r:\updownarrow^{\omega}_{*}(\mathfrak{r}_\circ)\,;\emptyset;\emptyset \vdash \overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r\rangle \mid T_1}\text{T-Par}_{\mathrm{L}}}{\Gamma, l:\mathfrak{l};\emptyset;\emptyset \vdash \mathcal{T}^3_{\mathrm{L}}[\![\, \pi_i.S_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}}\text{T-Res-B}_{\mathrm{L}}$$

$\Gamma, l : \mathfrak{l}, r : \uparrow^{\omega}(\mathfrak{r}_{\circ}) ; \emptyset; \emptyset \vdash \overline{p_i}\langle \varphi_{\mathrm{a}}^{\mathrm{m}}(y) , l, r\rangle$ for D follows from T-Out-B$_{\mathrm{L}}$ and Lemma A.1.4, and then T-Name$_{\mathrm{L}}$ on all subgoals.

$$D_1 = \frac{D_1' \quad D_2}{\Gamma, l : \mathfrak{l}, r : \downarrow_*(\mathfrak{r}_{\circ}) ; \emptyset; \emptyset \vdash r^{\star}(l_1, l_2, l_s, s, \varphi_{\mathrm{a}}^{\mathrm{m}}(x)) . T_2}\,\text{T-Rep-B}_{\mathrm{L}}$$

$\Gamma, l : \mathfrak{l}, r : \downarrow_*(\mathfrak{r}_{\circ}) \vdash r : \downarrow_*(\mathfrak{r}_{\circ})$ for D_1' follows from T-Name$_{\mathrm{L}}$. $\Gamma, l : \mathfrak{l}, r : \downarrow_*(\mathfrak{r}_{\circ}) , l_1, l_2, l_s : \mathfrak{l}, s : \uparrow^{\omega}(\mathfrak{v}_{\mathfrak{s}}) , \varphi_{\mathrm{a}}^{\mathrm{m}}(x) : \mathfrak{v}_{\mathfrak{n}}; \emptyset; \emptyset \vdash T_2$ for D_2 follows from several applications of Lemmata A.1.2 and Lemmata A.1.3, one application of T-Out-B$_{\mathrm{L}}$ and Lemma A.1.4 (for the output on the sender lock), and the induction hypothesis.

4. The derivation for $S = y^{\star}(x) . S_2$, where $x, y \in \mathcal{N}$ and $S' \in \mathcal{P}_{\mathrm{m}}$, is similar to the cases above.

□

A.2. Semantic Properties

Lemma 6.3.52 states that the encoding $[\![\cdot]\!]_{\mathrm{a}}^{\mathrm{s}}$ satisfies operational completeness.

Proof of Lemma 6.3.52. By Definition 3.3.4 it suffice to show that:

$$\forall S, S' \in \mathcal{P}_{\mathrm{s}} \,.\, S \longmapsto S' \text{ implies } \exists T \in \mathcal{P}_{\mathrm{a}} \,.\, [\![S]\!]_{\mathrm{a}}^{\mathrm{s}} \Longmapsto T \wedge T \mathrel{\dot{\cong}}^{\downarrow 1} [\![S']\!]_{\mathrm{a}}^{\mathrm{s}}$$

The lemma then holds by induction over the number of steps in $S \Longmapsto S'$. To prove the condition above, we perform an induction over the proof tree that leads to the step $S \longmapsto S'$.

Base Case: We consider the axioms in Figure 2.3.

Case of Pi-Tau$_{\mathrm{m,s}}$ **:** In this case S is a single sum, one branch of which is guarded by τ, and S' is the continuation of this τ guarded branch, i.e., there are some finite index set I, some guards π_i, and some processes $P_i \in \mathcal{P}_{\mathrm{s}}$ such that $S = \sum_{i \in I} \pi_i . P_i$ with $\pi_j = \tau$ for some $j \in I$ and $S' = P_j$. The corresponding encodings are given by the following terms:

$$[\![S]\!]_{\mathrm{a}}^{\mathrm{s}} \equiv (\nu l) \left(\overline{l}\langle \top \rangle \mid \prod_{i \in I, i \neq j} [\![\pi_i . P_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle \bot \rangle \mid [\![P_j]\!]_{\mathrm{a}}^{\mathrm{s}}\ \mathsf{else}\ \overline{l}\langle \bot \rangle \right)$$
$$[\![S']\!]_{\mathrm{a}}^{\mathrm{s}} = [\![P_j]\!]_{\mathrm{a}}^{\mathrm{s}}$$

We observe that $[\![S]\!]_{\mathrm{a}}^{\mathrm{s}}$ can emulate the step $S \longmapsto S'$ by reducing the test-statement in the encoding of the j's branch. By Lemma 6.3.10,

$$[\![S]\!]_{\mathrm{a}}^{\mathrm{s}} \Longmapsto (\nu l) \left(\prod_{i \in I, i \neq j} [\![\pi_i . P_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid \overline{l}\langle \bot \rangle \mid [\![P_j]\!]_{\mathrm{a}}^{\mathrm{s}} \right) = T.$$

Note that, because of Lemma 6.3.46, we silently omit junk that results from the reduction of test-constructs here and in the following proofs. Furthermore, we observe that $T \equiv (\nu l)\left(\prod_{i\in I, i\neq j} [\![\, \pi_i.P_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right) \mid [\![\, P_j \,]\!]^{\mathrm{s}}_{\mathrm{a}}$, because, by the renaming policy $\varphi^{\mathrm{s}}_{\mathrm{a}}$ and Lemma 6.2.51, the name l is not free in the term $[\![\, P_j \,]\!]^{\mathrm{s}}_{\mathrm{a}}$. By Lemma 6.3.49, the term $(\nu l)\left(\prod_{i\in I, i\neq j} [\![\, \pi_i.P_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$ is junk. Finnaly, by Lemma 6.3.45, we conclude that $T \mathrel{\dot{\cong}}^{\downarrow 1} [\![\, S' \,]\!]^{\mathrm{s}}_{\mathrm{a}}$.

