On the distributability of mobile ambients

Kirstin Peters, Uwe Nestmann

Angaben zur Veröffentlichung / Publication details:

Peters, Kirstin, and Uwe Nestmann. 2018. "On the distributability of mobile ambients." *Electronic Proceedings in Theoretical Computer Science* 276: 104–21. https://doi.org/10.4204/eptcs.276.9.

On the Distributability of Mobile Ambients

Kirstin Peters
TU Berlin

Uwe Nestmann
TU Berlin

Modern society is dependent on distributed software systems and to verify them different modelling languages such as mobile ambients were developed. To analyse the quality of mobile ambients as a good foundational model for distributed computation, we analyse the level of synchronisation between distributed components that they can express. Therefore, we rely on earlier established synchronisation patterns. It turns out that mobile ambients are not fully distributed, because they can express enough synchronisation to express a synchronisation pattern called M. However, they can express strictly less synchronisation than the standard pi-calculus. For this reason, we can show that there is no good and distributability-preserving encoding from the standard pi-calculus into mobile ambients and also no such encoding from mobile ambients into the join-calculus, i.e., the expressive power of mobile ambients is in between these languages. Finally, we discuss how these results can be used to obtain a fully distributed variant of mobile ambients.

1 Introduction

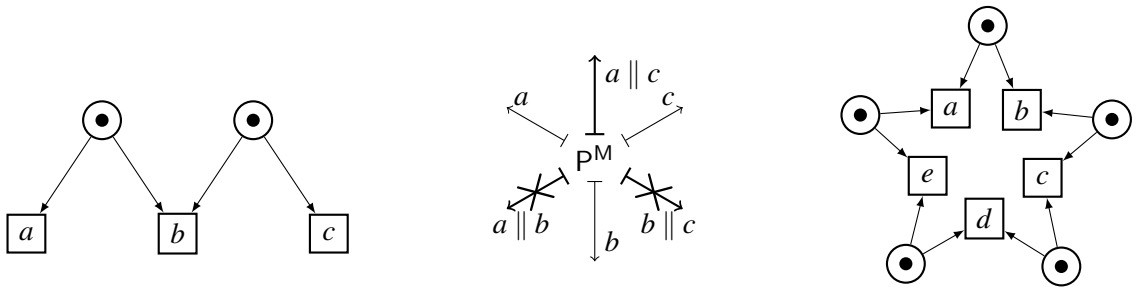
Modern society is increasingly dependent on large-scale software systems that are distributed, collaborative, and communication-centred. Most of the existing approaches that analyse the distributability of concurrent systems use special formalisms often equipped with an explicit notion of location, e.g. [2] in Petri nets or the distributed pi-calculus [14]. Other approaches implement locations implicitly, as e.g. the parallel operator in the pi-calculus that combines different distributed components of a system. In the latter case, we consider *distributability* and, thus, all possible explicitly-located variants of a calculus.

The pi-calculus [18] is a well-known and frequently used process calculus to model concurrent systems. Therein, intuitively, the *degree of distributability* corresponds to the number of parallel components that can act independently. Practical experience, though, 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 e.g. the join-calculus [17] or the distributed pi-calculus [14] were introduced as models of distributed computation.

To analyse the quality of an approach as a good foundational model for distributed computation, we compare the expressiveness of different such models w.r.t. to their power to express synchronisation between distributed components. Such synchronisations make the implementation of terms in an asynchronous setting difficult and, thus, indicate languages that are not suitable to describe distributed computation. In particular, we try to identify hidden sources of synchronisation, i.e., synchronisation that was not intended with the design of the calculus.

Distributability and Synchronisation Patterns. To analyse the degree of distribution in process calculi and to compare different calculi by their power to express synchronisation, [22, 20] defines a criterion for the preservation of distributability in encodings and introduces synchronisation patterns to describe minimal forms of synchronisation. Process calculi are then separated by their power to express such synchronisation patterns and, thus, by the kinds of synchronisation that they contain. Therefore, we show that no good and distributability-preserving encoding can exist from a calculus with enough synchronisation to express some synchronisation pattern into a calculus that cannot express this pattern. In

J.A. Pérez and S. Tini (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2018).
EPTCS 276, 2018, pp. 104–121, doi:10.4204/EPTCS.276.9



(a) (b) (c)

Figure 1: A fully reachable pure M in Petri nets (a), the M as state in a transition system (b), and the synchronisation pattern $\star$ in Petri nets (c).

this sense, synchronisation patterns have two purposes: (1) First, they describe some particular form or level of synchronisation in an abstract and model-independent way. Thereby, they help to spot forms of synchronisation—in particular, forms of synchronisation that were not intended with the design of the respective calculus. (2) Second, they allow to separate calculi along their ability to express the respective pattern and the respective level of synchronisation.

In [22], two synchronisation patterns, the pattern M and the pattern $\star$, are highlighted. An M, as visualised in Figure 1 (a), describes a Petri net that consists of two parallel transitions (a and c) and one transition (b) 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 to perform a single transition together. [11, 12] states that a Petri net specification can be implemented in an asynchronous, fully distributed setting iff 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 conditions on a state P^{M} in a step transition system, as visualized in Figure 1 (b), which allows us to directly use this pattern to reason about process calculi. Note that a, b, and c in Figure 1 (b) 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. Hence, a process calculus is distributable iff it does not contain a non-local M. A $\star$ is a chain of conflicting and distributable steps as they occur in an M that build a circle of odd length. The Figure 1 (c) nicely illustrates this circle of M. There is e.g. one M consisting of the transitions a, b, and c with their corresponding two places. Another M is build by the transitions b, c, and d with their corresponding two places and so on.

These patterns are then used to locate various π-like calculi within a hierarchy with respect to the level of synchronisation that can be expressed in these languages. More precisely, [22] shows that (1) the join-calculus is distributed, because it does not contain either of the two synchronisation patterns, (2) the asynchronous pi-calculus and its extension with separate choice can express the pattern M but no pattern $\star$, whereas the standard pi-calculus with mixed choice contains M's and $\star$'s.

Mobile Ambients. In the current paper, we use the technique derived in [22] to analyse the degree of distribution in mobile ambients. Mobile ambients were introduced in [4, 5]. Similar to the join-calculus, mobile ambients were designed as a calculus for distributed systems. But, in contrast to the join-calculus, they do contain the pattern M, as we show in the following. Accordingly, mobile ambients are not fully distributed and their implementation in a fully distributed setting is difficult. Fortunately, the little amount of synchronisation that is contained in mobile ambients is not enough to express the $\star$. Thus, mobile ambients are less synchronous than, e.g., the standard pi-calculus. Moreover, the nature of the

pattern M that we find in mobile ambients tells us what kind of features lead to synchronisation in mobile ambients. More precisely, we show that synchronisation in mobile ambients results from the so-called open-actions and the fact that different ambients may share the same name. This observation allows us to discuss ways to obtain a variant of mobile ambients that is free of hidden synchronisations and can, thus, be implemented easily in a distributed setting.

Overview. Section 2 introduces process calculi (§ 2.1), mobile ambients (§ 2.2), encodings (§ 2.3), and synchronisation patterns together with some results of [22] (§ 2.4) that are necessary for this paper. In Section 3, we show that mobile ambients can express enough synchronisation to contain pattern M and that this implies that there is no good and distributability-preserving encoding from mobile ambients into the join-calculus. Section 4 analyses the nature of conflicts in mobile ambients that limits the forms of synchronisation they can express. It is shown that mobile ambients do not contain $\star$-patterns; this separates them from the standard pi-calculus. The observations on the nature of synchronisation in mobile ambients is then used in Section 5 to discuss ways to obtain a distributed variant of mobile ambients. We conclude with Section 6. The missing proofs can be found in [21].

2 Technical Preliminaries

We start with some general observations on process calculi and the relevant notions that we need for the comparison of process calculi as described in [22]. Then we describe the calculus of mobile ambients as introduced in [4, 5] and "good" encodings as defined in [13]. Finally, we shortly revise the results of [22] that are relevant for our analysis of mobile ambients.

2.1 Process Calculi

A *process calculus* is a language $\mathscr{L} = \langle\, \mathscr{P}, \longmapsto \,\rangle$ that consists of a set of process terms $\mathscr{P}$ (its syntax) and a relation $\longmapsto : \mathscr{P} \times \mathscr{P}$ on process terms (its reduction semantics). We often refer to process terms also simply as processes or as terms and use upper case letters $P, Q, R, \ldots, P', P_1, \ldots$ to range over them.

