Variants of the Language Based Synthesis Problem for Petri Nets

Sebastian Mauser
Department of Applied Computer Science
Catholic University of Eichstätt-Ingolstadt, Germany
sebastian.mauser@ku-eichstaett.de

Robert Lorenz
Department of Computer Science
University of Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de

Abstract

The application of synthesis of Petri nets from languages for practical problems has recently attracted increasing attention. However, the classical synthesis problems are often not appropriate in realistic settings, because usually it is not asked for plain-vanilla Petri net synthesis, but specific additional requirements have to be considered. Having this in mind, we in this paper survey variants of the classical language based synthesis problems and develop respective solution algorithms. This yields a large repertoire of synthesis procedures presented in a uniform way.

1 Introduction

In early stages of system modelling, scenarios or use cases are often the most intuitive and appropriate modeling concept. However, for the final purposes of modelling, namely the follow-up with documentation, analysis, simulation, optimization, design or implementation of a system, usually integrated state-based system models are desired. To bridge the gap between the scenario view of a system and a final system model, automatic construction of a system model from a specification of the system behavior in terms of single scenarios is an important challenge in many application areas. In particular, in the field of software engineering the step of coming from a user oriented scenario specification to an implementation oriented state based model of a software system received much attention in the last years and offers great potential for automation [12]. Similar problems also occur in the domains of business process design (not only restricted to the well-known field of process mining [19, 4]), hardware design and controller synthesis. In all these areas a popular choice for a final system model, especially if concurrency is involved, are Petri nets and domain specific dialects of Petri nets.

In the field of Petri net theory, algorithmic construction of a Petri net model from a behavioral specification is known as synthesis [11, 2]. The classical *synthesis problem* is the problem to decide whether, for a given behavioral specification, there exists an unlabeled Petri net, such that the behavior of this net coincides with the specified behavior. In the positive case a synthesis algorithm usually constructs a witness net. Theoretical results in the literature mainly differ in the *Petri net class* and the *model for the behavioral specification* considered. Synthesis can be applied to various classes of Petri nets, including elementary nets [11], place/transition nets (p/t-nets) [2] and inhibitor nets [14]. On the one hand, the behavioral specification can be given by a transition system or by a step transition system [2], referred to as *synthesis up to isomorphism*. On the other hand, synthesis can be based on a language, the so called *synthesis up to language equivalence*. A language models scenarios of the searched net and therefore languages are the kind of specification we are interested in here. As languages, finite or infinite sets of scenarios given by occurrence sequences, step sequences [8, 1, 2] or partially ordered runs [13, 5] can be considered. In this paper we consider synthesis of a *p/t-net* from a *finite language of occurrence sequences*.

1.1 Synthesis with Regions

The theoretical basis of Petri net synthesis is the so called theory of regions. All approaches to Petri net synthesis based on regions of languages roughly follow the same idea (see e.g. [2, 13, 14]):
$\rightarrow$ Let $\mathcal{L}$ be the specified language. Instead of solving the synthesis problem (is there a net with the behavior specified by $\mathcal{L}$?) and then – in the positive case – synthesizing a witness net, a net is directly constructed from $\mathcal{L}$.
$\rightarrow$ The construction starts with the transitions T taken from the action names of $\mathcal{L}$.
$\rightarrow$ The behavior of the net is restricted by the addition of places (including their connections to transitions of the net and their initial markings).
$\rightarrow$ Places not generating dependencies which contradict the language specification $\mathcal{L}$ are candidates to be added to the net. If the behavior of a net consisting of the set of transi-

tions T and one place p includes the behavior specified by $\mathcal{L}$, then p is such candidate place. These candidate places are called *feasible w.r.t.* $\mathcal{L}$. Adding all feasible places yields the so called *saturated feasible net* $N_{\mathcal{L}}$, which includes the behavior specified by $\mathcal{L}$ and has minimal additional behavior. The language $\mathcal{L}(N_{\mathcal{L}})$ generated by $N_{\mathcal{L}}$ represents the best upper approximation to $\mathcal{L}$ by a Petri net language.
$\rightarrow$ The set of feasible places w.r.t. $\mathcal{L}$ is structurally defined by so called *regions of* $\mathcal{L}$. Each region $\mathbf{r}$ of $\mathcal{L}$ yields a corresponding feasible place $p_{\mathbf{r}}$ and each feasible place p corresponds to some region $\mathbf{r}$ of $\mathcal{L}$.
$\rightarrow$ In literature, there are several region definitions for various types of languages, e.g. [8, 1, 2, 13, 5, 14]. In all cases a region $\mathbf{r}$ of $\mathcal{L}$ can be given by a vector of integer numbers and the set of all regions is given as the set of solutions of an inequality system $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq c$.
$\rightarrow$ When all, or sufficiently many, regions $\mathbf{r}$ are identified, all places $p_{\mathbf{r}}$ of the synthesized net are constructed. The crucial point is that the set of all regions is in general infinite, whereas in most cases finite sets of regions suffice to represent $N_{\mathcal{L}}$. As discussed in [14], we distinguish two different principles of computing from the set of all regions a finite Petri net N representing $N_{\mathcal{L}}$, namely the *separation computation* and the *basis computation*. In the first case it is searched for solutions of $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq c$ which separate minimal behavior not specified in $\mathcal{L}$. In the second case a set of basis solutions of $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq c$ is computed.
$\rightarrow$ If the behavior $\mathcal{L}(N)$ of N coincides with the behavior specified by $\mathcal{L}$, then the synthesis problem has a positive answer; otherwise there is no net having the behavior specified by $\mathcal{L}$, i.e. the synthesis problem has a negative answer.

In our setting of the synthesis of a p/t-net from a finite language of occurrence sequences we consider the well established definition of p/t-net regions for languages of occurrence sequences presented e.g. in [2, 4]. In this definition the vector entries of $\mathbf{r}$ directly correspond to the initial marking of $p_{\mathbf{r}}$ and the weights of arcs connected with $p_{\mathbf{r}}$ (such regions are subsumed under the notion of transition region in [14]). Then, there are basically two procedures to solve the classical synthesis problem, one using separation representation and one using basis representation.

1.2 Synthesis Variants

In this paper, we are interested in variants of the classical synthesis problem varying the problem parameters by either relaxing or additionally restricting the requirements on the synthesized net. The problem variants cover aspects relevant for different potential applications of Petri net synthesis. We formulate six main problem variants as well as several interesting versions of these variants. For all presented variants we show solution algorithms. That means, this paper poses and solves a lot of interesting synthesis questions going beyond the classical synthesis problem. A key point is that all the stated problem variants can be solved by (mostly non-trivial) variations of the same two procedures to solve the classical problem. Thus, the paper presents a uniform treatment of the variants covering a wide range of synthesis problems. While some of the discussed problem variants are new, others have already been formulated in the literature (see the following table). But in the latter case there has usually been presented one solution algorithm for the respective variant. In contrast, in this paper we systematically consider possible solution algorithms for the problems regarding the ideas of both computation principles.