Case of Pi-Com$_{\mathrm{m,s}}$ **:** Here S is a parallel composition of two sums and S' is the parallel composition of the continuations of an input guarded branch of the first and a matching output guarded branch of the second sum, i.e., there are two finite index sets I_1, I_2, some guards π_i, and some processes $P_i, Q_i \in \mathcal{P}_{\mathrm{s}}$ such that $S = \sum_{i\in I_1} \pi_i.P_i \mid \sum_{i\in I_2} \pi_i.Q_i$ with $\pi_{j_1} = y(x)$ and $\pi_{j_2} = \overline{y}\langle z\rangle$ for some $j_1 \in I_1$, some $j_2 \in I_2$, and $x, y, z \in \mathcal{N}$ and $S' = \{\, z/x \,\}\, P_{j_1} \mid Q_{j_2}$. The encodings of S and S' are given by the following terms:

$$
\begin{aligned}
[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv (\nu l)\Big(\overline{l}\langle\top\rangle \mid & \prod_{i\in I_1, i\neq j_1} [\![\, \pi_i.P_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \mid (\nu r)\left(\overline{r} \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(y)\big(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big)\,.\right. \\
& \quad \mathsf{test}\ l\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_{j_1} \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \qquad\qquad \mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
& \quad \left.\mathsf{else}\ \overline{l}\langle\top\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\big\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big\rangle\,\right)\Big) \\
\mid (\nu l)\Big(\overline{l}\langle\top\rangle \mid & \prod_{i\in I_1, i\neq j_1} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \mid (\nu s)\left(\overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\langle l, s, \varphi^{\mathrm{s}}_{\mathrm{a}}(z)\rangle \mid s.[\![\, Q_{j_2} \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right)\Big) \\
[\![\, S' \,]\!]^{\mathrm{s}}_{\mathrm{a}} = [\![\, \{\, z/x \,\}\, (P_{j_1}) \,]\!]^{\mathrm{s}}_{\mathrm{a}} & \mid [\![\, Q_{j_2} \,]\!]^{\mathrm{s}}_{\mathrm{a}}
\end{aligned}
$$

To emulate the source term step $S \longmapsto S'$ first the receiver lock has to be reduced to enable a communication over $\varphi^{\mathrm{s}}_{\mathrm{a}}(y)$. Then the test-construct and the sender lock are reduced to complete the emulation of the source term step.

$$
\begin{aligned}
[\![\, S \,]\!]^{\mathrm{s}}_{\mathrm{a}} \longmapsto\!\!\!\!\!\!= (\nu l_1, l_2, s)\Big(\{\, l_1/l \,\}\Big(& \prod_{i\in I_1, i\neq j_1} [\![\, \pi_i.P_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\Big) \\
& \mid (\nu r)\left(\overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \{\, \varphi^{\mathrm{s}}_{\mathrm{a}}(z)/\varphi^{\mathrm{s}}_{\mathrm{a}}(x) \,\}\,([\![\, P_{j_1} \,]\!]^{\mathrm{s}}_{\mathrm{a}})\right. \\
& \quad \mid r^{\star}.\varphi^{\mathrm{s}}_{\mathrm{a}}(y)\big(l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big)\,. \\
& \qquad \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_{j_1} \,]\!]^{\mathrm{s}}_{\mathrm{a}} \\
& \qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r} \\
& \qquad\quad \left.\mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{\varphi^{\mathrm{s}}_{\mathrm{a}}(y)}\big\langle l', s, \varphi^{\mathrm{s}}_{\mathrm{a}}(x)\big\rangle\,\right) \\
& \mid \{\, l_2/l \,\}\left(\prod_{i\in I_1, i\neq j_2} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{s}}_{\mathrm{a}}\right) \mid [\![\, Q_{j_2} \,]\!]^{\mathrm{s}}_{\mathrm{a}}\Big) = T
\end{aligned}
$$

By Corollary 6.1.6, $\{\, {}^{\varphi_{\mathrm{a}}^{\mathrm{s}}(z)}/_{\varphi_{\mathrm{a}}^{\mathrm{s}}(x)} \,\} [\![P_{j_1}]\!]_{\mathrm{a}}^{\mathrm{s}} \equiv_\alpha [\![\{\, {}^{z}/_{x} \,\} P_{j_1}]\!]_{\mathrm{a}}^{\mathrm{s}}$. To show that $T \mathrel{\dot{\cong}}^{\downarrow 1} [\![S']\!]_{\mathrm{a}}^{\mathrm{m}}$ we stepwise reduce T by ignoring junk. Since $l_1, l_2, r, s \notin \mathsf{fn}\big([\![\{\, {}^{z}/_{x} \,\} P_{j_1}]\!]_{\mathrm{a}}^{\mathrm{m}}\big) \cup \mathsf{fn}\big([\![Q_{j_2}]\!]_{\mathrm{a}}^{\mathrm{m}}\big)$, we can reorder the term according to the restrictions on l_1, l_2, r and the restriction on s can be omitted. The term

$$(\nu r)\,\big(r^{\star}.\varphi_{\mathrm{a}}^{\mathrm{s}}(y)\big(l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x)\big)\,.\\ \mathsf{test}\ l_1\ \mathsf{then}\ \mathsf{test}\ l'\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{s} \mid [\![P_{j_1}]\!]_{\mathrm{a}}^{\mathrm{s}}\ \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l'}\langle\bot\rangle \mid \overline{r}\\ \mathsf{else}\ \overline{l_1}\langle\bot\rangle \mid \overline{\varphi_{\mathrm{a}}^{\mathrm{s}}(y)}\big\langle l', s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x)\big\rangle\,\big)$$

is obviously junk, since it is closed and can not perform any step. Moreover, by Lemma 6.3.49, the term $(\nu l)\left(\prod_{i\in I_1, i\neq j_1} [\![\pi_i.P_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid \overline{l}\langle\bot\rangle\right)$ and the term $(\nu l)\left(\prod_{i\in I_1, i\neq j_2} [\![\pi_i.Q_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid \overline{l}\langle\bot\rangle\right)$ are junk. So, by Lemma 6.3.45, we conclude $T \mathrel{\dot{\cong}}^{\downarrow 1} [\![S']\!]_{\mathrm{a}}^{\mathrm{s}}$.

Case of Pi-Rep$_{\mathrm{m,s}}$ **:** Here S is a parallel composition of a replicated input and a sum and S' is the parallel composition of the replicated input, the continuation of the replicated input, and the continuation of a matching output guarded branch, i.e., there is a finite index set I, some guards π_i, and some processes $P, Q_i \in \mathcal{P}_{\mathrm{s}}$ such that $S = y^{\star}(x)\,.P \mid \sum_{i\in I} \pi_i.Q_i$ with $\pi_j = \overline{y}\langle z\rangle$ for some $j \in I$ and $x, y, z \in \mathcal{N}$, and $S' = \{\, {}^{z}/_{x} \,\} P \mid Q_j \mid y^{\star}(x)\,.P$. The encodings of S and S' are given by the following terms:

$$[\![S]\!]_{\mathrm{a}}^{\mathrm{s}} \equiv \varphi_{\mathrm{a}}^{\mathrm{s}}(y)^{\star}(l, s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x))\,.\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid [\![P]\!]_{\mathrm{a}}^{\mathrm{s}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle\\ \mid (\nu l)\left(\overline{l}\langle\top\rangle \mid \prod_{i\in I_1, i\neq j} [\![\pi_i.Q_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid (\nu s)\left(\overline{\varphi_{\mathrm{a}}^{\mathrm{s}}(y)}\langle l, s, \varphi_{\mathrm{a}}^{\mathrm{s}}(z)\rangle \mid s.[\![Q_j]\!]_{\mathrm{a}}^{\mathrm{s}}\right)\right)$$

$$[\![S']\!]_{\mathrm{a}}^{\mathrm{s}} = [\![\{\, {}^{z}/_{x} \,\} (P)]\!]_{\mathrm{a}}^{\mathrm{s}} \mid [\![Q_j]\!]_{\mathrm{a}}^{\mathrm{s}} \mid [\![y^{\star}(x)\,.P]\!]_{\mathrm{a}}^{\mathrm{s}}$$