Assume a countably-infinite set $\mathscr{N}$, whose elements are called *names*. We use lower case letters $a, b, c, \ldots, a', a_1, \ldots$ to range over names. Let $\tau \notin \mathscr{N}$. The *syntax* of a process calculus is usually defined by a context-free grammar defining operators, i.e., functions $\mathrm{op} : \mathscr{N}^n \times \mathscr{P}^m \to \mathscr{P}$. An operator of arity 0, i.e., $m = 0$, is a *constant*. The arguments that are again process terms are called *subterms* of P.

Definition 2.1 (Subterms)**.** Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathscr{P}$. The set of *subterms* of $P = \mathrm{op}\,(x_1, \ldots, x_n, P_1, \ldots, P_m)$ is defined recursively as $\{\, P \,\} \cup \{\, P' \mid \exists i \in \{\, 1, \ldots, m \,\}\,.\ P' \text{ is a subterm of } P_i \,\}$.

With Definition 2.1, every term is a subterm of itself and constants have no further subterms. We add the special constant ✓ to each process calculus. Its purpose is to denote *success* (or *successful termination*) which allows us to compare the abstract behaviour of terms in different process calculi as described in Section 2.3. Therefore, we require that each language defines a predicate $P{\downarrow_{\checkmark}}$ that holds if the term P is successful (or has terminated successfully). Usually, this predicate holds if P contains an occurrence of ✓ that is *unguarded* (see mobile ambients below).

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 interaction between two processes or with an unknown and, hence, potentially untrusted environment. Names whose scope is restricted such that they cannot be used beyond their scope are called *bound names*. The remaining names are called *free names*. As ususal, we define three sets of names occurring in a process term: the set $\mathsf{n}(P)$ of all of P's

names, and its subsets $\mathsf{fn}(P)$ of free names and $\mathsf{bn}(P)$ of bound names. In the case of bound names, their syntactical representation as lower case letters serves as a place holder for any fresh name, i.e., any name that does not occur elsewhere in the term. To avoid confusion between free and bound names or different bound names, bound names can be replaced with fresh names by *α-conversion*. We write $P \equiv_\alpha Q$ if P and Q differ only by α-conversion.

We assume that the *semantics* is given as an *operational semantics* consisting of inference rules defined on the operators of the language [24]. For many process calculi, the semantics is provided in two forms, as *reduction semantics* and as *labelled transition semantics*. We assume that at least the reduction semantics $\longmapsto$ is given as part of the definition, because its treatment is easier in the context of encodings. A single application of the reduction semantics is called a *(reduction) step* and is written as $P \longmapsto P'$. If $P \longmapsto P'$, then P' is called *derivative* of P. Let $P \longmapsto$ (or $P \not\longmapsto$) denote the existence (absence) of a step from P, and let $\Longmapsto$ denote the reflexive and transitive closure of $\longmapsto$. A sequence of reduction steps is called a *reduction*. We write $P \longmapsto^\omega$ if P has an infinite sequence of steps and call P *convergent* if $\neg(P \longmapsto^\omega)$. 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, i.e., that is either infinite or of the form $P \Longmapsto P' \not\longmapsto$.

We extend the predicate $P\downarrow_\surd$ to reachability of success. A term $P \in \mathscr{P}$ reaches success, written as $P\Downarrow_\surd$, if it reaches a derivative that is successful, i.e., $P\Downarrow_\surd \triangleq \exists P'.\, P \Longmapsto P' \wedge P'\downarrow_\surd$. We write $P\Downarrow_{\surd!}$, if P reaches success in every finite maximal execution.

To reason about environments of terms, we use functions on process terms called contexts. More precisely, a *context* $\mathscr{C}([\cdot]_1,\ldots,[\cdot]_n) : \mathscr{P}^n \to \mathscr{P}$ with n holes is a function from n terms into a term, i.e., given $P_1,\ldots,P_n \in \mathscr{P}$, the term $\mathscr{C}(P_1,\ldots,P_n)$ is the result of inserting $P_1,\ldots,P_n$ in the corresponding order into the n holes of $\mathscr{C}$.

We assume the calculi π_{m} for the standard pi-calculus (with mixed choice) as defined in [18] and its subcalculi the pi-calculus with only separate choice (π_{s}), i.e., there all parts of the same choice construct are either all guarded by an input or all guarded by an output prefix, and the asynchronous pi-calculus (π_{a}) as introduced in [3, 15]. Moreover, we assume the join-calculus (J) as introduced in [8].

Definition 2.2 (Syntax, [22])**.** The sets of process terms are given by

$$\begin{array}{lll}
\mathscr{P}_{\mathrm{m}} ::= & P_1 \mid P_2 \;\mid\; \surd \;\mid\; (\nu n)P \;\mid\; !P \;\mid\; \sum_{i\in I}\pi_i.P_i & \\
& \pi ::= \overline{y}\langle z\rangle \;\mid\; y(x) \;\mid\; \tau & \\
\mathscr{P}_{\mathrm{s}} ::= & P_1 \mid P_2 \;\mid\; \surd \;\mid\; (\nu n)P \;\mid\; !P \;\mid\; \sum_{i\in I}\pi_i^O.P_i \;\mid\; \sum_{i\in I}\pi_i^I.P_i & \\
& \pi^O ::= \overline{y}\langle z\rangle \;\mid\; \tau \quad \text{and} \quad \pi^I ::= y(x) \;\mid\; \tau & \\
\mathscr{P}_{\mathrm{a}} ::= & 0 \;\mid\; P_1 \mid P_2 \;\mid\; \surd \;\mid\; (\nu n)P \;\mid\; !P \;\mid\; \overline{y}\langle z\rangle \;\mid\; y(x).P \;\mid\; \tau.P & \\
\mathscr{P}_{\mathrm{J}} ::= & 0 \;\mid\; P_1 \mid P_2 \;\mid\; \surd \;\mid\; y\langle z\rangle \;\mid\; \mathsf{def}\, D\, \mathsf{in}\, P & \\
& J ::= y(x) \;\mid\; J_1 \mid J_2 \quad \text{and} \quad D ::= J \triangleright P \;\mid\; D_1 \wedge D_2 &
\end{array}$$

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

In all languages the *empty process* is denoted by 0 and $P_1 \mid P_2$ defines *parallel composition*. Within the pi-calculi *restriction* $(\nu n)P$ restricts the scope of the name n to the definition of P and $!P$ denotes *replication*. 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 0 abbreviates the empty sum, i.e., where $I = \emptyset$. 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. The choice operators of π_{m} and π_{s} require that all branches of a choice are guarded by one of these prefixes. We omit the match prefix, because it does not influence the results.

In $\mathscr{P}_\mathrm{J}$ the operator $y\langle z\rangle$ describes an output prefix similar to $\mathscr{P}_\mathrm{a}$. 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 $|$, and P is a process. Note that $\mathsf{def}\,D\,\mathsf{in}\,P$ unifies the concepts of restriction, input prefix, and replication of the pi-calculus.

As usual, the continuation 0 is often omitted, so e.g. $y(x).0$ 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, e.g. $y(x).0$ is written as $y.0$ or just y, and $\mathsf{def}\,y(x) \triangleright 0\,\mathsf{in}\,y\langle z\rangle$ is abbreviated as $\mathsf{def}\,y \triangleright 0\,\mathsf{in}\,y$. Moreover, let $(\nu\tilde{x})P$ abbreviate the term $(\nu x_1)\ldots(\nu x_n)P$.

The definitions of free and bound names are completely standard, i.e., names are bound by restriction and as parameter of input and $\mathsf{n}(P) = \mathsf{fn}(P) \cup \mathsf{bn}(P)$ for all P. In the join-calculus the definition $\mathsf{def}\,D\,\mathsf{in}\,P$ binds for all elementary definitions $J_i \triangleright P_i$ in D and all join pattern $y_{i,j}(x_{i,j})$ in J_i the *received variables* $x_{i,j}$ in the corresponding P_i and the *defined variables* $y_{i,j}$ in P.