The following table (for details see the respective sections) lists the problem variants discussed in this paper. We formulate six main variants. In the ”Bound” variant a lower and an upper language bound for the behavior of the synthesized net is considered. The ”Place Bound” variant sets a bound for the number of places of the synthesized net. In the ”Identifying States” variant it is postulated that certain runs of the synthesized net lead to the same marking. In the ”Best Upper Approximation” respectively the ”Best Lower Approximation” variant it is searched for a net whose behavior is a best upper respectively lower approximation to the given language. In the ”Optimization Minimal Weights” variant, it is asked for a net with minimal arc weights and initial place markings. For most of these problem variants we also consider some slight variations, listed in the table as ”versions”. By references it is shown which problem variants have already been discussed in literature before. Thereby, we distinguish the applied computation principles (also references where a similar problem is discussed in literature possibly in some other context independently from region theory are shown in the column ”simil. prob. lit.”). The check marks indicate by which computation principles the problem variants are solved in this paper. As in the case of the classical synthesis problem, the ”Bound” variant and the ”Identifying States” variant are solved with polynomial time consumption. The runtime of the solution algorithms of the other variants may be problematic in large settings.

The paper is structured as follows: First, in Section 2, we show that the uniform discussion of variants of the classical synthesis problem adds a new dimension to a synthesis framework developed in [14]. Then, the technical part starts. The classical synthesis problem is solved with both computation principles in Section 3. In Section 4, solution algorithms for the considered variants of the classical problem are developed for separation and basis computation. For the interested reader, we provide some additional information on the topic of this paper in a technical report on our homepage (http://www.ku-eichstaett.de/Fakultaeten/MGF/Informatik/Publikationen/f_/2009ACSDVariantsTechRep.pdf). Namely, the two algorithms of Section 3 to solve the classical synthesis problem are presented in detailed pseudo-code and for the classical problem as well as for each considered problem variant an example including more detailed motivation of the problem is provided. Readers interested in certain variants can look up these examples to gain deeper insights.

Variant	separation comp.		basis comp.		simil. prob.
	lit.	here	lit.	here	lit.
Bound	-	✓	[7]	✓	-
Version Unwanted	-	✓	-	✓	-
Version Conform.	-	✓	-	✓	[16]
Place Bound	-	✓	-	-	[6, 19]
Version Minimal	-	✓	-	-	[6]
Version Sub-Net	[8]	✓	-	✓	[19]
Identifying States	-	✓	-	✓	-
Version Separ.	-	✓	-	✓	[2]
Best Upper Approx.	-	✓	[8]	✓	-
Best Lower Approx.	-	✓	-	(✓)	-
Version Subset	-	✓	-	(✓)	-
Opti. Min. Weights	-	✓	-	-	[19]
Version Bound	-	✓	-	-	-
Version Global	-	✓	-	-	[6, 10]

2 Framework

Recently, we examined the detailed analogies and differences in synthesis approaches based on regions of languages [14]. We identified two different types of regions, called *transition regions* and *token flow regions* (this is not explained in more detail here), and two different principles of computing from the (infinite) set of all regions a finite Petri net representing the saturated feasible net, namely the already mentioned principles of *separation computation* and *basis computation* [14]. Instead of solving the synthesis problem for a certain net class and a certain language specification, we presented a framework for region based synthesis of Petri nets from languages, which integrates almost all approaches known in literature and filled several remaining gaps [14]. The framework has four dimensions defining a synthesis setting: *Petri net class, language type, region type and computation principle* (see Fig.1). The first two dimensions define the synthesis problem, while the latter two determine a solution principle. Given a language type and a Petri net class, usually all four combinations of region type and computation principle yield a solution algorithm for the respective synthesis problem. The region type together with the language type and the Petri net class determines the inequality system $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq c$ defining the set of regions. A synthesis algorithm then only depends on this inequality system and on the computation principle. The two principles of separation and basis computation are applicable for almost all inequality systems in the same way, i.e. with the central ideas of each of the two computation

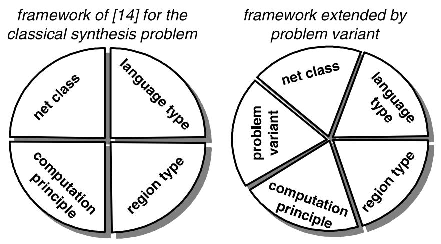

Figure 1. Synthesis framework.

principles nearly all synthesis settings can be solved. This systematic classification of synthesis approaches has been developed for the standard classical synthesis problem.

In this paper we solve the mentioned variants of the classical problem in the setting of the synthesis of a *p/t-net* (net class) from a *finite language of occurrence sequences* (language type) using *transition regions* (region type), whereas we apply both *separation and basis computation* (computation principle). The problem variants are solved by appropriate modifications of the solution algorithms for the classical synthesis problem. For each variant, we highlight the main ideas for adapting the two computation principles such that, similarly as for the classical synthesis problem (see [14]), the solution algorithms of the variants can in most cases easily be adapted to other synthesis settings, i.e. to other net classes such as inhibitor nets, to other language types such as partial languages and to the region type of token flow regions. Again, switching to other synthesis settings by changing net class, language type or region type only changes the inequality system $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq c$ defining regions, but the algorithmic ideas depending on the computation principle stay the same (apart from some adjustments). That means, as sketched in Fig. 1, considering different variants of the classical synthesis problem may be seen as a fifth dimension added to the synthesis framework presented in [14]. Altogether, each problem variant, although presented in our example setting, is discussed in the context of the general framework shown in Fig. 1 by highlighting the crucial ideas of the two computation principles.

Note that the setting considered in this paper is restricted to finite languages and adaptations of the synthesis algorithms to finitely represented infinite languages entail several difficulties not discussed here (requiring considerations going beyond [14]). In the case of occurrence sequences an overview how to deal with infinite languages is described in [8] and in [5] a synthesis approach for infinite partial languages is shown. Concerning applicability of the algorithms, finiteness is a crucial restriction. We consider exact synthesis algorithms which will not generate loop behavior if this is not specified. However, real systems often exhibit loop behavior. In order to be able to handle certain kinds of finite specifications of loop behavior, the algorithms still have to be generalized along the lines shown e.g. in [8, 5].

3 Classical Synthesis Problem

We start the technical part of the paper with basic notions. A *p/t-net* N is a triple (P, T, W), where P is a (possibly infinite) set of *places*, T is a finite set of *transitions* satisfying $P \cap T = \emptyset$, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is an *(arc) weight function* defining the flow relation ($\mathbb{N}$ denotes the non-negative integers). A *marking* of a net N is a function $m : P \to \mathbb{N} \in \mathbb{N}^P$ (called *multi-set* over P) assigning $m(p)$ tokens to a place $p \in P$. A *marked p/t-net* is a pair (N, m_0), where N is a p/t-net, and m_0 is a marking of N, called *initial marking*. A transition t is *enabled to occur* in a marking m of a p/t-net N, if $m(p) \geq W(p, t)$ for each $p \in P$. In this case, its occurrence leads to the marking $m'(p) = m(p) + W(t, p) - W(p, t)$, abbreviated by $m \xrightarrow{t} m'$. A finite sequence of transitions $\sigma = t_1 \ldots t_n \in T^*$, $n \in \mathbb{N}$, is called an *occurrence sequence enabled in a marking m and leading to m_n*, denoted by $m \xrightarrow{\sigma} m_n$, if there exists a sequence of markings $m_1, \ldots, m_n$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$. The set of all occurrence sequences enabled in m_0 is denoted by $\mathcal{L}(N, m_0)$. The set $\mathcal{L}(N, m_0) \subseteq T^*$ is a language over the alphabet T. The language $\mathcal{L}(N, m_0)$ is *prefix closed*, i.e. if $t_1 \ldots t_n \in \mathcal{L}(N, m_0)$ then each proper *prefix* $t_1 \ldots t_i$, $i < n$, (denoted by $t_1 \ldots t_i < t_1 \ldots t_n$) is also in $\mathcal{L}(N, m_0)$. The commutative image of a sequence of transitions $\sigma \in T^*$ is the multiset $[\sigma] \in \mathbb{N}^T$ whose respective entries $[\sigma](t)$ count the number of occurrences of $t \in T$ in σ.