To emulate the source term step $S \longmapsto S'$, first the two subprocesses of $[\![S]\!]_{\mathrm{a}}^{\mathrm{s}}$ communicate over $\varphi_{\mathrm{a}}^{\mathrm{s}}(y)$. Then the test-construct and the sender lock are reduced to complete the emulation of the source term step.

$$\begin{aligned}
[\![S]\!]_{\mathrm{a}}^{\mathrm{s}} &\Longmapsto (\nu l, s)\Big(\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid \{\, {}^{\varphi_{\mathrm{a}}^{\mathrm{s}}(z)}/_{\varphi_{\mathrm{a}}^{\mathrm{s}}(x)} \,\} ([\![P]\!]_{\mathrm{a}}^{\mathrm{s}})\ \mathsf{else}\ \overline{l}\langle\bot\rangle\\
&\qquad \mid \varphi_{\mathrm{a}}^{\mathrm{s}}(y)^{\star}(l, s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x))\,.\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid [\![P]\!]_{\mathrm{a}}^{\mathrm{s}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle\\
&\qquad \mid \overline{l}\langle\top\rangle \mid \prod_{i\in I_1, i\neq j} [\![\pi_i.Q_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid s.[\![Q_j]\!]_{\mathrm{a}}^{\mathrm{s}}\Big)\\
&\Longmapsto (\nu l, s)\Big(\overline{l}\langle\bot\rangle \mid \{\, {}^{\varphi_{\mathrm{a}}^{\mathrm{s}}(z)}/_{\varphi_{\mathrm{a}}^{\mathrm{s}}(x)} \,\} ([\![P]\!]_{\mathrm{a}}^{\mathrm{s}})\\
&\qquad \mid \varphi_{\mathrm{a}}^{\mathrm{s}}(y)^{\star}(l, s, \varphi_{\mathrm{a}}^{\mathrm{s}}(x))\,.\mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid \overline{s} \mid [\![P]\!]_{\mathrm{a}}^{\mathrm{s}}\\
&\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l}\langle\bot\rangle\\
&\qquad \mid \prod_{i\in I_1, i\neq j} [\![\pi_i.Q_i]\!]_{\mathrm{a}}^{\mathrm{s}} \mid [\![Q_j]\!]_{\mathrm{a}}^{\mathrm{s}}\Big) = T
\end{aligned}$$

By Corollary 6.1.6, $\{\ {}^{\varphi^{\mathrm{s}}_{\mathrm{a}}(z)}/_{\varphi^{\mathrm{s}}_{\mathrm{a}}(x)}\ \}\ [\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \equiv_\alpha [\![\ \{\ {}^{z}/_{x}\ \}\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. Since l, s are not free in $[\![\ \{\ {}^{z}/_{x}\ \}\ P\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$ or $[\![\ Q_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$, we can reorder the term according to the restriction on l and the restriction on s can be omitted. Because of Lemma 6.3.49, $(\nu l)\left(\prod_{i\in I, i\neq j} [\![\ \pi_i.Q_i\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$ is junk. By Lemma 6.3.39, $\dot{\cong}^{\downarrow 1}$ includes structural congruence. Thus, by Lemma 6.3.45, $T \dot{\cong}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$.

Induction Hypothesis: $S_1 \longmapsto S_1'$ implies $\exists T_1 \in \mathcal{P}_{\mathrm{a}}\ .\ [\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1 \wedge T_1 \dot{\cong}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$

Induction Step: We have to consider the remaining reduction rules of π_{s} in Figure 2.3.

Case of Pi-Par$_{\mathrm{m,s,a,p}}$ **:** Then $S = S_1 \mid S_2$ for some $S_1, S_2 \in \mathcal{P}_{\mathrm{s}}$, $S_1 \longmapsto S_1'$, and $S' = S_1' \mid S_2$. By the induction hypothesis there is some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $[\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1$ and $T_1 \dot{\cong}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. Since $[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ translates the parallel operator homomorphically, i.e., $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} = [\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} = [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$, we can apply rule Pi-Par$_{\mathrm{m,s,a,p}}$ to conclude from $[\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1$ that $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1 \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}} = T$. By Definition 6.3.37, $T_1 \dot{\cong}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ implies $\mathcal{C}(T_1) \dot{\approx}^{\downarrow 1} \mathcal{C}([\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$ for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{s}}$. Since $[\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{s}}$, the quantification over $\mathcal{C}$ includes all contexts $\mathcal{C}$ such that $\mathcal{C}([\cdot]) = \mathcal{C}'([\cdot] \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$. Because of that, $\mathcal{C}'(T_1 \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \dot{\approx}^{\downarrow 1} \mathcal{C}'([\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mid [\![\ S_2\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$ for all such contexts $\mathcal{C}'([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$. By Definition 6.3.37, we conclude $T \dot{\cong}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$.

Case of Pi-Res$_{\mathrm{m,s,a,p}}$ **:** Then $S = (\nu x)\, S_1$ for some $x \in \mathcal{N}$ and some $S_1 \in \mathcal{P}_{\mathrm{s}}$, $S_1 \longmapsto S_1'$, and $S' = (\nu x)\, S_1'$. By the induction hypothesis there is some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $[\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1$ and $T_1 \dot{\cong}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. Since $[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ translates restriction homomorphically, i.e., $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} = (\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$, we can apply rule Pi-Res$_{\mathrm{m,s,a,p}}$ to conclude from $[\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1$ that $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto (\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, T_1 = T$. By Definition 6.3.37, $T_1 \dot{\cong}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ implies $\mathcal{C}(T_1) \dot{\approx}^{\downarrow 1} \mathcal{C}([\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$ for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{s}}$. Since $(\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{s}}$, the quantification over $\mathcal{C}$ includes all contexts $\mathcal{C}$ such that $\mathcal{C}([\cdot]) = \mathcal{C}'((\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\cdot])$. Because of that, $\mathcal{C}'((\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, T_1) \dot{\approx}^{\downarrow 1} \mathcal{C}'((\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\, [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$ for all such contexts $\mathcal{C}'([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$. By Definition 6.3.37, we conclude $T \dot{\cong}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$.