To compare process terms, process calculi usually come with different well-studied equivalence relations (see [10] for an overview). A special kind of equivalence with great importance to reason about processes are *congruences*, i.e., the closure of an equivalence with respect to contexts. Process calculi usually come with a special congruence $\equiv \subseteq \mathscr{P} \times \mathscr{P}$ called *structural congruence*. Its main purpose is to equate syntactically different process terms that model quasi-identical behaviour. For the above variants of the pi-calculus we have:

$$P \equiv Q \quad \text{if } P \equiv_\alpha Q \qquad P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad !P \equiv P \mid !P$$
$$(\nu n)\,0 \equiv 0 \qquad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \quad \text{if } n \notin \mathsf{fn}(P)$$

The entanglement of input prefix and restriction within the definition operator of the join-calculus limits the flexibility of relations defined by sets of equivalence equations. Instead structural congruence is given by an extension of the chemical approach in [1] by the heating and cooling rules. They operate on so-called solutions $\mathscr{R} \vdash \mathscr{M}$, where $\mathscr{R}$ and $\mathscr{M}$ are multisets. We have (1) $\vdash P \mid Q \rightleftharpoons \vdash P, Q$, (2) $D \wedge E \vdash \rightleftharpoons D, E \vdash$, and (3) $\vdash \mathsf{def}\,D\,\mathsf{in}\,P \rightleftharpoons \sigma_{dv}(D) \vdash \sigma_{dv}(P)$, where only elements—separated by commas—that participate in the rule are mentioned and σ_{dv} instantiates the defined variables in D to distinct fresh names. Then $P \equiv Q$ if P and Q differ only by applications of the $\rightleftharpoons$-rules, i.e., if $\vdash P \rightleftharpoons \vdash Q$.

The semantics of the above variants of the pi-calculus is given by the axioms

$$(\ldots + \tau.P + \ldots) \longmapsto P \qquad (\ldots + y(x).P + \ldots) \mid (\ldots + \overline{y}\langle z\rangle.Q + \ldots) \longmapsto \{z/x\}P \mid Q$$

for π_m and π_s, the axioms $\tau.P \longmapsto P$ and $y(x).P \mid \overline{y}\langle z\rangle \longmapsto \{z/x\}P$ for π_a, and the three rules

$$\frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \frac{P \longmapsto P'}{(\nu n)P \longmapsto (\nu n)P'} \qquad \frac{P \equiv Q \qquad Q \longmapsto Q' \qquad Q' \equiv P'}{P \longmapsto P'}$$

that hold for all three variants π_m, π_s, and π_a. The operational semantics of J is given by the heating and cooling rules (see structural congruence) and the reduction rule $J \triangleright P \vdash \sigma_{rv}(J) \longmapsto J \triangleright P \vdash \sigma_{rv}(P)$, where σ_{rv} substitutes the transmitted names for the distinct received variables.

Recursion or replication distinguishes itself from other operators by the fact that (one of) its subterms can be copied within rules of structural congruence in the pi-calculus or by reduction rules in the join-calculus while the operator itself is usually never removed during reductions. We call such operators and capabilities *recurrent*. We denote the parts of a term that are removed in reduction steps as *capabilities*.

2.2 Mobile Ambients

Mobile ambients (MA) were introduced in [4, 5] as a process calculus for distributed systems with mobile computations. They define *ambients* as bounded places on that computations may happen and that can be moved (with their computations). Their syntax is defined in two stages: the first stage describes *ambient processes* and the nesting of ambients; the second stage describes the movements of ambients.

Definition 2.3 (Syntax, [5])**.** The set of *ambient processes* $\mathscr{P}_{\mathrm{MA}}$ is given as

$$\begin{array}{rcl} \mathscr{P}_{\mathrm{MA}} ::= & 0 \mid P_1 \mid P_2 \mid \checkmark \mid (\nu n)\,P \mid !P \mid n[P] \mid M.P \\ M ::= & \mathsf{in}\,n \mid \mathsf{out}\,n \mid \mathsf{open}\,n \end{array}$$

for some names $n \in \mathscr{N}$.

The *empty process* is denoted by 0 and $P_1 \mid P_2$ define *parallel composition*. *Restriction* $(\nu n)\,P$ restricts the scope of the name n to the definition of P. *Replication* $!P$ provides potentially infinitely many copies of P. The $n[P]$ describes an *ambient* n in which the process P is located. Ambients may exhibit a tree structure induced by the nesting of ambient brackets. The term $M.P$ defines the exercise of capability M, which could be either "$\mathsf{in}\,n$" to *enter* ambient n, or "$\mathsf{out}\,n$" to *exit* from ambient n, or "$\mathsf{open}\,n$" to *open* ambient n. As usual, the continuation 0 is often omitted. Moreover, we often abbreviate $n[0]$ by $n[\,]$ and let $(\nu\tilde{x})\,P$ abbreviate the term $(\nu x_1)\ldots(\nu x_n)\,P$.

Restriction is the only binder of mobile ambients, i.e., the names are bound by restriction and all names of a process that are not bound by restriction are free. The "." in $M.P$ denotes sequential composition, where the M guards the subterm P. A subterm of a process is unguarded if it is not hidden behind a guard. As usual, $P\downarrow_{\checkmark}$ if P contains an unguarded occurrence of success.

For mobile ambients, [5] define structural congruence as the least congruence that satisfies the rules of $\equiv$ defined above and additionally the rules $!0 \equiv 0$ and $(\nu n)\,(m[P]) \equiv m[\,(\nu n)\,P\,]$ if $n \neq m$.

The reduction semantics of mobile ambients in [5] consists of the axioms

$$n[\,\mathsf{in}\,m.P \mid Q\,] \mid m[R] \longmapsto m[\,n[\,P \mid Q\,] \mid R\,]$$

$$m[\,n[\,\mathsf{out}\,m.P \mid Q\,] \mid R\,] \longmapsto n[\,P \mid Q\,] \mid m[R] \qquad \mathsf{open}\,n.P \mid n[Q] \longmapsto P \mid Q$$

and the rules:

$$\frac{P \longmapsto P'}{(\nu n)\,P \longmapsto (\nu n)\,P'} \qquad \frac{P \longmapsto P'}{n[P] \longmapsto n[P']} \qquad \frac{P \longmapsto P'}{P \mid R \longmapsto P' \mid R} \qquad \frac{P \equiv Q \qquad Q \longmapsto Q' \qquad Q' \equiv P'}{P \longmapsto P'}$$

The first axiom moves an ambient n with all its content (except for the consumed $\mathsf{in}\,m$-capability) into a sibling ambient with name m, where it is composed in parallel to the content of m. The second axiom allows an ambient n with all its content (except for the consumed $\mathsf{out}\,m$-capability) to exit its parent ambient m. As result ambient n is placed in parallel to m. The third axiom dissolves the boundary of an ambient named n that is located at the same level as the open-capability. The next three rules propagate reduction across scopes, ambient nesting, and parallel composition. By the last rule reductions are defined modulo structural congruence.

Note that [5] explicitly states, that the same name can be used to name different ambients, i.e., ambients with separate identities. Moreover, if there are several ambients with the same name at the same hierarchical level all in and open-capabilities that affect an ambient with this name can chose freely (non-deterministically) between the alternatives.

Following [22], we denote the operator $!P$ for replication as recurrent, because (in contrast to the other operators) it is itself never removed during reductions. Similarly, we denote an ambient that is not opened or moved in a step as recurrent for this step and, otherwise, as non-recurrent w.r.t. this step. To distinguish between different occurrences of syntactically the same subterm in a term, we assume that all capabilities of processes in the following are implicitly labelled as described in [22].

2.3 Encodings and Quality Criteria

Let $\mathscr{L}_{\mathrm{S}} = \langle\, \mathscr{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \,\rangle$ and $\mathscr{L}_{\mathrm{T}} = \langle\, \mathscr{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \,\rangle$ be two process calculi, denoted as *source* and *target language*. An *encoding* from $\mathscr{L}_{\mathrm{S}}$ into $\mathscr{L}_{\mathrm{T}}$ is a function $[\![\,\cdot\,]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$. We often use $S, S', S_1, \ldots$ to

range over $\mathscr{P}_S$ and $T, T', T_1, \ldots$ to range over $\mathscr{P}_T$. 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.

To analyse the quality of encodings and to rule out trivial or meaningless encodings, they are augmented with a set of quality criteria. In order to provide a general framework, Gorla in [13] suggests five criteria well suited for language comparison. They 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*. It turns out that we do not need the second criterion to derive the separation results of this paper. Thus, we omit it. Note that a behavioural equivalence $\asymp$ on the target language is assumed for the definition of name invariance and operational correspondence. Moreover, let $\varphi : \mathscr{N} \to \mathscr{N}^k$ be a *renaming policy*, i.e., a mapping from a name to a vector of names that can be used by encodings to reserve special names, such that no two different names are translated into overlapping vectors of names.