In our setting, the synthesis starts with a prefix closed finite language of occurrence sequences $\mathcal{L} \subseteq T^*$ over a fixed finite alphabet (of transitions) T [8, 4].

Problem 1 (Classical Synthesis Problem)**.** *Given $\mathcal{L}$, decide whether there exists a p/t-net (N, m_0) fulfilling $\mathcal{L}(N, m_0) = \mathcal{L}$ and compute such net in the positive case.*

As stated we apply the region type called *transition region* for computing places of the synthesized net. Denoting $T = \{t_1, \ldots, t_n\}$, a *region* of $\mathcal{L}$ is a tuple $\mathbf{r} = (r_0, \ldots, r_{2n}) \in \mathbb{N}^{2n+1}$ satisfying
$(*)$ $\quad r_0 + \sum_{i=1}^{n}([w](t_i) \cdot r_i - [wt](t_i) \cdot r_{n+i}) \geq 0$
for every $wt \in \mathcal{L}$ ($w \in \mathcal{L}, t \in T$). Every region $\mathbf{r}$ of $\mathcal{L}$ defines a place $p_\mathbf{r}$ via $m_0(p_\mathbf{r}) := r_0$, $W(t_i, p_\mathbf{r}) := r_i$ and $W(p_\mathbf{r}, t_i) := r_{n+i}$ for $1 \leqslant i \leqslant n$. The place $p_\mathbf{r}$ is called *corresponding place to* $\mathbf{r}$. Regions exactly define so called feasible places: If (N, m_p), $N = (\{p\}, T, W_p)$ is a marked p/t-net with only one place p (W_p, m_p are defined according to the definition of p), the place p is called *feasible (w.r.t. $\mathcal{L}$)*, if $\mathcal{L} \subseteq \mathcal{L}(N, m_p)$, otherwise *non-feasible*. That means, feasible places do not prohibit behavior specified by $\mathcal{L}$. A central theorem establishes that each place corresponding to a region of $\mathcal{L}$ is feasible w.r.t. $\mathcal{L}$ and each place feasible w.r.t. $\mathcal{L}$ corresponds to a region of $\mathcal{L}$ [8, 4].

The set of regions can be characterized as the (infinite) set of non-negative integral solutions of a homogenous linear inequality system $\mathbf{A}_\mathcal{L} \cdot \mathbf{r} \geq \mathbf{0}$ [4] with integer coefficients. The matrix $\mathbf{A}_\mathcal{L}$ encodes property $(*)$. It consists of rows $\mathbf{a}_{wt}$ satisfying $\mathbf{a}_{wt} \cdot \mathbf{r} \geq \mathbf{0} \Leftrightarrow (*)$ for each $wt \in \mathcal{L}$. Note that $(*)$ only depends on $[w]$ and t, i.e. $\mathbf{a}_{wt}$ may be equal for different $wt \in \mathcal{L}$ (duplicate $\mathbf{a}_{wt}$ are omited).

3.1 Separation Computation

To compute a finite net, we first use the principle of separation computation. The idea behind this strategy is to add such feasible places to the constructed net, which *separate* specified behavior from non-specified behavior. For each $w \in \mathcal{L}$ and each $t \in T$ such that $wt \notin \mathcal{L}$, one searches for a feasible place p_{wt}, which prohibits the occurrence of wt. Such wt is called *wrong continuation* and the set comprising of all wrong continuations is denoted by WC. A feasible place p_{wt} prohibiting a wrong continuation wt is called *separating feasible place w.r.t. wt*. If there is such a separating feasible place for $wt \in WC$, it is added to the net. The number of wrong continuations (and thus the number of places) is bounded by $|\mathcal{L}| \cdot |T|$. The constructed net (N, m_0) containing one separating feasible place for each wrong continuation, for which such place exists, is finite. Separating feasible places are computed through *separating regions*. A region $\mathbf{r}$ of $\mathcal{L}$ is a *separating region* w.r.t. a wrong continuation wt if
$(**)$ $\quad r_0 + \sum_{i=1}^{n}([w](t_i) \cdot r_i - [wt](t_i) \cdot r_{n+i}) < 0.$
In [8, 4, 1] it is shown that a separating feasible place w.r.t. a wrong continuation wt corresponds to a separating region w.r.t. wt and vice versa. A separating region $\mathbf{r}$ w.r.t. $wt \in WC$ can be calculated (if it exists) as a non-negative integer solution of a homogenous linear inequality system with integer coefficients of the form $\mathbf{A}_\mathcal{L} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$, where the vector $\mathbf{b}_{wt}$ is defined in such a way that $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0} \Leftrightarrow (**)$. Note that $(**)$ only depends on $[w]$ and t, i.e. $\mathbf{b}_{wt}$ may be equal for different $wt \in WC$ (duplicate $\mathbf{b}_{wt}$ can be neglected). If there exists no non-negative integer solution of the system $\mathbf{A}_\mathcal{L} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$, there exists no separating region w.r.t. wt, and thus no separating feasible place prohibiting wt. If there exists a non-negative integer solution, any such solution defines a separating feasible place prohibiting wt. In order to decide the solvability of the inequality system and to compte a solution in the positive case several linear programming solvers, such as the Simplex method, the method by Khachyan or the method of Karmarkar, can be applied [18] (the latter two have polynomial runtime). It is important here that the homogeneity of the system enables the use of such solvers searching for rational solutions, since multiplying with the common denominator of the entries of a rational solution yields an integer solution.

The final synthesis algorithm **Algorithm 1** to solve Problem 1 works as follows [4]: Each wrong continuation is processed. For a wrong continuation the solvability of the corresponding inequality system is decided. If there is no solution, the synthesis problem has a negative answer. If there exists a solution, one such solution is chosen and the corresponding separating feasible place is added to the net. If there is such a separating feasible place for every wrong continuation, it was shown in [8, 4] (by proving that it is enough to prohibit the set of wrong continuations of $\mathcal{L}$ in order to prohibit all $w \notin \mathcal{L}$) that the synthesis problem has a positive answer and the constructed net is a respective witness net. Choosing a solver running in polynomial time, the synthesis algorithm has a polynomial time consumption [1, 4]. This basic synthesis procedure is improved by the following principle: For not yet processed wrong continuations, that are prohibited by feasible places already added to the constructed net, we do not have to calculate a separating feasible place. Therefore, we choose a certain ordering of the wrong continuations. We first add a separating feasible place for the first wrong continuation (if such place exists). Then, we only add a separating feasible place for the second wrong continuation, if it is not prohibited by an already added feasible place, and so on.