Case of Pi-Cong$_{\mathrm{m,s,a,p}}$ **:** Then $S \equiv S_1$ for some $S_1 \in \mathcal{P}_{\mathrm{s}}$, $S_1 \longmapsto S_1'$, and $S_1' \equiv S'$. By Lemma 6.3.41, the encoding $[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ preserves structural congruence of source terms modulo $\dot{\cong}^{\downarrow 1}$. So $S \equiv S_1$ and $S_1' \equiv S'$ imply $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \dot{\cong}^{\downarrow 1} [\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ and $[\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \dot{\cong}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. By Definition 6.3.37, for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![\ P\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}\restriction_{[\![\ \cdot\ ]\!]^{\mathrm{s}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{s}}$ we have $\mathcal{C}([\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \dot{\approx}^{\downarrow 1} \mathcal{C}([\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}})$ and, hence, $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \dot{\approx}^{\downarrow 1} [\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. Thus, by Definition 6.3.32, for each sequence $[\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T$ there is a sequence $[\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \Longmapsto T_1$ for some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $T \dot{\approx}^{\downarrow 1} T_1$. The same holds for all contexts $\mathcal{C}$, i.e., for each sequence

$\mathcal{C}([\![\ S\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \Longmapsto \mathcal{C}(T)$ there is a sequence $\mathcal{C}([\![\ S_1\ ]\!]^{\mathrm{s}}_{\mathrm{a}}) \Longmapsto \mathcal{C}(T_1)$ for some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}(T) \mathrel{\dot{\approx}}^{\downarrow 1} \mathcal{C}(T_1)$. So, by Definition 6.3.37, $T \mathrel{\dot{\cong}}^{\downarrow 1} T_1$. By the induction hypothesis, $T_1 \mathrel{\dot{\cong}}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$. Then $T \mathrel{\dot{\cong}}^{\downarrow 1} T_1$, $T_1 \mathrel{\dot{\cong}}^{\downarrow 1} [\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$, and $[\![\ S_1'\ ]\!]^{\mathrm{s}}_{\mathrm{a}} \mathrel{\dot{\cong}}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$ imply $T \mathrel{\dot{\cong}}^{\downarrow 1} [\![\ S'\ ]\!]^{\mathrm{s}}_{\mathrm{a}}$.

□

Lemma 6.3.53 states that the encodings $[\![\ \cdot\ ]\!]^{\mathrm{m}}_{\mathrm{p}}$ and $[\![\ \cdot\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$ satisfy operational completeness.

Proof of Lemma 6.3.53. By Definition 3.3.4 it suffice to show that:

$$\forall S, S' \in \mathcal{P}_{\mathrm{m}}\ .\ S \longmapsto S' \text{ implies } \exists T \in \mathcal{P}_{\mathrm{a}}\ .\ [\![\ S\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T \wedge T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![\ S'\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$$

The lemma then holds by induction over the number of steps in $S \Longmapsto S'$. To prove the condition above, we perform an induction over the proof tree that leads to the step $S \longmapsto S'$.

Base Case: We consider the axioms in Figure 2.3.

Case of Pi-Tau$_{\mathrm{m,s}}$ **:** In this case S is a single sum, one branch of which is guarded by τ, and S' is the continuation of this τ guarded branch, i.e., there are some finite index set I, some guards π_i, and some processes $P_i \in \mathcal{P}_{\mathrm{m}}$ such that $S = \sum_{i \in I} \pi_i.P_i$ with $\pi_j = \tau$ for some $j \in I$ and $S' = P_j$. The corresponding encodings are given by the following terms:

$$[\![\ S\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv (\nu l)\left(\overline{l}\langle\top\rangle \mid \prod_{i \in I, i \neq j} [\![\ \pi_i.P_i\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{test}\ l\ \mathsf{then}\ \overline{l}\langle\bot\rangle \mid [\![\ P_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}\ \mathsf{else}\ \overline{l}\langle\bot\rangle\right)$$
$$[\![\ S'\ ]\!]^{\mathrm{m}}_{\mathrm{a}} = [\![\ P_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$$

We observe that $[\![\ S\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$ can emulate the step $S \longmapsto S'$ by reducing the test-construct in the encoding of the jth branch. By Lemma 6.3.10,

$$[\![\ S\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto (\nu l)\left(\prod_{i \in I, i \neq j} [\![\ \pi_i.P_i\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle \mid [\![\ P_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}\right) = T.$$

Hence, $T \equiv (\nu l)\left(\prod_{i \in I, i \neq j} [\![\ \pi_i.P_i\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right) \mid [\![\ P_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$, because l is not free in $[\![\ P_j\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$. By Lemma 6.3.49, $(\nu l)\left(\prod_{i \in I, i \neq j} [\![\ \pi_i.P_i\ ]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$ is junk. By Lemma 6.3.45, we conclude that $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![\ S'\ ]\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of Pi-Com$_{\mathrm{m,s}}$ **:** Here S is a parallel composition of two sums and S' is the parallel composition of the continuations of an input guarded branch of the first and a matching output guarded branch of the second sum, i.e., there are two finite index sets I_1, I_2, some guards π_i, and some processes $P_i, Q_i \in \mathcal{P}_{\mathrm{m}}$

such that $S = \sum_{i\in I_1} \pi_i.P_i \mid \sum_{i\in I_2} \pi_i.Q_i$ with $\pi_{j_1} = y(x)$ and $\pi_{j_2} = \overline{y}\langle z\rangle$ for some $j_1 \in I_1$, some $j_2 \in I_2$, and $x, y, z \in \mathcal{N}$, and $S' = \{\, {}^{z}\!/\!_{x} \,\} P_{j_1} \mid Q_{j_2}$.