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 [13] to be parametrised on the free names of the respective source term.

Definition 2.4 (Compositionality, [13])**.** The encoding $[\![\cdot]\!]$ is *compositional* if, for every operator $\mathbf{op} : \mathscr{N}^n \times \mathscr{P}_S^m \to \mathscr{P}_S$ of $\mathscr{L}_S$ and for every subset of names N, there exists a context $\mathscr{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathscr{N}^n \times \mathscr{P}_S^m \to \mathscr{P}_T$ such that, for all $x_1, \ldots, x_n \in \mathscr{N}$ and all $S_1, \ldots, S_m \in \mathscr{P}_S$ with $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$, it holds that $[\![\mathbf{op}(x_1, \ldots, x_n, S_1, \ldots, S_m)]\!] = \mathscr{C}^N_{\mathbf{op}}(\varphi(x_1), \ldots, \varphi(x_n), [\![S_1]\!], \ldots, [\![S_m]\!])$.

The first semantic criterion 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 2.5 (Operational Correspondence, [13])**.** The encoding $[\![\cdot]\!]$ satisfies *operational correspondence* if it satisfies:

Completeness: For all $S \Longmapsto_S S'$, it holds $[\![S]\!] \Longmapsto_T \asymp [\![S']\!]$.

Soundness: For all $[\![S]\!] \Longmapsto_T T$, there exists an S' such that $S \Longmapsto_S S'$ and $T \Longmapsto_T \asymp [\![S']\!]$.

The definition of operational correspondence relies on the equivalence $\asymp$ to get rid of junk possibly left over within computations of target terms. 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 2.6 (Divergence Reflection, [13])**.** The encoding $[\![\cdot]\!]$ *reflects divergence* if, for every source term S, $[\![S]\!] \longmapsto^{\omega}_T$ implies $S \longmapsto^{\omega}_S$.

The last criterion links the behaviour of source terms to the behaviour of their encodings. With Gorla [13], we assume a *success* operator $\checkmark$ as part of the syntax of both the source and the target language. Since $\checkmark$ cannot 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. We choose may-testing to test for the reachability of success, i.e., $P \Downarrow_{\checkmark} \triangleq \exists P'.\, P \Longmapsto P' \wedge P' \downarrow_{\checkmark}$. However, this choice is not crucial. 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 2.7 (Success Sensitiveness, [13])**.** The encoding $[\![\cdot]\!]$ is *success-sensitive* if, for every source term S, $S \Downarrow_{\checkmark}$ iff $[\![S]\!] \Downarrow_{\checkmark}$.

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 2.8 (Success Respecting, [13]). $\asymp$ is *success respecting* if, for every P and Q with $P \Downarrow_{\surd}$ and $Q \not\Downarrow_{\surd}$, it holds that $P \not\asymp Q$.

By [13] a "good" equivalence $\asymp$ is often defined in the form of a barbed equivalence (as described e.g. in [19]) or can be derived directly from the reduction semantics and is often a congruence, at least with respect to parallel composition. For the separation results presented in this paper, we require only that $\asymp$ is a success respecting reduction bisimulation.

Definition 2.9 ((Weak) Reduction Bisimulation). The equivalence $\asymp$ is a *(weak) reduction bisimulation* if, for every $T_1, T_2 \in \mathscr{P}_{\mathrm{T}}$ such that $T_1 \asymp T_2$, for all $T_1 \Longmapsto_{\mathrm{T}} T_1'$ there exists a T_2' such that $T_2 \Longmapsto_{\mathrm{T}} T_2'$ and $T_1' \asymp T_2'$.

Note that the best known encoding from the asynchronous pi-calculus into the join-calculus in [8] is not compositional, but consists of an inner, compositional encoding surrounded by a fixed context—the implementation of so-called firewalls—that is parametrised on the free names of the source term. In order to capture this and similar encodings and as done in [22] we relax the definition of compositionality in our notion of a good encoding.

Definition 2.10 (Good Encoding). We consider an encoding $[\![\cdot]\!]$ to be *good* if it is (1) either compositional or consists of an inner, compositional encoding surrounded by a fixed context that can be parametrised on the free names of the source term, (2) satisfies operational correspondence, (3) reflects divergence, and (4) is success-sensitive. Moreover we require that the equivalence $\asymp$ is a success respecting (weak) reduction bisimulation.

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

Lemma 2.11 ([23]). *For all success respecting reduction bisimulations $\asymp$ and all convergent target terms T_1, T_2 such that $T_1 \asymp T_2$, it holds $T_1 \Downarrow_{\surd!}$ iff $T_2 \Downarrow_{\surd!}$.*

Then success sensitiveness preserves the ability to reach success in all finite maximal executions.

Lemma 2.12 ([23]). *For all operationally sound, divergence reflecting, and success-sensitive encodings $[\![\cdot]\!]$ with respect to some success respecting equivalence $\asymp$ and for all convergent source terms S, if $S \Downarrow_{\surd!}$ then $[\![S]\!] \Downarrow_{\surd!}$.*

2.4 Distributability and Synchronisation Pattern

Intuitively, a distribution of a process means the extraction (or: separation) of its (sequential) components and their association to different locations. However, not all process calculi in the literature—as e.g. the standard pi-calculus in [18]—consider locations explicitly. For the calculi without an explicit notion of location [22] defines a general notion of *distributability* that focuses on the possible division 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.

Definition 2.13 (Distributability, [22]). Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus, $\equiv$ be its structural congruence, and $P \in \mathscr{P}$. P is *distributable* into $P_1, \ldots, P_n \in \mathscr{P}$ if there exists $P' \equiv P$ such that

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

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 (different from 0) 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.

Accordingly, a pi-term P is distributable into $P_1, \ldots, P_n$ if $P \equiv (\nu\tilde{a})\,(P_1 \mid \ldots \mid P_n)$. The $\mathscr{P}_{\mathrm{J}}$-term $\mathsf{def}\, a \triangleright 0 \,\mathsf{in}\, (\mathsf{def}\, b \triangleright c\langle a\rangle \,\mathsf{in}\, (a \mid b))$ is distributable into $\mathsf{def}\, a \triangleright 0 \,\mathsf{in}\, a$ and $\mathsf{def}\, b \triangleright c\langle a\rangle \,\mathsf{in}\, b$, but e.g. also into $\mathsf{def}\, a \triangleright 0 \,\mathsf{in}\, 0$, $\mathsf{def}\, b \triangleright c\langle a\rangle \,\mathsf{in}\, 0$, a, and b, because $\vdash \mathsf{def}\, a \triangleright 0 \,\mathsf{in}\, (\mathsf{def}\, b \triangleright c\langle a\rangle \,\mathsf{in}\, (a \mid b)) \rightleftharpoons \mathsf{def}\, a \,\mathsf{in}\, 0, \mathsf{def}\, b \,\mathsf{in}\, c\langle a\rangle \vdash a \mid b \rightleftharpoons \mathsf{def}\, a \,\mathsf{in}\, 0, \mathsf{def}\, b \,\mathsf{in}\, c\langle a\rangle \vdash a, b \rightleftharpoons \vdash \mathsf{def}\, a \triangleright 0 \,\mathsf{in}\, 0, \mathsf{def}\, b \triangleright c\langle a\rangle \,\mathsf{in}\, 0, a, b$.

Mobile ambients come with an explicit notion of locations: ambients. A term of $\mathscr{P}_{\mathrm{MA}}$ is distributable into pairwise intersected subsets of its outermost ambients. Applying the Definition 2.13 results into exactly these distributable components. Because of the rule $!P \equiv P \mid !P$, the replication of an ambient, e.g. by $!(n[P])$ or $!((\nu n)\, n[P])$, is a distributable recurrent operation.

Preservation of distributability means that the target term is at least as distributable as the source term.

Definition 2.14 (Preservation of Distributability, [22])**.** An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *preserves distributability* if for every $S \in \mathscr{P}_{\mathrm{S}}$ and for all terms $S_1, \ldots, S_n \in \mathscr{P}_{\mathrm{S}}$ that are distributable within S there are some $T_1, \ldots, T_n \in \mathscr{P}_{\mathrm{T}}$ 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. It respects both the intuition on distribution as separation on different locations—an 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—implemented by $T_i \asymp [\![S_i]\!]$.