3.2 Basis Computation

Instead of the separation computation, we can also use the principle of basis computation to compute a finite net. The idea here is to add a finite subset of the infinite set of feasible places to the constructed net, such that this subset restricts the behavior of the net in the same way as the set of all feasible places. The set of solutions of the system $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$ defines a pointed polyhedral cone. Since all values in $\mathbf{A}_{\mathcal{L}}$ are integral, there always exists a minimal set of integer solutions $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, such that each solution $\mathbf{x}$ is a non-negative linear combination of $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ of the form $\mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{y}_i$ for real numbers $\lambda_1, \ldots, \lambda_n \geqslant 0$ [18]. This set is unique up to dilation and given by the rays of the cone. It can be computed e.g. by the algorithm of Chernikova, which has exponential runtime in the worst case. The problem is that the size of $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ may be exponential, but it often has a reasonable size. We call the elements of this set *basis regions*. It is shown in [8, 4] that the saturated feasible net has the same set of occurrence sequences as the finite net (N, m_0) consisting only of feasible places corresponding to basis regions. Thus, (N, m_0) represents a best upper approximation to $\mathcal{L}$, i.e. $\mathcal{L} \subseteq \mathcal{L}(N, m_0)$ and $\forall (N' m_0') : (\mathcal{L} \subseteq \mathcal{L}(N', m_0')) \Longrightarrow (\mathcal{L}(N, m_0) \subseteq \mathcal{L}(N', m_0'))$. Consequently, either (N, m_0) solves the synthesis problem positively or the problem has a negative answer. It remains to check whether (N, m_0) solves the synthesis problem positively or not. For this one can either compute $\mathcal{L}(N, m_0)$ (which is finite [4]) and test whether $\mathcal{L}(N, m_0) \subseteq \mathcal{L}$, or one can check whether each wrong continuation of $\mathcal{L}$ is not enabled in (N, m_0).

The final synthesis algorithm **Algorithm 2** to solve Problem 1 works as follows [4]: The set of basis regions of $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$ is computed, and the finite set of feasible places corresponding to basis regions is added to the net. For the resulting net (N, m_0) it is checked whether $\mathcal{L}(N, m_0) = \mathcal{L}$ or not. In the positive case the synthesis problem has a positive answer and (N, m_0) is a respective witness net. In the negative case the synthesis problem has a negative answer.

4 Variants of the Classical Synthesis Problem

In this section we discuss variants of the classical synthesis problem accounting for typical requirements on a system model going beyond the classical synthesis question. Since the variants considered cover a wide field of problems, the solution methods presented in this section constitute a large repertoire of synthesis procedures.

4.1 Bound Variant

Instead of specifying a language and asking whether the language can exactly be reproduced by a net, one can specify two languages representing a lower and an upper bound for the behavior of a net. It is asked whether there is a net having more behavior than specified by the first language, but less behavior than specified by the second one. This problem variant is useful in the frequent situations of incomplete specifications, since it is possible to specify some range of tolerance for the behavior of the synthesized net. The variant is particularly relevant in the application field of control synthesis [7, 10] (see e.g. [9, 15, 3, 10] for refined versions of the supervisory control problem).

Problem 2 (Bound Variant). *Given $\mathcal{L}, \mathcal{L}'$, $\mathcal{L} \subseteq \mathcal{L}'$, decide whether there exists a marked p/t-net (N, m_0) fulfilling $\mathcal{L} \subseteq \mathcal{L}(N, m_0) \subseteq \mathcal{L}'$ and compute such net in the positive case.*

The bound variant can be solved by considering regions w.r.t. $\mathcal{L}$ as before (nets only having feasible places w.r.t $\mathcal{L}$ are candidates), but wrong continuations resp. the set inclusion test is considered for $\mathcal{L}'$. Using Algorithm 2 the bound variant is solved in [7] considering a regular language and pure nets or p/t-nets. Regarding the best upper approximation property of (N, m_0), it is clear that if in Algorithm 2 the test whether $\mathcal{L}(N, m_0) \subseteq \mathcal{L}$ is replaced by a test whether $\mathcal{L}(N, m_0) \subseteq \mathcal{L}'$, the algorithm solves Problem 2. More efficiently, Problem 2 can be solved by changing Algorithm 1 as follows: for each wrong continuation $wt \in WC$ w.r.t. $\mathcal{L}'$ it is searched for a separating region. If there is no such separating region, there are two cases: Either $w \in \mathcal{L}$ (it always holds $\epsilon \in \mathcal{L}$), then the formulated problem has a negative answer, or $w \notin \mathcal{L}$, then w is considered as a wrong continuation by adding it to the set WC. If and only if the first case never occurs, a positive answer to the synthesis problem having (N, m_0) as a witness can be deduced analogously as for Algorithm 1, since in this case some prefix of each wrong continuation w.r.t. $\mathcal{L}'$ is separated.

Version Unwanted: In applications an upper bound $\mathcal{L}'$ may be specified indirectly by a set $\tilde{L}$ representing a set $\tilde{\mathcal{L}} = \{w \mid \exists w' \in \tilde{L} : w' \text{ prefix of } w\}$ of *unwanted behavior*. The upper bound $\mathcal{L}'$ is then given by the complement of $\tilde{\mathcal{L}}$. The lower bound $\mathcal{L}$ is given as usual by a specification of wanted behavior. In this paper, $\tilde{L}$ is finite. But the respective upper bound $\mathcal{L}'$ may be infinite. In the case of Algorithm 2, instead of checking whether $\mathcal{L}(N, m_0) \subseteq \mathcal{L}'$, one can test whether for some $w \in \tilde{L}$, there holds $w \in \mathcal{L}(N, m_0)$. In the positive case, the bound variant has a negative answer (by the best upper approximation property of (N, m_0)). In the negative case, it has a positive answer having (N, m_0) as a witness. In the case of Algorithm 1, instead of considering wrong continuations w.r.t. $\mathcal{L}'$, the set of occurrence sequences that have to be separated is directly given by $\tilde{L}$.

Version Conformance: In some applications, behavioral bounds are given by a *conformance measure* μ appropriately scaling the degree of conformance of a language and a p/t-net. Then, a value μ_0 specifies a lower conformance bound. Given a language $\mathcal{L}$, a p/t-net (N, m_0) fulfilling $\mathcal{L} \subseteq \mathcal{L}(N, m_0)$ and respecting the conformance bound through $\mu((N, m_0), \mathcal{L}) \geq \mu_0$ may be searched. While $\mathcal{L}$ defines the lower behavioral bound, the conformance bound can be seen as an upper bound for the behavior of the searched net. In the case μ is monotonic for upper approximations of the language in the sense that for $\mathcal{L} \subseteq \mathcal{L}(N, m_0) \subseteq \mathcal{L}(N', m_0')$ there holds $\mu((N, m_0), \mathcal{L}) \geq \mu((N', m_0'), \mathcal{L})$, the formulated problem can be solved as follows: Start computing a best upper approximation (N, m_0) to $\mathcal{L}$ as shown later on, and then accomplish a conformance test whether $\mu((N, m_0), \mathcal{L}) \geq \mu_0$. In the case of a synthesis algorithm that adds feasible places stepwise such as Algorithm 1, the conformance test can also be accomplished in each step of the algorithm, and if the test is positive, the so far computed net solves the problem. A trivial monotonic conformance measure is $\mu((N, m_0), \mathcal{L}) = 1 - (|\mathcal{L}(N, m_0) \setminus \mathcal{L}|)/|\mathcal{L}(N, m_0)|$ (μ is 0 for infinite $\mathcal{L}(N, m_0)$ and 1 for $\mathcal{L}(N, m_0) = \mathcal{L}$). Examples for more advanced conformance measures are shown in [16]. In particular, the behavioral appropriateness metrics a_B of [16] is monotonic.