$$
\begin{aligned}
&[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv \\
&(\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
&\quad (\nu p_o, p_i)\,(\\
&\quad\quad (\nu l)\,(\,\overline{l}\langle\top\rangle \mid \prod_{i\in I_1, i\neq j_1} [\![\, \pi_i.P_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\quad\quad\quad\quad \mid (\nu r)\,(\,\overline{p_i}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, r\rangle \mid r^{\star}(l_1, l_2, -, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,. \\
&\quad\quad\quad\quad\quad\quad \mathsf{test}\ l_1\ \ \mathsf{then\ test}\ l_2\ \ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\, P_{j_1} \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \\
&\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{else}\ \overline{l_1}\langle\bot\rangle\,)\,) \\
&\quad\quad \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
&\quad \mid (\nu p_o, p_i)\,(\ (\nu l)\,(\,\overline{l}\langle\top\rangle \mid \prod_{i\in I_2, i\neq j_2} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\quad\quad\quad\quad\quad\quad \mid (\nu s)\,(\overline{p_o}\langle \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s.[\![\, Q_{j_2} \,]\!]^{\mathrm{m}}_{\mathrm{a}})\,)\\
&\quad\quad\quad\quad \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\quad \mid \mathsf{pushReq})\\
&[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}} = \ (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
&\quad\quad (\nu p_o, p_i)\,([\![\, \{\, {}^{z}\!/\!_{x} \,\} P_{j_1} \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
&\quad\quad \mid (\nu p_o, p_i)\,([\![\, Q_{j_2} \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\quad\quad \mid \mathsf{pushReq})
\end{aligned}
$$

To emulate the source term step $S \longmapsto S'$, the endings of the two sums in S have to interact with the encoding of the parallel operator between them. First the input and output register themselves to the encoding of the parallel operator by pushing requests. These requests are then combined and a test on the respective sum locks[1] is induced by providing an output on the receiver lock. Finally the test-construct is reduced to complete the emulation of the source term step.

$$[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}, l_a, l_b, r, s)\,($$

[1] In order to avoid a deadlock caused by multiple simultaneous such tests on sum locks, the sum locks are ordered by ensuring that always the left one is checked first.

$$
\begin{aligned}
&(\nu p_o, p_i)\,\big(\,\{\,{}^{l_a}/_{l}\,\}\,\Big(\prod_{i\in I_1, i\neq j_1} [\![\,\pi_i.P_i\,]\!]^{\mathrm{m}}_{\mathrm{a}}\Big) \mid \overline{l_a}\langle\bot\rangle \mid \overline{l_b}\langle\bot\rangle \\
&\quad \mid \{\,{}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(z)}/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x)}\,\}\,[\![\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\quad \mid r^{\star}(l_1, l_2, -, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle \\
&\quad \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \mid \overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle\,\big) \\
&\mid (\nu p_o, p_i)\,\big(\,\{\,{}^{l_b}/_{l}\,\}\,\Big(\prod_{i\in I_2, i\neq j_2} [\![\,\pi_i.Q_i\,]\!]^{\mathrm{m}}_{\mathrm{a}}\Big) \mid [\![\,Q_{j_2}\,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\quad \mid (\nu m_{i,up})\,\big(\,{m_i}^{\star}\big(y', l_r, r\big)\,.\big(\big[y' = \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\big]\,\overline{r}\langle l_r, l_b, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \\
&\qquad\qquad \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big) \\
&\qquad \mid \overline{m_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle \\
&\qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \mathsf{procRightOutReq})\,\big) \\
&\quad \mid \overline{p_{o,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid \mathsf{procRightInReq}\big) \\
&\mid \mathsf{pushReq}\big) = T
\end{aligned}
$$

By Corollary 6.1.6, $\{\,{}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(z)}/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x)}\,\}\,[\![\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv_{\alpha} [\![\,\{\,{}^{z}/_{x}\,\}\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}}$. To show that $T \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\,S'\,]\!]^{\mathrm{m}}_{\mathrm{a}}$, we stepwise reduce T by ignoring junk. By Lemma 6.3.48, the requests $\overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle$, $\overline{m_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle$, and $\overline{p_{o,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle$ are junk. Next, by Lemma 6.3.50, we can ignore the term

$$
\begin{aligned}
&r^{\star}(l_1, l_2, -, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l_1\ \mathsf{then\ test}\ l_2\ \mathsf{then}\ \overline{l_1}\langle\bot\rangle \mid \overline{l_2}\langle\bot\rangle \mid \overline{s} \mid [\![\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\top\rangle \mid \overline{l_2}\langle\bot\rangle \\
&\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_1}\langle\bot\rangle\,.
\end{aligned}
$$

And, by Lemma 6.3.51, we can ignore $[y' = \varphi^{\mathrm{m}}_{\mathrm{a}}(y)]\,\overline{r}\langle l_r, l_b, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle$, so

$$
{m_i}^{\star}\big(y', l_r, r\big)\,.\,\big(\big[y' = \varphi^{\mathrm{m}}_{\mathrm{a}}(y)\big]\,\overline{r}\langle l_r, l_b, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big)
$$

becomes $m_i \twoheadrightarrow m_{i,up}$. Note that this forwarder and the following forwarder $m_{i,up} \twoheadrightarrow m_i$ for an other instance of m_i may be necessary to emulate further source term steps, but since they perform only invisible steps, they do not influence the state of T modulo $\overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$ in comparison to a fresh chain of right requests as in $[\![\,S'\,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Finally, since $l_a, l_b, r, s \notin \mathsf{fn}\big([\![\,P_{j_1}\,]\!]^{\mathrm{m}}_{\mathrm{a}}\big) \cup \mathsf{fn}\big([\![\,P_{j_2}\,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)$, we can reorder the term according to the restrictions on l_a, l_b, r and the restriction on s can be omitted. By Lemma 6.3.49, $(\nu l)\,\Big(\prod_{i\in I_1, i\neq j_1} [\![\,\pi_i.P_i\,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\Big)$ and $(\nu l)\,\Big(\prod_{i\in I_1, i\neq j_2} [\![\,\pi_i.Q_i\,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\Big)$ are junk. So, by Lemma 6.3.45, we conclude $T \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\,S'\,]\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of PI-REP$_{\mathrm{m,s}}$ **:** Here S is a parallel composition of a replicated input and a sum and S' is the parallel composition of the replicated input, the continuation of the replicated input, and the continuation of a matching output guarded branch, i.e., there is a finite index set I, some guards π_i, and some processes $P, Q_i \in \mathcal{P}_{\mathrm{m}}$ such that $S = y^\star(x)\,.P \mid \sum_{i\in I} \pi_i.Q_i$ with $\pi_j = \overline{y}\langle z\rangle$ for some $j \in I$ and $x, y, z \in \mathcal{N}$, and $S' = \{\, {}^{z}/_{x} \,\} P \mid Q_j \mid y^\star(x)\,.P$.