If a single process—of an arbitrary process calculus—can perform two different 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 2.15 (Distributable Steps, [22])**.** Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus and $P \in \mathscr{P}$ a process. Two alternative steps of P are in *conflict*, if performing one step disables the other step, i.e., if both reduce the same not recurrent capability. Otherwise they are *parallel*. Two parallel steps of P are *distributable*, if each recurrent capability reduced by both steps is distributable, else the steps are *local*.

Remember that the "same" means "with the same label", i.e., in $(\mathsf{open}\, n \mid n[P_1] \mid n[P_2])$ the two steps that open one of the ambients n are in conflict but $(\mathsf{open}\, n \mid n[P_1] \mid \mathsf{open}\, n \mid n[P_2])$ can perform two parallel steps—using different open-capabilities and ambients—to open both ambients n.

Next we define parallel and distributable sequences of steps.

Definition 2.16 (Distributable Executions, [22])**.** Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus, $P \in \mathscr{P}$, 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.

Two executions of a term P are distributable iff P is distributable into two subterms such that each performs one of these executions. Hence, an operationally complete encoding is distributability-preserving only if it preserves the distributability of sequences of source term steps.

Lemma 2.17 (Distributability-Preservation, [22])**.** *An operationally complete encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ that preserves distributability also preserves distributability of executions, i.e., for all source terms $S \in \mathscr{P}_{\mathrm{S}}$ 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 $[\![S]\!]$.*

As described in the introduction, we consider a process calculus is distributable iff it does not contain a non-local M.

Definition 2.18 (Synchronisation Pattern M, [22])**.** Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus and $\mathsf{P}^{\mathsf{M}} \in \mathscr{P}$ such that:

1. P^{M} can perform at least three alternative steps a: $\mathsf{P}^{\mathsf{M}} \longmapsto P_a$, b: $\mathsf{P}^{\mathsf{M}} \longmapsto P_b$, and c: $\mathsf{P}^{\mathsf{M}} \longmapsto P_c$ such that P_a, P_b, and P_c are pairwise different.
2. The steps a and c are parallel in P^{M}.
3. But b is in conflict with both a and c.

In this case, we denote the process P^{M} as M. If the steps a and c are distributable in P^{M}, then we call the M *non-local*. Otherwise, the M is called *local*.

As shown in [22], all M in the join-calculus (J) are local but the asynchronous pi-calculus (π_{a}) contains the non-local M: $\overline{y}\langle u\rangle \mid y(x).P_1 \mid \overline{y}\langle v\rangle \mid y(x).P_2$ with $P_1, P_2 \in \mathscr{P}_{\mathrm{a}}$, where the steps a, b, and c are the reduction of the first out- and input, the first input and the second output, and the second out- and input, respectively. Because of that, there is no good and distributability-preserving encoding from π_{a} into J. To further distinguish different variants of the pi-calculus, [22] introduces a second synchronisation pattern called $\star$. Interestingly, it reflects a well-known standard problem in the area of distributed systems, namely the problem of the dining philosophers [7].

Definition 2.19 (Synchronisation Pattern $\star$, [22])**.** Let $\langle\, \mathscr{P}, \longmapsto \,\rangle$ be a process calculus and $\mathsf{P}^{\star} \in \mathscr{P}$ such that:

1. $\mathsf{P}^{\star}$ can perform at least five alternative reduction steps $i : \mathsf{P}^{\star} \longmapsto P_i$ for $i \in \{a,b,c,d,e\}$ such that the P_i are pairwise different.
2. 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. Finally,
3. every pair of steps in $\{a,b,c,d,e\}$ that is not in conflict due to the previous condition is parallel in $\mathsf{P}^{\star}$.

In this case, we denote the process $\mathsf{P}^{\star}$ as $\star$. The synchronisation pattern $\star$ is visualised by the Petri net in Figure 1 (c). If all pairs of parallel steps in $\{a,b,c,d,e\}$ are distributable in $\mathsf{P}^{\star}$, then we call the $\star$ *non-local*. Otherwise, it is called *local*.

Note that we need at least four steps in this cycle, to have two steps that are distributable, and a cycle of odd degree to distinguish different variants of the pi-calculus. Accordingly, the $\star$ is the smallest structure with these requirements. To see the connection with the dining philosophers problem, consider the places in Figure 1 (c) 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.

[22] then shows that the asynchronous pi-calculus (π_{a}) and also the pi-calculus with separate choice (π_{s}) do not contain the pattern $\star$, whereas the standard pi-calculus (π_{m}) with mixed choice has $\star$.

Example 2.20 (Non-Local $\star$ in π_{m})**.** Consider a term $\mathsf{S}^{\star}_{\mathrm{m}}\big(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5\big) \in \mathscr{P}_{\mathrm{m}}$ such that

$$\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5) = \overline{a} + b.\mathsf{S}^{\star}_1 \mid \overline{b} + c.\mathsf{S}^{\star}_2 \mid \overline{c} + d.\mathsf{S}^{\star}_3 \mid \overline{d} + e.\mathsf{S}^{\star}_4 \mid \overline{e} + a.\mathsf{S}^{\star}_5$$

for some $\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5 \in \{0,\checkmark\}$. Then, $\mathsf{S}^{\star}_{\mathrm{m}}\big(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5\big)$ can perform the steps a, …, e, where the step $i \in \{a,\ldots,e\}$ is a communication on channel i. By Definition 2.19, $\mathsf{S}^{\star}_{\mathrm{m}}\big(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5\big)$ is a non-local $\star$.

Actually, the above term $\mathsf{S}^{\star}_{\mathrm{m}}\big(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5\big)$ is a $\star$ in CCS with mixed choice, because for this counterexample the communication of values was not relevant. Adding (unused) values to the communication prefixes is straight forward. By using the $\star$-pattern $\mathsf{S}^{\star}_{\mathrm{m}}\big(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5\big)$ as counterexample, [22, 23] shows that there is no good and distributability-preserving encoding from π_{m} into π_{s} (or π_{a}).

3 Mobile Ambients are not Distributable

Similar to the join-calculus, mobile ambients were designed in order to be distributed (or distributable), where ambients were introduced as an explicit representation of locations. But in opposite to the join-calculus there are non-local M in mobile ambients, i.e., some form of synchronisation between ambients.

Example 3.1 (Non-Local M in Mobile Ambients.)**.** Consider the $\mathscr{P}_{\mathrm{MA}}$-term

$$\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}} = (\mathsf{open}\, n_1 \mid n_1[P_1]) \mid (n_1[\mathsf{in}\, n_2.P_2] \mid n_2[P_3])$$

with $P_1, P_2, P_3 \in \mathscr{P}_{\mathrm{MA}}$. $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ can perform modulo structural congruence the steps

- a: $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto P_1 \mid (n_1[\mathsf{in}\, n_2.P_2] \mid n_2[P_3])$
- b: $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto n_1[P_1] \mid \mathsf{in}\, n_2.P_2 \mid n_2[P_3]$
- c: $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto (\mathsf{open}\, n_1 \mid n_1[P_1]) \mid (n_2[n_1[P_2] \mid P_3])$

Here, the steps a and b compete for the non-recurrent open-capability. The steps b and c compete for the right ambient n_1 that is non-recurrent in both steps. Hence, both of these pairs of steps are in conflict, while the pair of steps a and c is distributable. Thus $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ is a non-local M.

Similar to the proof, that there is no good and distributability-preserving encoding from π_{a} into J, we use this $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ as a counterexample to show that there is no good and distributability-preserving encoding from MA into J. Therefore, we instantiate the processes P_1, P_2, and P_3 such that the conflicting step b can be distinguished by success from the distributable steps a and c. We choose $P_1 = n_3[\,]$, $P_2 = \mathsf{in}\, n_3.\checkmark$, and $P_3 = \mathsf{open}\, n_1$, such that $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ reaches success iff the steps a and c are performed.