4.2 Place Bound Variant

A crucial requirement in practice is synthesizing compact, manually interpretable reference models [19]. In particular, Petri nets having a small number of components are desired. Thus, an interesting problem is whether there exists a net having at most a specified number of places, which has the specified behavior.

Problem 3 (Place Bound Variant)**.** *Given $\mathcal{L}$ and a bound $b \in \mathbb{N}\backslash\{0\}$, decide whether there exists a p/t-net $(N, m_0) = (P, T, W, m_0)$, $|P| \leq b$, fulfilling $\mathcal{L}(N, m_0) = \mathcal{L}$ and compute such net in the positive case.*

Using Algorithm 1 the problem can basically be solved by partitioning the set of wrong continuations to b sets $WC_1, \ldots, WC_b$. If for one such partition (there are exponentially many of these partitions) it is possible to separate the b sets of wrong continuations each by one feasible place, the synthesis problem has a positive answer, otherwise a negative answer. An advantage of this approach is that still standard (polynomial) linear programming techniques can be applied to small problem instances.

But it is also possible to apply a more advanced technique following ideas developed in [6]. Although, as stated in [6], the approach in [6] is not a region based synthesis procedure, the presented principle of considering an integer linear programming problem can also be used in our setting of regions of languages. The place bound variant can be solved by solving the following system: $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$, $i \in \{1, \ldots, b\} \mid -k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$, $i \in \{1, \ldots, b\}$, $wt \in WC \mid \sum_{i=1}^{b} s_{wt,i} \leq b - 1$, $wt \in WC \mid \mathbf{r}^i \in \mathbb{N}^{2n+1}$, $i \in \{1, \ldots, b\} \mid k \in \mathbb{N} \mid s_{wt,i} \in \{0, 1\}$, $i \in \{1, \ldots, b\}$, $wt \in WC$. The vectors $\mathbf{r}^i$ represent b regions by the inequalities $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$. If $s_{wt,i} = 0$ then the constraint $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$ is active yielding the usual constraint to separate the wrong continuation wt by the region $\mathbf{r}^i$. If $s_{wt,i} = 1$ the constraint can be easily verified by choosing the variable k large enough, thus resulting in a redundant constraint. Moreover, the condition $\sum_{i=1}^{b} s_{wt,i} \leq b - 1$ implies that at least one $s_{wt,i}$ is equal to zero, i.e. for each wrong continuation wt one constraint is active ensuring that wt is separated by one of the regions $\mathbf{r}^i$. This guarantees that if there is a solution of the system, all wrong continuations are separated by one of the b feasible places corresponding to $\mathbf{r}^1, \ldots, \mathbf{r}^b$, i.e. we have a solution net to Problem 3. Conversely, if Problem 3 has a solution net, the integer linear programming problem has a solution as follows: $\mathbf{r}^i$ can be chosen such that each place of the net corresponds to one $\mathbf{r}^i$. Furthermore, set $s_{wt,i} = 0$ if the place corresponding to $\mathbf{r}^i$ separates wt, and otherwise $s_{wt,i} = 1$. Choosing k large enough, this ensures that all constraints $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$ are satisfied. Since every wrong continuation is separated by one place of the net, the constraint $\sum_{i=1}^{b} s_{wt,i} \leq b - 1$ is fulfilled.

The arising integer linear programming problem can be solved by standard methods [18] (solving a series of usual linear programming problems) such as branch and bound algorithms or the cutting-plane (Gomory) method. Both have exponential runtime in the worst case. The presented inequality system is large, which may cause performance problems. But state of the art integer linear programming solvers are very efficient, such that also large problems can be handled.

Lastly, concerning Problem 3, Algorithm 2 is only useful as a kind of pre-processing: Instead of defining the set of feasible places by solutions of an inequality system, they can be given as linear combinations of the places corresponding to basis regions.

Version Minimal: An algorithm solving the place bound problem can be used to *construct a net with a minimal number of places solving the synthesis problem*. First it has to be checked if the synthesis problem is solvable. In the positive case one decides whether the place bound variant is solvable for $b = 0$, then for $b = 1$, then for $b = 2$ and so on. The smallest b giving a positive answer, yields the solution to the formulated problem and the algorithm terminates.

Again following ideas in [6], the version of the synthesis problem optimizing the number of places can also be solved by an integer linear programming problem. If the classical synthesis problem is solvable, then consider the number b of places of a solution net and solve the following problem: $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$, $i \in \{1, \dots, b\}$ | $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$, $i \in \{1, \dots, b\}$, $wt \in WC$ | $\sum_{i=1}^{b} s_{wt,i} \leq b - 1$, $wt \in WC$ | $k \cdot z_i - \sum_{j=0}^{2n} r_j^i \geq 0$, $i \in \{1, \dots, b\}$ | $\mathbf{r}^i \in \mathbb{N}^{2n+1}$, $i \in \{1, \dots, b\}$ | $k \in \mathbb{N}$ | $z_i, s_{wt,i} \in \{0, 1\}$, $i \in \{1, \dots, b\}$, $wt \in WC$ | $min! \sum_{i=1}^{b} z_i$. The inequality system is the same as before with additional binary variables z_i and inequalities $k \cdot z_i - \sum_{j=0}^{2n} r_j^i \geq 0$. If $z_i = 1$ this constraint is trivially fulfilled by choosing k large enough. In the case $z_i = 0$ the constraint is only satisfied if $\mathbf{r}^i = 0$, i.e. the place defined by $\mathbf{r}^i$ is the redundant zero place. Additionally, the integer linear program minimizes $\sum_{i=1}^{b} z_i$. Therefore, as many as possible $\mathbf{r}^i$ are set to zero such that the corresponding places can be omitted from the computed net. Thus, solving this problem yields a net solving the synthesis problem and having a minimal number of places.

Version Sub-Net: A relaxed and simpler problem is *constructing a net solving the synthesis problem which has no sub-net also solving the synthesis problem*. This can be achieved by answering the usual synthesis problem and in the positive case exploring all places of the synthesized net in an arbitrary order. In the case of Algorithm 1 for each place it is checked whether the place can be removed and still all wrong continuations are separated by the remaining places. Note that choosing another ordering of exploring the places may lead to a net with a smaller number of places [8]. Actually, the proposed procedure means to check for each place if it is implicit. Searching for implicit places can also be applied in the case of Algorithm 2. Heuristics to find implicit places can be used to construct approximate solutions.