$$
\begin{aligned}
[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv\ & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
& (\nu p_o, p_i)\,(\\
& (\nu l, r, c_{r1}, c_{r2}, r_o, r_i)\,(\\
& r^\star(-,-,l_s,s,\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(x)\rangle\\
& \qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \overline{l_s}\langle\bot\rangle\\
& \overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid \overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid \overline{l}\langle\top\rangle\,\overline{c_{r2}}\langle r_o, r_i\rangle\\
& \mid {c_{r1}}^\star(\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.c_{r2}(r_o, r_i)\,.\\
& (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
& \quad \mathsf{pushReqIn}\\
& \quad \mid (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
& \quad \mid (\nu r_o, r_i)\,((\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut}))\,))\\
& \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
& (\nu p_o, p_i)\,(\ (\nu l)\,(\,\overline{l}\langle\top\rangle \mid \prod_{i\in I_2, i\neq j} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\\
& \qquad\qquad \mid (\nu s)\,(\overline{p_o}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid s.[\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}})\,)\\
& \qquad \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
& \mathsf{pushReq})\\
[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}} = \ & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
& (\nu p_o, p_i)\,(\\
& (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
& (\nu p_o, p_i)\,(\ [\![\, \{\, {}^{z}/_{x} \,\}(P) \,]\!]^{\mathrm{m}}_{\mathrm{a}}\\
& \qquad\qquad \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
& (\nu p_o, p_i)\,([\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
& \mathsf{pushReq})\\
& \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq})\\
& (\nu p_o, p_i)\,([\![\, y^\star(x)\,.P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
& \mathsf{pushReq})
\end{aligned}
$$

To emulate the source term step $S \longmapsto S'$, the two subterms of $[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ have to interact with the encoding of the parallel operator between them. First the replicated input and the output register themselves to the encoding of the

parallel operator by pushing requests. Then the requests are combined and a test on the sum lock of the sender is induced by providing an output on the receiver lock. Next the test-statement is reduced. To complete the emulation of the source term step, the continuation of the replicated input encoding is unguarded and placed within an adoption of the parallel operator encoding.

$$
\begin{aligned}
&[\![\, S \,]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}, l_a, l_b, r, s)\,\big(\\
&\quad (\nu p_o, p_i)\,\big(\,(\nu c_{r1}, c_{r2}, r_o, r_i)\,\big(\\
&\qquad \overline{l_b}\langle\bot\rangle \mid r^{\star}(-,-,l_s,s,\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(x)\rangle\ \mathsf{else}\ \overline{l_s}\langle\bot\rangle\\
&\qquad \mid \overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle \mid \overline{l_a}\langle\top\rangle \mid c_{r1}{}^{\star}(\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.c_{r2}(r_o, r_i)\,.\\
&\qquad\quad (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,\big(\mathsf{pushReqIn}\\
&\qquad\qquad \mid (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\qquad\qquad \mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,\big)\\
&\qquad \mid (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,\big(\mathsf{pushReqIn}\\
&\qquad\qquad \mid (\nu p_o, p_i)\,\big(\,\{\,{}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(z)}\!/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x)}\,\}\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}})\\
&\qquad\qquad\qquad\qquad \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}\big)\\
&\qquad\qquad \mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,\big)\big)\\
&\qquad \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \mid \overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle\,\big)\\
&\mid (\nu p_o, p_i)\,\big(\,\{\,{}^{l_b}\!/_{l}\,\}\left(\prod_{i\in I_2, i\neq j} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{m}}_{\mathrm{a}}\right) \mid [\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}}\\
&\qquad\quad \mid (\nu m_{i,up})\,\big(\,m_i{}^{\star}\big(y', l_r, r\big)\,.\big([y' = \varphi^{\mathrm{m}}_{\mathrm{a}}(y)]\,\overline{r}\langle l_r, l_b, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle\\
&\qquad\qquad\qquad\quad \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big)\\
&\qquad\qquad\qquad \mid \overline{m_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_a, r\rangle\\
&\qquad\qquad\qquad \mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \mathsf{procRightOutReq})\,\big)\\
&\qquad\quad \mid \overline{p_{o,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid \mathsf{procRightInReq}\big)\\
&\mid \mathsf{pushReq}\big) = T
\end{aligned}
$$

By Corollary 6.1.6, $\{\,{}^{\varphi^{\mathrm{m}}_{\mathrm{a}}(z)}\!/_{\varphi^{\mathrm{m}}_{\mathrm{a}}(x)}\,\}\,[\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \equiv_{\alpha} [\![\,\{\,{}^{z}\!/_{x}\,\}\,P\,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Here it does not suffice to ignore junk to prove that $T \overset{\cdot}{\leftrightarrows}{}^{\downarrow 3}_{\mathrm{c}}\ [\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, because in $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ there are two encoded parallel operators whereas in T there is only one. Nevertheless, we start reducing T by omitting junk. Since the sum lock l_b is instantiated by false, by Lemma 6.3.48, the request $\overline{p_{o,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y)\,, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle$ is junk. Moreover, by Lemma 6.3.50, the term

$$
m_i{}^{\star}\big(y', l_r, r\big)\,.\,\big([y' = \varphi^{\mathrm{m}}_{\mathrm{a}}(y)]\,\overline{r}\langle l_r, l_b, l_b, s, \varphi^{\mathrm{m}}_{\mathrm{a}}(z)\rangle \mid \overline{m_{i,up}}\big\langle y', l_r, r\big\rangle\big)
$$

reduces to the forwarder $m_i \twoheadrightarrow m_{i,up}$. Since $l_a, l_b, r, s \notin \mathsf{fn}([\![\,\{\,{}^{z}\!/_{x}\,\}\,P\,]\!]^{\mathrm{m}}_{\mathrm{a}}) \cup \mathsf{fn}\big([\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}}\big)$, we can reorder the term according to the restrictions on l_a, l_b, and r and the restriction on s can be omitted. By Lemma 6.3.49, then

$(\nu l)\left(\prod_{i\in I, i\neq j} [\![\, \pi_i.Q_i \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \overline{l}\langle\bot\rangle\right)$ is junk. By Lemma 6.3.45, $T \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} T'$, where

$$
\begin{aligned}
&(\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
&\quad(\nu p_o, p_i)\,(\\
&\quad\quad(\nu l, r, c_{r1}, c_{r2}, r_o, r_i)\,(\\
&\quad\quad\quad r^{\star}(-,-,l_s,s,\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(x)\rangle\ \mathsf{else}\ \overline{l_s}\langle\bot\rangle\\
&\quad\quad\quad \mid \overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid \overline{l}\langle\top\rangle\\
&\quad\quad\quad \mid {c_{r1}}^{\star}(\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,.c_{r2}(r_o, r_i)\,.\\
&\quad\quad\quad\quad(\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\\
&\quad\quad\quad\quad\mathsf{pushReqIn}\\
&\quad\quad\quad\quad\quad\mid (\nu p_o, p_i)\,([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\quad\quad\quad\quad\quad\mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,)\\
&\quad\quad\quad\mid (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\,(\mathsf{pushReqIn}\\
&\quad\quad\quad\quad\mid (\nu p_o, p_i)\,(\,[\![\, \{\, z/x \,\}\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})\\
&\quad\quad\quad\quad\mid (\nu r_o, r_i)\,(\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,)\,)\\
&\quad\quad\mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq} \mid \overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle\,)\\
&\quad\mid (\nu p_o, p_i)\,(\,[\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightInReq}\\
&\quad\quad\quad\quad\mid (\nu m_{i,up})\,(\,m_i \twoheadrightarrow m_{i,up} \mid \overline{m_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle\\
&\quad\quad\quad\quad\quad\mid (\nu m_i)\,(m_{i,up} \twoheadrightarrow m_i \mid \mathsf{procRightOutReq})\,)\,)\\
&\quad\mid \mathsf{pushReq}) = T'
\end{aligned}
$$