Example 3.2 (Counterexample)**.** The non-local M

$$\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} = (\mathsf{open}\, n_1 \mid n_1[n_3[\,]]) \mid (n_1[\mathsf{in}\, n_2.\mathsf{in}\, n_3.\checkmark] \mid n_2[\mathsf{open}\, n_1])$$

reaches success iff $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ performs both of the distributable steps a and c, where

- a: $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto S_a$ with $S_a = n_3[\,] \mid (n_1[\mathsf{in}\, n_2.\mathsf{in}\, n_3.\checkmark] \mid n_2[\mathsf{open}\, n_1])$ and $S_a \Downarrow_{\checkmark!}$,
- b: $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto S_b$ with $S_b = n_1[n_3[\,]] \mid \mathsf{in}\, n_2.\mathsf{in}\, n_3.\checkmark \mid n_2[\mathsf{open}\, n_1]$ and $S_b \not\Downarrow_{\checkmark}$, and
- c: $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \longmapsto S_c$ with $S_c = (\mathsf{open}\, n_1 \mid n_1[n_3[\,]]) \mid n_2[n_1[\mathsf{in}\, n_3.\checkmark] \mid \mathsf{open}\, n_1]$ and $S_c \Downarrow_{\checkmark!}$.

Any good encoding that preserves distributability has to translate $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ 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. We show that every distributability-preserving encoding has to distribute b and, afterwards, that this distribution of b violates the criteria of a good encoding.

Lemma 3.3. *Every encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{MA}} \to \mathscr{P}_{\mathrm{J}}$ that is good and distributability-preserving has to split up the conflict in $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ of b with a and c such that there exists a maximal execution in $[\![\, \mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \,]\!]$ in which a is emulated but not c, and vice versa.*

In [22] we show a similar result for all encodings from π_{a} into J (Lemma 4 in [22]) using a counterexample E1. Since the counterexample $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ in MA is in its properties very similar to the counterexample E1 of [22], the proof of Lemma 3.3 is exactly the same as the proof of Lemma 4 in [22] as presented in [23]. The main idea of this proof is as follows: Any good encoding that preserves distributability has to translate $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ such that the emulations of the steps a and c are again distributable. Moreover any good

encoding has to translate the conflicts between a and b as well as between b and c into conflicts between the respective emulations. This either leads to a non-local M again or it results into an emulation of b with at least two steps such that the conflicts with the emulation of b are emulated by two different steps. Next we show that this distribution of the conflict violates the criteria of a good encoding with respect to the considered source language, i.e., w.r.t. our counterexample $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ and an adaptation of this example.

Also the proof that there is no good and distributability-preserving encoding from MA into J is very similar to the proof for the non-existence of such an encoding from π_{a} into J in [22, 23].

Theorem 3.4. *There is no good and distributability-preserving encoding from* MA *into* J.

Proof. Assume the opposite. Then there is a good and distributability-preserving encoding of $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$. By the proof of Lemma 3.3, there is a maximal execution of $[\![\, \mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \,]\!]$ in that a but not c is emulated or vice versa. Since $S_a \Downarrow_{\checkmark !}$ and $S_c \Downarrow_{\checkmark !}$ and because of success sensitiveness, the corresponding emulation leads to success. So there is an execution such that the emulation of a leads to success without the emulation of c or vice versa. Let us assume that a but not c is emulated. The other case is similar.

For encodings with respect to the relaxed definition of compositionality in Definition 2.10, there exists a context $\mathscr{C}_{\cdot}^{[\cdot]_1,[\cdot]_2}(\mathscr{P}_{\mathrm{J}})^2 \to \mathscr{P}_{\mathrm{J}}$—the combination of the surrounding context and the context introduced by compositionality (Definition 2.4)—such that $[\![\, \mathsf{S}^{\mathsf{M}}_{\mathrm{MA}} \,]\!] = \mathscr{C}([\![\, S_1 \,]\!], [\![\, S_2 \,]\!])$, where $S_1, S_2 \in \mathscr{P}_{\mathrm{MA}}$ with $S_1 = \mathsf{open}\, n_1 \mid n_1[n_3[\,]]$ and $S_2 = n_1[\mathsf{in}\, n_2.\mathsf{in}\, n_3.\checkmark] \mid n_2[\mathsf{open}\, n_1]$. Let $S_2' = n_1[\mathsf{out}\, n_2.\mathsf{in}\, n_3.\checkmark] \mid n_2[\mathsf{open}\, n_1]$. 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., $[\![\, S_1 \mid S_2' \,]\!] = \mathscr{C}([\![\, S_1 \,]\!], [\![\, S_2' \,]\!])$. $\mathsf{S}^{\mathsf{M}}_{\mathrm{MA}}$ 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 $\mathscr{C}([\![\, S_1 \,]\!], [\![\, S_2 \,]\!])$ reaches some $T_a \Downarrow_{\checkmark !}$ without the emulation of c, then $\mathscr{C}([\![\, S_1 \,]\!], [\![\, S_2' \,]\!])$ 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. □

Note that the only differences in the proof above and the proof for the the non-existence of a good and distributability-preserving encoding from π_{a} into J in [23] are the due to the different counterexample and the corresponding choice of its adaptation with S_2'.

4 Conflicts in Mobile Ambients

Both of the above-defined synchronisation patterns rely on the notion of conflict. In mobile ambients, the same ambient can be considered as recurrent in one step, but non-recurrent in another step. This fact, i.e., the existence of operators that are recurrent for some but non-recurrent for other steps, distinguishes mobile ambients from all other calculi considered in [22] and generates a new notion of conflict.

Example 4.1 (Asymmetric Conflict)**.** Consider the mobile ambient term:

$$P = n_1[\mathsf{in}\, n_2] \mid n_2[\mathsf{in}\, n_3] \mid n_3[\,]$$

P can perform two alternative steps

- $s_1 : P \longmapsto P_1$ with $P_1 = n_2[n_1[\,] \mid \mathsf{in}\, n_3] \mid n_3[\,]$ and
- $s_2 : P \longmapsto P_2$ with $P_2 = n_1[\mathsf{in}\, n_2] \mid n_3[n_2[\,]]$

that both use the ambient n_2 (but no other operator is used in both steps). In s_1, the ambient n_2 is a recurrent capability but in s_2 the ambient n_2 is moved and, thus, is non-recurrent. Accordingly, s_2 disables s_1, i.e., $P_2 \not\longmapsto$, but not vice versa, i.e., P_1 can perform the step s_2 such that $P_1 \longmapsto n_3[n_2[n_1[\,]]]$.

Accordingly, we denote a conflict as *symmetric* if the steps compete for an operator that is non-recurrent in both, i.e., if both steps disable the respective other step, and otherwise as *asymmetric*. The example above can be extended to a cyclic structure of odd degree. The term

$$a[\mathsf{in}\, b] \mid b[\mathsf{in}\, c] \mid c[\mathsf{in}\, d] \mid d[\mathsf{in}\, e] \mid e[\mathsf{in}\, a]$$

even satisfies Definition 2.19, i.e., it describes a non-local $\star$, if we were to relax in the required conflicts in Definition 2.19 by requiring only asymmetric conflicts. However, because of the asymmetric conflicts within this structure, it can be encoded much more easily than a $\star$ with symmetric conflicts. This is also reflected by the fact that in the proofs for the separation result between π_{m} and π_{a} in [22] we have to rely on the mutually exclusive nature of the conflicts in the $\star$ of the counterexample $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$. Accordingly, we cannot use an M or a $\star$ with asymmetric conflicts to derive separation results as done above. Instead, we show that, despite of the $\star$ with asymmetric conflicts, mobile ambients can be separated from π_{m} by the synchronisation pattern $\star$, because they cannot express a $\star$ with symmetric conflicts.

It turns out that the symmetric conflict in the pattern M of the step b with a and c as given in Example 3.1 can only be expressed with an open-action.

Lemma 4.2. *Let $P \in \mathscr{P}_{\mathrm{MA}}$ be an M. Then one of the two conflicts is asymmetric or the step b reduces an* open*-action.*

Since the synchronisation pattern $\star$ consists of several cyclic overlapping M, all five steps of a $\star$ in mobile ambients have to reduce an open-capability or at least one of the conflicts is asymmetric. However, five steps on open-capabilities cannot be combined in a cycle of odd degree. Thus, in all $\star$-like structures there is at least one asymmetric conflict. But there are no $\star$ (without asymmetric conflicts) in mobile ambients.

Lemma 4.3. *For all $\star$-like structures $P \in \mathscr{P}_{\mathrm{MA}}$ one of the conflicts in P that exist according to Definition 2.19 is asymmetric.*

A $\star$ with an asymmetric conflict cannot be extended to a $\star$ that can be used as counterexample similarly to $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$ in [22, 23]. The proof to separate π_{m} from π_{s} and π_{a} in [22, 23] exploits the fact that every maximal execution of $\star$ contains exactly two distributable steps of the five alternative steps that form the $\star$. But, if we replace a conflict in the $\star$ by an asymmetric conflict, then three steps are possible in one execution.