4.3 Identifying States Variant

In a system specification, there may be partial information about states of the system, e.g. error states and normally terminating states. Typically, it is specified that some executions yield the same state. Such information can be integrated in synthesis methods.

Problem 4 (Identifying States Variant)**.** *Given $\mathcal{L}$ and pairwise disjoint $\mathcal{L}_1, \dots, \mathcal{L}_l \subseteq \mathcal{L}$, decide whether there exists a p/t-net (N, m_0) fulfilling $\mathcal{L}(N, m_0) = \mathcal{L}$ such that for each $j \in \{1, \dots, l\}$ the occurrence of all $\sigma \in \mathcal{L}_j$ leads to the same marking, and compute such net in the positive case.*

The additional requirements define additional restrictions for regions. Fix $\sigma_j \in \mathcal{L}_j$ for each $j \in \{1, \dots, l\}$. Then for each $\sigma \neq \sigma_j$, $\sigma \in \mathcal{L}_j$, add two rows $\mathbf{s}_\sigma = (s_{\sigma,0}, \dots, s_{\sigma,2n})$ and $-\mathbf{s}_\sigma$ to matrix $\mathbf{A}_{\mathcal{L}}$, such that $\mathbf{s}_\sigma \cdot \mathbf{r} \geq \mathbf{0}$ and $-\mathbf{s}_\sigma \cdot \mathbf{r} \geq \mathbf{0}$ if and only if the occurrence of σ and σ_j lead to the same number of tokens in $p_{\mathbf{r}}$:

$$\mathbf{s}_{\sigma,i} = \begin{cases} 0 & \text{for } i = 0 \\ [\sigma](t_i) - [\sigma_j](t_i) & \text{for } i = 1, \dots, n \\ -[\sigma](t_{i-n}) + [\sigma_j](t_{i-n}) & \text{for } i = n+1, \dots, 2n \end{cases}$$

Considering this extended matrix $\mathbf{A}_{\mathcal{L}}$, Algorithm 1 and 2 solve Problem 4.

Version Separating: The state variant can be generalized by not only specifying equal states but also *separated states*. That means, for certain pairs $(\mathcal{L}_i, \mathcal{L}_j)$ the two markings defined by $\mathcal{L}_i$ and $\mathcal{L}_j$ are specified to be different. For this the marking in one place separating these two states has to be different. In the case of Algorithm 1 feasible places separating such states can be computed similarly as feasible places separating wrong continuations, i.e. for each such pair of states it is tried to solve the inequality system $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$ considered in this subsection together with (instead of $\mathbf{b}_w \cdot \mathbf{r} < \mathbf{0}$) an inequality ensuring that the resulting feasible place separates the two states (defined by rows similar to $\mathbf{s}_\sigma$). Concerning Algorithm 2, the net (N, m_0) computed with the inequality system of this subsection is a candidate to solve the problem, i.e. it is sufficient to check if for each specified pair $(\mathcal{L}_i, \mathcal{L}_j)$ of different markings, the two final markings given by the occurrence of σ_i and σ_j in the net (N, m_0) are different. If this is not the case, no feasible place satisfying the requirement of Problem 4 separates these two states, since each such feasible place is a non-negative linear combination of the places of (N, m_0). Consequently, the problem has a negative answer. Completely specifying which states are equal and which states are separated yields the classical problem of *synthesis up to isomorphism* [2].

4.4 Best Upper Approximation

The previous synthesis problems do not require the computation of a net, if exact synthesis is not possible. But typical applications ask for the construction of a reasonable system model from arbitrary specifications. For this purpose, synthesis of (best) approximate solutions is appropriate. Thereby, synthesizing upper approximations to specifications is useful, because in many applications of net synthesis the behavior explicitly specified by a language should definitely be included in the language of the synthesized model. Best upper approximations ensure that only necessary additional behavior is added to the synthesized net. Thus, computing a best upper approximation may be seen as a natural completion of the specified language by a Petri net. Generally, in applications often approximate system models are sufficient and sometimes upper approximations to specifications are even desired, since system specifications in practice are typically incomplete. Formally, we here consider synthesis algorithms generating a net with a language which is included in each net language larger than the specified language and is itself larger than the specified language [8]. The theory of regions shows that such net language exists [8, 4]. This language is of course unique. In particular, it is the unique smallest net language larger than the specified language.

Problem 5 (Best Upper Approximation Variant)**.** *Given $\mathcal{L}$, compute a marked p/t-net (N, m_0) fulfilling $\mathcal{L} \subseteq \mathcal{L}(N, m_0)$ and $(\forall (N' m_0') : (\mathcal{L}(N, m_0) \setminus \mathcal{L}(N', m_0') \neq \emptyset) \Longrightarrow (\mathcal{L} \not\subseteq \mathcal{L}(N', m_0')))$.*

As shown in Section 3, the first part of Algorithm 2 (without the check whether $\mathcal{L}(N, m_0) = \mathcal{L}$) already solves Problem 5. But computing the complete basis often leads to performance problems. The net (N, m_0) computed with Algorithm 1 in general does not solve Problem 5, but (N, m_0) is an upper approximation to $\mathcal{L}$, i.e. $\mathcal{L} \subseteq \mathcal{L}(N, m_0)$. The reason is that even if there is no feasible place prohibiting a wrong continuation w, there might be one prohibiting wt – but such places are not added to (N, m_0). Therefore, the following adaptation of Algorithm 1 is necessary to solve Problem 5: If there is no feasible place prohibiting a wrong continuation w, for each transition t try to construct a feasible place prohibiting wt by considering wt as a wrong continuation, and if there is no such place, for each t' try to construct a feasible place prohibiting wtt', and so on. In this algorithm the set of wrong continuations WC may grow, but the algorithm terminates: A sequence in which a transition t occurs more often than the maximal number of occurrences of t in a sequence of $\mathcal{L}$ can always be separated by the feasible place p defined by $W(p, t) = 1$, $W(p, t') = 0$ for $t' \in T \setminus \{t\}$, $W(t', p) = 0$ for $t' \in T$, $m_0(p) = max\{[w](t) \mid w \in \mathcal{L}\}$. Thus, the length of a sequence added to WC is bounded by $\sum_{t \in T} max\{[w](t) \mid w \in \mathcal{L}\} + 1$, and by construction no sequence is added twice to WC. The net (N, m_0) computed by the sketched adaptation of Algorithm 1 fulfills the best upper approximation property, since every sequence in $\mathcal{L}(N, m_0) \setminus \mathcal{L}$ cannot be prohibited by a feasible place, i.e. such sequence is included in the behavior $\mathcal{L}(N', m_0')$ of every net (N', m_0') fulfilling $\mathcal{L} \subseteq \mathcal{L}(N', m_0')$.

4.5 Best Lower Approximation

Although upper approximations are more common, there are also examples requiring the synthesis of lower approximations. Lower approximations are nets having only behavior specified by the given language. This is useful in the case the specification is complete, i.e. all behavior not specified in the language is faulty behavior. In Petri net theory, best lower approximations exhibit some difficulties compared to best upper approximations. In particular, there is no unique largest net language smaller than a specified language, e.g. $\mathcal{L} = \{b, a, aa, aab\}$ is no p/t-net language, but $\{b, a, aa\}$ and $\{a, aa, aab\}$ are both p/t-net languages. Therefore, it is not reasonable to formulate the best lower approximation variant in such a strict way as in the case of the best upper approximation variant. We here consider a best lower approximation to be a lower approximation having maximal behavior in the sense that no other lower approximation has a larger number of occurrence sequences specified by the language.