In comparison to $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ the encoded subterms $[\![\, \{\, z/x \,\}\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, $[\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, and the term representing $[\![\, y^{\star}(x)\,.P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ appear in the wrong order. However, since $S' \equiv S'' = (y^{\star}(x)\,.P \mid \{\, z/x \,\}\, P) \mid Q_j$ and $\overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$, by Lemma 6.3.42, preserves structural congruence of source terms, we have $[\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}} \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\, S'' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. As in the case before, on the right hand side of the parallel operator encoding there are the two forwarders $m_i \twoheadrightarrow m_{i,up}$ and $m_{i,up} \twoheadrightarrow m_i$ (for different instances of m_i). Again they are necessary to emulate further source term steps on the continuation $[\![\, Q_j \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, but, since they perform only invisible steps, they do not influence the state of T' modulo $\overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$.

Moreover there is the request $\overline{m_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$, to enable an emulation of a communication of Q_j and $y^{\star}(x)\,.P$. Note that there is also the request $\overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$ at the right hand side of the parallel operator encoding, but the request $\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$, which belongs to $[\![\, y^{\star}(x)\,.P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, is missing. However, since by $m_{i,up} \twoheadrightarrow m_i$ the request $\overline{p_{i,up}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$ is forwarded to $\overline{m_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$ by administrative steps and since this configuration is equal to one application of $\mathsf{procLeftInReq}$ on $\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$—again administrative steps—these two requests in comparison to $\overline{p_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$ do not influence the state of T' modulo $\overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$.

What remains as difference of T' and $[\![\, S'' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ is the fact that in T' the encoding of $\{\, z/x \,\} P$ appears within a branch of the replicated input encoding whereas in $[\![\, S'' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ it appears as right branch of a parallel operator encoding, i.e., it remains to show that $T'' \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\, y^{\star}(x).P \mid \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$, where

$$
\begin{aligned}
T'' = (\nu l, r, c_{r1}, c_{r2}, r_o, r_i)\, \big(\\
& r^{\star}(-,-,l_s,s,\varphi^{\mathrm{m}}_{\mathrm{a}}(x)).\mathsf{test}\ l_s\ \mathsf{then}\ \overline{l_s}\langle\bot\rangle \mid \overline{s} \mid \overline{c_{r1}}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(x)\rangle\ \mathsf{else}\ \overline{l_s}\langle\bot\rangle \\
& \mid \overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle \mid \overline{l}\langle\top\rangle \\
& \mid c_{r1}{}^{\star}(\varphi^{\mathrm{m}}_{\mathrm{a}}(x)).c_{r2}(r_o, r_i). \\
& \quad (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\, (\mathsf{pushReqIn} \\
& \qquad \mid (\nu p_o, p_i)\, ([\![\, P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \qquad \mid (\nu r_o, r_i)\, (\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,) \\
& \mid (\nu m_o, m_i, p_{o,up}, p_{i,up}, r_{o,up}, r_{i,up}, c_o, c_i, m_{o,up}, m_{i,up})\, (\mathsf{pushReqIn} \\
& \qquad \mid (\nu p_o, p_i)\, ([\![\, \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \qquad \mid (\nu r_o, r_i)\, (\overline{c_{r2}}\langle r_o \mid r_i\rangle \mid \mathsf{pushReqOut})\,)\,)
\end{aligned}
$$

Note that the term

$$(\nu p_o, p_i)\, ([\![\, \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq})$$

exactly corresponds to the right branch of $[\![\, y^{\star}(x).P \mid \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. If we compare pushReqIn with procLeftOutReq and procLeftInReq, we observe that the former includes exactly the same forwarders as the latter but also some additional forwarders. The same holds for pushReqOut and pushReq. Note that the additional forwarders ensure that each request of each branch of the replicated input encoding is forwarded to each next right branch, and so these additional forwarders are necessary in case there is more than one branch. Also note that the given forwarders guarantee that each pair of requests, such that one is an input and the other one an output request and both requests do not origin from the same sum, can be combined. Moreover, note that the only request from the left side, i.e., of the encoding of the replicated input, is transmitted to the right side, i.e., the only branch of the replicated input, by the request $\overline{r_i}\langle\varphi^{\mathrm{m}}_{\mathrm{a}}(y), l, r\rangle$ and pushReqIn. So these forwarders do not distinguish T' and $[\![\, S'' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ modulo $\overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}}$.

Since T'' and $[\![\, y^{\star}(x).P \mid \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$ do only differ by the forwarding of requests but nevertheless allow for the same combinations, we deduce that $T'' \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\, y^{\star}(x).P \mid \{\, z/x \,\} P \,]\!]^{\mathrm{m}}_{\mathrm{a}}$. Thus, $T \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\, S' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$.

Induction Hypothesis: $S_1 \longmapsto S_1'$ implies $\exists T_1 \in \mathcal{P}_{\mathrm{a}}\ .\ [\![\, S_1 \,]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1 \wedge T_1 \overset{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![\, S_1' \,]\!]^{\mathrm{m}}_{\mathrm{a}}$

Induction Step: We have to consider the remaining reduction rules of π_{m} in Figure 2.3.