Lemma 4.4. *All $\star$-like structures $P \in \mathscr{P}_{\mathrm{MA}}$ have an execution that executes three of the five alternative steps that exist according to Definition 2.19.*

Proof. By Lemma 4.3, all $\star$ in mobile ambients have an asymmetric conflict. Thus, whenever some $\mathsf{S}^{\star}_{\mathrm{m}}([\cdot]_a,\ldots,[\cdot]_e) : \mathscr{P}^5_{\mathrm{MA}} \to \mathscr{P}_{\mathrm{MA}}$ is such that for all $\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5 \in \{\mathsf{0},\checkmark\}$ the term $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$ is a $\star$ except for asymmetric conflicts, then there is a maximal execution of $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$ that contains three steps of the set $\{a,\ldots,e\}$: the two steps that are related by the asymmetric conflict (executing first the step that is not in conflict to the other and then the one-sided conflicting step) and the step that is in parallel to both of the former neighbouring steps. □

To show that there is no good and distributability-preserving encoding from π_{m} into MA we proceed as in [22, 23]. First, we observe that every conflict in our counterexample $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$ has to be translated into conflicts of the respective emulations in mobile ambients.

Lemma 4.5. *Any good and distributability-preserving encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{m}} \to \mathscr{P}_{\mathrm{MA}}$ has to translate the conflicts in $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1,\ldots,\mathsf{S}^{\star}_5)$ into conflicts of the corresponding emulations.*

The proof of this Lemma is exactly the same as the proof for the corresponding Lemma for encodings from π_{m} into π_{s} in [23] but using the lemmas above, because this proof relies on the encodability criteria and the abstract notion of conflicts that is the same for π_{s} and MA. Note that this proof assumes an encoding that satisfies compositionality as defined in Definition 2.4, but, as already stated in [22], it also holds in case of the relaxed version of compositionality that is used here. Then, similar to Lemma 3.3, we show that each good encoding of the counterexample requires that a conflict has to be distributed.

Lemma 4.6. *Any good and distributability-preserving encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{m}} \to \mathscr{P}_{\mathrm{MA}}$ has to split up at least one of the conflicts in $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1, \ldots, \mathsf{S}^{\star}_5)$ (or in $\mathsf{S}^{\star}_{\mathrm{m}}([\cdot]_a, \ldots, [\cdot]_e)$) such that there exists a maximal execution of $[\![\mathsf{S}^{\star}_{\mathrm{m}}([\cdot]_a, \ldots, [\cdot]_e)]\!]$ that emulates only one source term step, i.e., unguards exactly one of the five holes.*

Again, the above proof is in its main idea similar to the respective proof of the corresponding result for encodings from π_{m} into π_{s} in [23]. However, since that proof depends on the expressive power of the considered target language to reason about the properties of the counterexample, we have to adapt it to mobile ambients. Finally, we show again that this distribution of the conflict rules out the possibility of a good and distributability-preserving encoding.

Theorem 4.7. *There is no good and distributability-preserving encoding from π_{m} into* MA.

The proof of this Theorem very closely follows the proof of the corresponding Theorem for encodings from π_{m} into π_{s} in [23]. It picks the maximal execution of the translation that unguards—according to Lemma 4.6—only one hole $[\cdot]_x$ by emulating only one step x of $\mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1, \ldots, \mathsf{S}^{\star}_5)$. Then, we can choose $\mathsf{S}^{\star}_1, \ldots, \mathsf{S}^{\star}_5 \in \{0, \checkmark\}$ such that $\mathsf{S}^{\star}_x = 0 = \mathsf{S}^{\star}_y$, where y is one of the two steps that is parallel to x, and $\mathsf{S}^{\star}_z = \checkmark$ for all other cases. Accordingly, for the result S_x of the step $x : \mathsf{S}^{\star}_{\mathrm{m}}(\mathsf{S}^{\star}_1, \ldots, \mathsf{S}^{\star}_5) \longmapsto S_x$, we have $S_x \not\Downarrow_{\checkmark !}$, by doing y next, but $S_x \Downarrow_{\checkmark}$, because of success in the respective other step that can be executed after x. However, the maximal execution of $\mathsf{S}^{\star}_{\mathrm{m}}([\cdot]_a, \ldots, [\cdot]_e)$ that unguards only $[\cdot]_x$ and emulates only x cannot have the same behaviour w.r.t. success. After emulating x we reach a term that cannot offer the possibility to reach success (without the emulation of another source term step) as well as to deadlock without reaching success. This violates our requirements on good encodings.

5 Distributing Mobile Ambients

Theorem 3.4 shows that mobile ambients are not as distributable as the join-calculus. Nonetheless, [9] presents an encoding from MA into J in order to build a distributed implementation of mobile ambients in Jocaml ([6]). Let us consider what this encoding does with our counterexample $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ for the non-existence of a good and distributability-preserving encoding from MA into J. The encoding in [9] translates each ambient into a single unique join definition. Then it splits in, out, and open-actions into respective subactions that are controlled by the join definition that represents the parent ambient in the source. Therefore, to perform the emulations of the distributed steps a and c of $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$, the respective parts of the implementation first have to register their desire to do these steps with their parent join definition. Unfortunately, as each join definition is a single location, these two steps interact with the same join definition, so they cannot be considered as distributed. Accordingly, the encoding presented in [9] is not distributability-preserving in our sense, because the emulations of a and c are synchronised.

Indeed, the authors of [9] already state that the explicit control of subactions by the translation of the parent ambient introduces some form of synchronisation. However, they claim that the form of synchronisation introduced by the presented encoding is less crucial than, e.g., a centralised solution. Our results support the quality of their solution, by proving that no good and fully distributability-preserving encoding from MA into J exists. So, a bit of synchronisation is indeed necessary. But, our results also

suggest possible ways to circumvent the problems in the distribution of mobile ambients altogether by proposing small alterations of the source calculus itself in order to prevent M-patterns from the outset.

By Lemma 4.2, all M in mobile ambients rely on a conflict with an open-action that addresses two different ambients with the same name. A natural solution to circumvent this problem is to avoid different ambients with the same name. By Lemma 4.2, mobile ambients with unique ambient names cannot express the pattern M.

Corollary 5.1. *There are no* M *in mobile ambients, where all ambient names are unique.*

Without such an M as counterexample, our proof of Theorem 3.4 would no longer work. Instead, we can show that there is then no good and distributability-preserving encoding from π_a into MA, by using the example of an M in π_a of [22] as counterexample and following a similar proof strategy as for the separation result between π_a and J.

Claim 5.2. *If mobile ambients forbid for ambients with the same name, then there is no good and distributability-preserving encoding from* π_a *into* MA.

The proof of the above claim relies of the formalisation of the requirement that no two different ambients have the same name in the definition of the calculus. More precisely, we need to adapt the proof that every good and distributability-perserving encoding has to split up the conflict in the M of b with a and c to the target language MA with unique ambient names. Since there are several different ways to implement this requirement in the syntax of mobile ambients, we do not formally prove the above claim here. However, we expect that this proof would exploit the same strategy as in [23] and require only small adaptations due to the definition of the calculus.

Actually, the possibility to have different ambients with the same name was already identified as problematic in the encoding of [9]. To circumvent this problem, the encoding introduces unique identifiers for all ambients and one of the reasons for the interaction with the respective translation of the parent ambient to control the translations of ambient actions is that these translations of parent ambients keep the knowledge about the unique identifiers of their children. Thus, forbidding different ambients with the same name not only allows for completely distributed implementations of the calculus but also significantly simplifies translations that follow the strategy of [9].

To obtain strategies to implement this requirement, we can have a look at other distributed calculi with unique location names. The join-calculus ([8]) ensures the uniqueness of its locations by combining input prefixes with restriction in join definitions. Thus, every join definition, i.e., location, introduces its own name space. Interaction is limited to such restricted names with a clear and unique destination. The advantage is that the uniqueness of location names is ensured by definition; the disadvantage is that some forms of interaction—e.g. a two-way handshake—are syntactically more difficult due to these sharp restriction borders. The distributed pi-calculus ([14]) has a flat structure of locations and ensures uniqueness by the structural congruence rule $n[P] \mid n[Q] \equiv n[P \mid Q]$ that unifies different parts of a location. However, adding such a rule to mobile ambients requires a non-trivial adaptation of the semantics, because the open, in, and out-actions would need to first collect all ambient parts that are possibly dispersed over the term structure before they can proceed. Moreover, following this approach would not completely rule out different ambients with the same name but only different such ambients in the same parent ambient (or at top-level). This is, however, sufficient to ensure that there are no M.