Problem 6 (Best Lower Approximation Variant)**.** *Given $\mathcal{L}$, compute a marked p/t-net (N, m_0) fulfilling $\mathcal{L} \supseteq \mathcal{L}(N, m_0)$ and $(\forall (N' m_0') : (|\mathcal{L}(N', m_0')| > |\mathcal{L}(N, m_0)|) \Longrightarrow (\mathcal{L} \not\supseteq \mathcal{L}(N', m_0')))$.*

This problem can obviously be solved by solving the classical synthesis problem for each prefix closed subset of $\mathcal{L}$. There are subsets with a maximal number of elements for which Algorithm 1 resp. 2 yields a positive answer. A net computed in such a case is a best lower approximation.

Using Algorithm 1 the problem can be solved a lot more efficiently by applying the following procedure: First, apply Algorithm 1 once and in doing so store all remaining wrong continuations that cannot be separated. The computed net is the starting point. In the following, it is only necessary to consider the remaining stored wrong continuations. Thus, simplify Algorithm 1 by substituting the usually large set WC by the usually small set of remaining stored wrong continuations. Apply this simplification of Algorithm 1 to all prefix closed subsets of $\mathcal{L}$ in decreasing order (w.r.t. the number of elements) until discovering some subset yielding a positive answer. The places computed in such case supplement the places of the starting net. This yields a net with maximal behavior separating all wrong continuations.

The problem can also be encoded in an integer linear program: $k \cdot z_{wt} + \mathbf{a}_{wt} \cdot \mathbf{r}^{vu} \geq \mathbf{0}$, $wt \in \mathcal{L}$, $vu \in WC \mid -k \cdot (1 - z_{wt} + \sum_{w't' < wt} z_{w't'}) + \mathbf{a}_{wt} \cdot \mathbf{r}^{vu} < \mathbf{0}$, $wt \in \mathcal{L}$, $vu \in WC$, $wt < vu \mid -k \cdot (\sum_{wt < vu} z_{wt}) + \mathbf{b}_{vu} \cdot \mathbf{r}^{vu} < \mathbf{0}$, $vu \in WC \mid z_{w't'} \leq z_{wt}$, $w't' < wt \in \mathcal{L} \mid \mathbf{r}^{vu} \in \mathbb{N}^{2n+1}$, $vu \in WC \mid k \in \mathbb{N} \mid z_{wt} \in \{0, 1\}$, $wt \in \mathcal{L} \mid min! \sum_{wt \in \mathcal{L}} z_{wt}$. There is one vector $\mathbf{r}^{vu}$ defining a place for each wrong continuation vu. All wrong continuations have to be separated by places such that the resulting net is a lower approximation. A wrong continuation is also prohibited if some prefix is separated. The constraints $-k \cdot (\sum_{wt<vu} z_{wt}) + \mathbf{b}_{vu} \cdot \mathbf{r}^{vu} < \mathbf{0}$ require that vu is separated by $\mathbf{r}^{vu}$ or that $z_{wt} = 1$ for some prefix wt of vu (in the second case the constraint is redundant, because k can be chosen arbitrarily large). If $z_{w't'} = 1$, then for all $wt > w't'$ we have $z_{wt} = 1$ by the constraints $z_{w't'} \leq z_{wt}$. If $z_{wt} = 1$ the constraints $-k \cdot (1 - z_{wt} + \sum_{w't'<wt} z_{w't'}) + \mathbf{a}_{wt} \cdot \mathbf{r}^{vu} < \mathbf{0}$ ensure that wt is separated by $\mathbf{r}^{vu}$, where $wt < vu \in WC$, or that $z_{w't'} = 1$ for some prefix $w't'$ of wt (in the second case and in the case $z_{wt} = 0$ the constraint is redundant, because k can be chosen arbitrarily large). It is possible to use $\mathbf{r}^{vu}$ to separate wt, because $\mathbf{r}^{vu}$ is not longer needed to separate vu, i.e. $\mathbf{r}^{vu}$ separates the minimal prefix wt of vu with $z_{wt} = 1$. Since all words wt with $z_{wt} = 1$ are separated directly or indirectly by separating some prefix, we do only require sequences wt with $z_{wt} = 0$ to be enabled by the constraints $k \cdot z_{wt} + \mathbf{a}_{wt} \cdot \mathbf{r}^{vu} \geq \mathbf{0}$. Altogether, the places corresponding to $\mathbf{r}^{vu}$, $vu \in WC$, prohibit all wrong continuations, i.e. the resulting net is a lower approximation to $\mathcal{L}$, and all $wt \in \mathcal{L}$ with $z_{wt} = 1$ are prohibited while all $wt \in \mathcal{L}$ with $z_{wt} = 0$ are enabled. The objective function $min! \sum_{wt \in \mathcal{L}} z_{wt}$ minimizes the number of prohibited sequences $wt \in \mathcal{L}$, such that the resulting net is a best lower approximation to $\mathcal{L}$.

Version Subset: A weaker problem emerges by replacing $(|\mathcal{L}(N', m_0')| > |\mathcal{L}(N, m_0)|)$ by $(\mathcal{L}(N', m_0') \supsetneq \mathcal{L}(N, m_0))$ in Problem 6. Then one searches for a *lower approximation having maximal behavior w.r.t. set inclusion*, not w.r.t. the number of elements (for infinite languages only this version is reasonable). This problem can be solved analogously.

4.6 Optimization

As stated before, a major challenge in Petri net synthesis is the creation of concise, readable nets (note that readability is actually an empirical notion). A promising approach to generate such nets is linear programming. A problem that can be solved by linear programming methods is minimizing arc weights and initial markings of places [6]. This problem is exemplarily considered here, but also other objective functions are possible. In literature, objective functions for guiding the construction of Petri nets have been considered in [6, 4, 19].

Problem 7 (Minimal Weights Variant)**.** *Given $\mathcal{L}$, decide whether there exists a p/t-net (N, m_0) fulfilling $\mathcal{L}(N, m_0) = \mathcal{L}$ and in the positive case, compute such p/t-net $(N, m_0) = (P, T, W, m_0)$ satisfying additionally $max\{m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p)) \mid p \in P\} \leq max\{m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p)) \mid p \in P'\}$ for all $(N', m_0') = (P', T, W', m_0')$ fulfilling $\mathcal{L}(N', m_0') = \mathcal{L}$.*

This problem can be solved by adding to the inequality systems considered in Algorithm 1 the linear objective function $min! \sum_{i=0}^{2n} r_i$. That means, instead of just computing one arbitrary solution of the considered inequality systems, a solution optimizing the objective function is computed. This ensures that a place p constructed to separate a wrong continuation has a minimal value $m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p))$ among all such places. Thus, the requirement of Problem 7 is satisfied. Actually, the computed net even fulfills stronger requirements, because locally for each considered wrong continuation a "minimal" place is computed. The arising integer linear programming problems can be solved by standard integer linear programming solvers. Note that, while in Algorithm 1 it is possible to apply rational solvers, this is not any more the case here, because multiplying an optimal rational solution vector by the common denominator of the vector entries may lead to a non-optimal integer solution w.r.t the objective function.