Case of Pi-Par$_{\mathrm{m,s,a,p}}$ **:** Then $S = S_1 \mid S_2$ for some $S_1, S_2 \in \mathcal{P}_{\mathrm{m}}$, $S_1 \longmapsto S_1'$, and $S' = S_1' \mid S_2$. By the induction hypothesis there is some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1$ and $T_1 \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}}$. The corresponding encodings are given by the following terms:

$$\begin{aligned}
[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} = \; & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}) \big(\\
& (\nu p_o, p_i) \, ([\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}) \\
& \mid (\nu p_o, p_i) \, ([\![S_2]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \mid \mathsf{pushReq} \big) \\
[\![S']\!]^{\mathrm{m}}_{\mathrm{a}} = \; & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}) \big(\\
& (\nu p_o, p_i) \, \big([\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}\big) \\
& \mid (\nu p_o, p_i) \, ([\![S_2]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \mid \mathsf{pushReq} \big)
\end{aligned}$$

Since $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1$ and since $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ is not guarded in $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}}$, we can use the rules Pi-Par$_{\mathrm{m,s,a,p}}$, Pi-Res$_{\mathrm{m,s,a,p}}$, and Pi-Cong$_{\mathrm{m,s,a,p}}$ in the asynchronous calculus to show that:

$$\begin{aligned}
[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto \; & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}) \big(\\
& (\nu p_o, p_i) \, (T_1 \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}) \\
& \mid (\nu p_o, p_i) \, ([\![S_2]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \mid \mathsf{pushReq} \big) = T
\end{aligned}$$

By Definition 6.3.37, $T_1 \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}}$ implies $\mathcal{C}(T_1) \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3} \mathcal{C}\big([\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}}\big)$ for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![P]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}^{=}\!\uparrow_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{m}}$. Since $[\![P \mid S_2]\!]^{\mathrm{m}}_{\mathrm{a}} \in \mathcal{P}_{\mathrm{a}}^{=}\!\uparrow_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$, the quantification over $\mathcal{C}$ includes all contexts $\mathcal{C}$ such that:

$$\begin{aligned}
\mathcal{C}([\cdot]) = \mathcal{C}'\big(\; & (\nu m_o, m_i, p_{o,up}, p_{i,up}, c_o, c_i, m_{o,up}, m_{i,up}) \big(\\
& (\nu p_o, p_i) \, ([\cdot] \mid \mathsf{procLeftOutReq} \mid \mathsf{procLeftInReq}) \\
& \mid (\nu p_o, p_i) \, ([\![S_2]\!]^{\mathrm{m}}_{\mathrm{a}} \mid \mathsf{procRightOutReq} \mid \mathsf{procRightInReq}) \\
& \mid \mathsf{pushReq} \big) \big) \\
= \mathcal{C}'\Big(\mathcal{C}''([\cdot])\Big)
\end{aligned}$$

Because of that, we have $\mathcal{C}'\Big(\mathcal{C}''(T_1)\Big) \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3} \mathcal{C}'\Big(\mathcal{C}''([\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}})\Big)$ for all contexts $\mathcal{C}'([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}'([\![P]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}_{\mathrm{a}}^{=}\!\uparrow_{[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{m}}$. By Definition 6.3.37, we conclude $T \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of Pi-Res$_{\mathrm{m,s,a,p}}$ **:** Then $S = (\nu x)\, S_1$ for some $x \in \mathcal{N}$ and some $S_1 \in \mathcal{P}_{\mathrm{m}}$, $S_1 \longmapsto S_1'$, and $S' = (\nu x)\, S_1'$. By the induction hypothesis, there is some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1$ and $T_1 \stackrel{\cdot}{\leftrightarrows}^{\downarrow 3}_{\mathrm{c}} [\![S_1']\!]^{\mathrm{m}}_{\mathrm{a}}$. Since $[\![\cdot]\!]^{\mathrm{m}}_{\mathrm{a}}$

translates restriction homomorphically, i.e., $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S']\!]^{\mathrm{m}}_{\mathrm{a}} = (\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,[\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}$, we can apply rule Pi-Res$_{\mathrm{m,s,a,p}}$ to conclude from $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1$ that $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto (\nu\varphi^{\mathrm{s}}_{\mathrm{a}}(x))\,T_1 = T$. By Definition 6.3.37, $T_1 \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ implies $\mathcal{C}(T_1) \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} \mathcal{C}\big([\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}\big)$ for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![P]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{m}}$. Since $(\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,[\![P]\!]^{\mathrm{m}}_{\mathrm{a}} \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}$, the quantification over $\mathcal{C}$ includes all contexts $\mathcal{C}$ such that $\mathcal{C}([\cdot]) = \mathcal{C}'((\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,[\cdot])$. Because of that, $\mathcal{C}'((\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,T_1) \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} \mathcal{C}'\big((\nu\varphi^{\mathrm{m}}_{\mathrm{a}}(x))\,[\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}\big)$ for all contexts $\mathcal{C}'([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}'([\![P]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{m}}$. By Definition 6.3.37, we conclude $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$.

Case of Pi-Cong$_{\mathrm{m,s,a,p}}$ **:** Then $S \equiv S_1$ for some $S_1 \in \mathcal{P}_{\mathrm{m}}$, $S_1 \longmapsto S'_1$, and $S'_1 \equiv S'$. By Lemma 6.3.42, the encoding $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}$ preserves structural congruence of source terms modulo $\mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}}$. So $S \equiv S_1$ and $S'_1 \equiv S'$ imply $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$ and $[\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$. By Definition 6.3.37, for all contexts $\mathcal{C}([\cdot]) \in \mathcal{P}_{\mathrm{a}} \to \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}([\![P]\!]^{\mathrm{m}}_{\mathrm{a}}) \in \mathcal{P}^{=}_{\mathrm{a}}{\restriction}_{[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{a}}}$ for all $P \in \mathcal{P}_{\mathrm{m}}$, we have $\mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} \mathcal{C}([\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}})$ and, hence, $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} [\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}}$. Thus, by Definition 6.3.32, for each sequence $[\![S]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T$ there is a sequence $[\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}} \Longmapsto T_1$ for some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} T_1$. The same holds for all contexts $\mathcal{C}$, i.e., since $\mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} \mathcal{C}([\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}})$, for each sequence $\mathcal{C}([\![S]\!]^{\mathrm{m}}_{\mathrm{a}}) \Longmapsto \mathcal{C}(T)$ there is a sequence $\mathcal{C}([\![S_1]\!]^{\mathrm{m}}_{\mathrm{a}}) \Longmapsto \mathcal{C}(T_1)$ for some $T_1 \in \mathcal{P}_{\mathrm{a}}$ such that $\mathcal{C}(T) \mathrel{\dot{\leftrightarrows}}^{\downarrow 3} \mathcal{C}(T_1)$. So, by Definition 6.3.37, $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} T_1$. By the induction hypothesis, $T_1 \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}$. Then $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} T_1$, $T_1 \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}}$, and $[\![S'_1]\!]^{\mathrm{m}}_{\mathrm{a}} \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$ imply $T \mathrel{\dot{\leftrightarrows}}^{\downarrow 3}_{\mathrm{c}} [\![S']\!]^{\mathrm{m}}_{\mathrm{a}}$.

The argumentation for $[\![\,\cdot\,]\!]^{\mathrm{m}}_{\mathrm{p}}$ is similar. □