6 Conclusions

We proved that there is no good and distributability-preserving encoding from mobile ambients (MA) into the join-calculus (J) and neither from the standard pi-calculus with mixed choice (π_m) into mobile

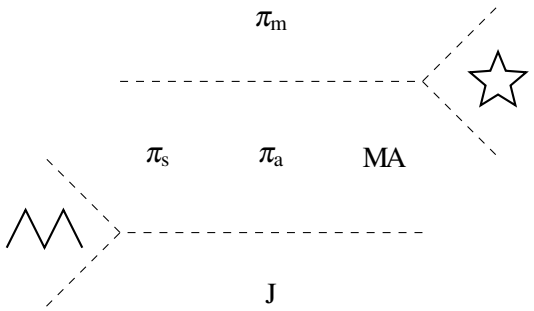


Figure 2: Distributability in Pi-like Calculi.

ambients. Note that these results stay valid also for the extension of MA with communication prefixes as described in [4, 5], because these communications are local steps that cannot be in conflict to steps with in, out, or open-actions. Thus, all conflicts added by the extension with communication primitives are local and not relevant for the preservation of distributability. Consequently, by extending the results of [22], we place mobile ambients on the same level as the pi-calculus with separate choice (π_s) and the asynchronous pi-calculus (π_a) above J and below π_m. As visualized in Figure 2, mobile ambients contain non-local M but cannot express a non-local $\star$ without asymmetric conflicts.

Asymmetric conflicts, as present in mobile ambients, constitute a variant of conflicts that turns out to be not as crucial for distributed implementations as the standard symmetric conflicts that we usually find in calculi. Nonetheless, the existence of non-local M make fully distributed implementations of mobile ambients difficult—as already observed in [9]. However, since the reason for these difficulties is now clearly captured in a simple synchronisation pattern, we can more easily derive strategies to adapt mobile ambients to a distributed calculus without such problems.

Interestingly, the extension of mobile ambients into mobile safe ambients in [16] does not solve this problem. The main idea of safe ambients is that actions require an explicit agreement on this action by both participating ambients. Therefore, safe ambients augment the respective target ambient of an action a with a matching complementary action $\overline{a}$. This extension, however does neither change the power to express the pattern M nor the asymmetric nature of conflicts with steps that do not rely on an open-action. In fact, the $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$ in mobile ambients, i.e., the pattern M, becomes

$$(\mathsf{open}\, n_1 \mid n_1[\overline{\mathsf{open}}\, n_1 \mid P_1]) \mid \big(n_1[\overline{\mathsf{open}}\, n_1 \mid \mathsf{in}\, n_2.P_2] \mid n_2\big[\overline{\mathsf{in}}\, n_1 \mid P_3\big]\big)$$

in safe ambients. This term is again an M sharing the kind of steps and properties of $\mathsf{P}^{\mathsf{M}}_{\mathrm{MA}}$. Thus, we obtain the same separation result as in Theorem 3.4 with safe ambients using the above counterexample. Moreover, since safe ambients do also not contain $\star$, also Theorem 4.7 stays valid for safe ambients.

The most obvious way to obtain a fully distributed variant of mobile ambients is to ensure uniqueness of ambient names. As a consequence, actions of mobile ambients have a clear and unique destination. Note that, having clear and unique destinations for all actions that travel location borders is also crucial for the distributability of other calculi such as the join-calculus or the distributed pi-calculus. Such unique destinations significantly limit the possibility of conflicts and ensure that all remaining conflicts of the language are local. As a consequence, distributed implementations of such languages do not need to introduce synchronisations and, thus, do not change their semantics. Hence, keeping the destinations for all actions that travel location borders unique, is a good strategy to build distributed calculi in general.

References

[1] G. Berry & G. Boudol (1990): *The Chemical Abstract Machine*. In: *Proc. of POPL*, SIGPLAN-SIGACT, pp. 81–94, doi:10.1145/96709.96717.

[2] E. Best & P. Darondeau (2011): *Petri Net Distributability*. In: *Proc. of PSI*, *LNCS* 7162.

[3] G. Boudol (1992): *Asynchrony and the π-calculus (note)*. Note, INRIA.

[4] L. Cardelli & A.D. Gordon (1998): *Mobile ambients*. In: *Proc. of FoSSaCS*, *LNCS* 1378, pp. 140–155, doi:10.1007/BFb0053547.

[5] L. Cardelli & A.D. Gordon (2000): *Mobile ambients*. *Theoretical Computer Science* 240(1), pp. 177–213, doi:10.1016/S0304-3975(99)00231-5.

[6] S. Conchon & F. Le Fessant (1999): *Jocaml: mobile agents for Objective-Caml*. In: *Proc. of ASA/MA*, IEEE, pp. 22–29, doi:10.1109/ASAMA.1999.805390.

[7] E.W. Dijkstra (1971): *Hierarchical Ordering of Sequential Processes*. *Acta Informatica* 1(2), pp. 115–138, doi:10.1007/BF00289519.

[8] C. Fournet & G. Gonthier (1996): *The Reflexive CHAM and the Join-Calculus*. In: *Proc. of POPL*, SIGPLAN-SIGACT, pp. 372–385, doi:10.1145/237721.237805.

[9] C. Fournet, J.-J. Lévy & A. Schmitt (2000): *An Asynchronous, Distributed Implementation of Mobile Ambients*. In: *Proc. of TCS*, *LNCS* 1872, pp. 348–364, doi:10.1007/3-540-44929-9_26.

[10] R. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes*. *Handbook of Process Algebra*, pp. 3–99.

[11] R. van Glabbeek, U. Goltz & J.-W. Schicke (2008): *On Synchronous and Asynchronous Interaction in Distributed Systems*. In: *Proc. of MFCS*, *LNCS* 5162, pp. 16–35, doi:10.1007/978-3-540-85238-4.

[12] R. van Glabbeek, U. Goltz & J.-W. Schicke-Uffmann (2012): *On Distributability of Petri Nets*. In: *Proc. of FoSSaCS*, *LNCS* 7213, pp. 331–345, doi:10.1007/978-3-642-28729-9_22.

[13] D. Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[14] M. Hennessy (2007): *A Distributed Pi-Calculus*. Cambridge University Press, doi:10.1017/CBO9780511611063.

[15] K. Honda & M. Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In: *Proc. of ECOOP*, *LNCS* 512, pp. 133–147, doi:10.1007/BFb0057011.

[16] F. Levi & D. Sangiorgi (2003): *Mobile Safe Ambients*. In: *Proc. of TOPLAS*, 25, ACM, pp. 1–69, doi:10.1145/596980.596981.

[17] J.-J. Lévy (1997): *Some Results in the Join-Calculus*. In: *Theoretical Aspects of Computer Software*, *LNCS* 1281, pp. 233–249, doi:10.1007/BFb0014554.

[18] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, Part I and II*. *Information and Computation* 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4, 10.1016/0890-5401(92)90009-5.

[19] R. Milner & D. Sangiorgi (1992): *Barbed Bisimulation*. In: *Proc. of ICALP*, *LNCS* 623, pp. 685–695, doi:10.1007/3-540-55719-9_114.

[20] K. Peters (2012): *Translational Expressiveness*. Ph.D. thesis, TU Berlin, doi:10.14279/depositonce-3416.

[21] K. Peters & U. Nestmann (2018): *On the Distributability of Mobile Ambients (Technical Report)*. Technical Report, TU Berlin, https://arxiv.org.

[22] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi*. In: *Proc. of ESOP*, *LNCS* 7792, pp. 310–329, doi:10.1007/978-3-642-37036-6_18.

[23] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi (Appendix)*. Technical Report, TU Berlin. `http://www.mtv.tu-berlin.de/fileadmin/a3435/pubs/distProcCal.pdf`.

[24] G.D. Plotkin (2004): *A structural approach to operational semantics*. *Journal of Logic and Algebraic Programming* 60, pp. 17–140. [An earlier version of this paper was published as technical report at Aarhus University in 1981.].