As in the case of the place bound variant, for optimization, computing basis regions is only useful as a preprocessing step.

Version Bound: Using optimization, it is also possible to consider *bounds for objective functions*, e.g. a bound b for $m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p))$ (for each place p). To decide the synthesis problem under this requirement, the above optimization algorithm is changed as follows: If some optimal separating region does not fulfill $r_0 + \ldots + r_{2n} \leq b$, the decision problem has a negative answer and the corresponding place is not added.

Version Global: Also, *global optimization* over a set of places is possible. By considering each partition $WC = WC_1 \uplus \ldots \uplus WC_a$ of the set of wrong continuations, trying to separate the a sets of wrong continuations each by one region $\mathbf{r}$, and regarding the objective function $min! \sum_{i=0}^{2n} r_i$ in each case, the synthesis problem can be solved yielding in the positive case an optimal net w.r.t. the global objective function $min! \sum_{p \in P}(m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p)))$.

Such global optimization problems can also be tackled by integer linear programming. One can proceed similarly as in the case of Problem 3. There are only two differences: First, we allow as many places as wrong continuations. This is the maximal number of places, which may be necessary (smaller numbers are then also possible, because all-zero regions are possible and the corresponding places can be omitted). Second, we simply add a respective objective function to the integer linear programming problem. Considering the global objective function $\sum_{p \in P}(m_0(p) + \sum_{t \in T}(W(p,t) + W(t,p)))$, the system looks as follows: $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$, $i \in \{1, \ldots, |WC|\}$ $\mid$ $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$, $i \in \{1, \ldots, |WC|\}$, $wt \in WC$ $\mid$ $\sum_{i=1}^{|WC|} s_{wt,i} \leq |WC| - 1$, $wt \in WC$ $\mid$ $\mathbf{r}^i \in \mathbb{N}^{2n+1}$, $i \in \{1, \ldots, |WC|\}$ $\mid$ $k \in \mathbb{N}$ $\mid$ $s_{wt,i} \in \{0, 1\}$, $i \in \{1, \ldots, |WC|\}$, $wt \in WC$ $\mid$ $min! \sum_{i=1}^{|WC|} \sum_{j=0}^{2n} r_j^i$.

Developing good synthesis methods using optimization is one of the main tasks for better practical applicability of synthesis [10]. In particular, it is possible to further simplify the synthesized nets, if they do not have to exactly reproduce the given language. There is a trade-off between a simpler less precise model and a more complex more precise model. To account for this, on the one hand, arc weights and initial markings of places (or similar parameters characterizing the complexity of a net) have to be considered as structural costs that have to be minimized as shown in this section. On the other hand, behavior of the net that is not specified by the language (or some other unwanted behavior) yields behavioral costs, i.e. instead of solving a certain exact synthesis problem, one may impose costs for additional unwanted behavior. Thus, places with small arc weights (to yield small structural costs) separating many wrong continuations (to reduce behavioral costs) are desired. Also other kinds of costs may be considered, e.g. for communication in distributed components or for relaxing a conformance measure. This leads to complex optimization (also non-linear) problems and games with equilibria. It is mainly asked for heuristical procedures and approximate solutions. Examples in literature regarding costs are [3, 17, 15].

5 Conclusion

Although a lot of interesting language based synthesis problems have been discussed in a systematic way, the overview given is incomprehensive. It is of course possible to formulate further problem variants. In literature, the following additional variants have been considered: Synthesis of nets fulfilling certain properties [6, 19, 4], e.g. restrictions of arc weights by certain values, restrictions of markings in certain system states, structural restrictions to nets of certain types such as free-choice nets, marked graphs or state machines, fulfillment of transition invariants or correctness properties such as soundness of workflow nets (soundness is a certain requirement on liveness and boundedness of workflow nets), synthesis of bounded nets [1, 13, 4] (also relaxed versions of boundedness [9]) and synthesis of so called distributable nets [9, 7, 10].

The main direction for future theoretical work is to solve further problem variants with the same systematic approach followed in this paper. Furthermore, it is important to consider more practical aspects. In particular, results from empirical studies are significant for fine-tuning the presented synthesis methods in the view of the application in real industrial settings.

References

[1] E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial Algorithms for the Synthesis of Bounded Nets. In *TAPSOFT 1995, LNCS 915*, pages 364–378, 1995.

[2] E. Badouel and P. Darondeau. Theory of Regions. In *Petri Nets, LNCS 1491*, pages 529–586, 1996.

[3] F. Basile, P. Chiacchio, and C. Seatzu. Optimal Petri Net Monitor Design. In *Synthesis and Control of Discrete Event Systems, Kluwer*, pages 141–152, 2002.

[4] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In *BPM 2007, LNCS 4714*, pages 375–383, 2007.

[5] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri Nets from Infinite Partial Languages. In *ACSD 2008, IEEE*, pages 170–179, 2008.

[6] M. P. Cabasino, A. Giua, and C. Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007.

[7] P. Darondeau. Region Based Synthesis of P/T-Nets and Its Potential Applications. In *ICATPN 2000, LNCS 1825*, pages 16–23, 2000.

[8] P. Darondeau. Unbounded Petri Net Synthesis. In *Lectures on Concurrency and Petri Nets, LNCS 3098*, pages 413–438, 2003.

[9] P. Darondeau. Distributed Implementations of Ramadge-Wonham Supervisory Control with Petri Nets. In *CDC-ECC 2005, IEEE*, pages 2107–2112, 2005.

[10] P. Darondeau. Synthesis and Control of Asynchronous and Distributed Systems. In *ACSD 2007, IEEE*, pages 13–22, 2007.

[11] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-Structures. Part I: Basic Notions and the Representation Problem / Part II: State Spaces of Concurrent Systems. *Acta Inf.*, 27(4):315–368, 1989.

[12] H. Liang, J. Dingel, and Z. Diskin. A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches. In *SCESM 2006, ACM*, pages 5–12, 2006.

[13] R. Lorenz, R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. In *ACSD 2007, IEEE*, pages 157–166, 2007.

[14] R. Lorenz, G. Juhás, and S. Mauser. How to Synthesize Nets from Languages - a Survey. In *WSC 2007*, pages 638–647, 2007.

[15] S. A. Reveliotis and J.-Y. Choi. Designing Reversibility-Enforcing Supervisors of Polynomial Complexity for Bounded Petri Nets through the Theory of Regions. In *ICATPN 2006, LNCS 4024*, pages 322–341, 2006.

[16] A. Rozinat and W. M. P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In *Business Process Management Workshops, LNCS 3812*, pages 163–176, 2005.

[17] K. Rudie and S. Lafortune. Minimal Communication in a Distributed Discrete-Event System. In *Trans. on Automatic Control 48(6), IEEE*, pages 957–975, 2003.

[18] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[19] J. M. E. M. v. d. Werf, B. F. v. Dongen, C. A. J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. In *Petri Nets 2008, LNCS 5062*, pages 368–387, 2008.