Variants of the Language Based Synthesis Problem for Petri Nets

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

Robert Lorenz
Department of Computer Science
University of Augsburg
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. Practitioners in industry are in need of a library of proper adaptations of the standard synthesis methods offering solutions for various typical requirements in practice. Having this in mind, we in this paper survey variants of the classical language based synthesis problems and develop respective solution algorithms. The presented methods may be seen as a large repertoire of synthesis procedures covering a lot of typical settings where synthesis is applicable.

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 [13]. Similar problems also occur in the domains of business process design (not only restricted to the well-known field of process mining [20, 5]), 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 [12, 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 [12], place/transition nets (p/t-nets) [2] and inhibitor nets [15]. 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. As languages, finite or infinite sets of scenarios given by occurrence sequences, step sequences [9, 1, 2] or partially ordered runs [14, 6] can be considered. In this paper we restrict ourselves to synthesis of a Petri net from a language. 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, 14]):

→ 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, first a net is constructed from $\mathcal{L}$.

→ The construction starts with the transitions T taken from the action names of $\mathcal{L}$.

→ The behavior of the net is restricted by the addition of places (including their connections to transitions of the net and their initial markings).

→ 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 transitions 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. [9, 1, 2, 14, 6, 15]. 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 $N_{\mathcal{L}}$. The synthesized net N is some finite representation of $N_{\mathcal{L}}$.

$\rightarrow$ 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.

Recently, we examined the detailed analogies and differences in synthesis approaches based on regions of languages. We identified two different types of regions, called *transition regions* and *token flow regions*, and two different principles of computing from the (infinite) set of all regions a finite Petri net representing the saturated feasible net, namely the *separation computation* and the *basis computation* [15]. 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 [15]. The framework has four dimensions defining a synthesis setting: *Petri net class, language type, region type and computation principle* (see Figure 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 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 consider variants of the classical language based 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 applications of Petri net synthesis in practice. We formulate six main 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. Although the motivations come from practical examples we still restrict ourselves to formal problems that have to be further tuned to practical applicability. In particular, runtime and memory consumption of some of the algorithms may be problematic in large realistic settings.

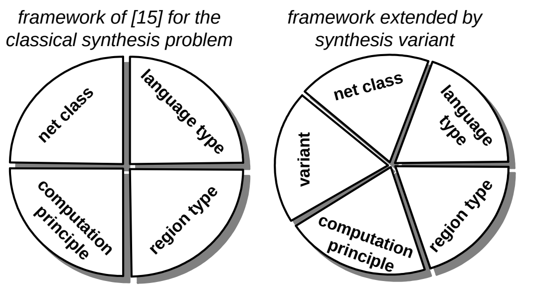

Figure 1. Synthesis framework.

The considered variants can in most cases be solved by appropriate modifications of the solution algorithms for the classical synthesis problem. We exemplarily develop algorithms to solve the variants 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). For each variant, we highlight the main ideas of the two computation principles such that, similarly as for the classical synthesis problem (see [15]), the solution algorithms in our standard synthesis settings 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 Figure 1, considering different variants of the classical synthesis problem may be seen as a fifth dimension added to the synthesis framework presented in [15]. Note that the example settings considered in this paper are 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 [15]). In the case of occurrence sequences an overview how to deal with infinite languages is described in [9] and in [6] a synthesis approach for infinite partial languages is shown (see the last paragraph of the example to Problem 1 in the Appendix for an outlook on infinity). But anyway for typical applications of language based Petri net synthesis the finite case is often sufficient, since most system specifications occurring in practice define finite languages.

Altogether, we present a broad survey of variants of the classical language based synthesis problem. Each variant, although presented in our example settings, is discussed in the context of the general framework shown in Figure 1 by highlighting the crucial ideas of the two computation principles. The variants presented in this paper are either not solved in the literature so far, or they are only solved in one specific synthesis setting (in contrast to the general discussion in this paper considering the ideas of both computation principles in the example setting as far as possible and supporting the adaptation to other net classes, language types and region types). The following table (for details see the respective sections) shows by references which variants have already been discussed in literature in some specific setting, where we only distinguish the computation principle used (also references where a similar problem is discussed in literature possibly in some other context independently from region theory are shown in the column "similar prob. lit."). The check marks indicate which variants are solved in this paper applying the respective computation principle

Variant	separation comp.		basis comp.		similar prob.
	lit.	here	lit.	here	lit.
Bound	-	✓	[8]	✓	-
Version 1	-	✓	-	✓	-
Version 2	-	✓	-	✓	[17]
Place Bound	-	✓	-	-	[7, 20]
Version 1	-	✓	-	-	[7]
Version 2	[9]	✓	-	✓	[20]
Identifying States	-	✓	-	✓	-
Version 1	-	✓	-	✓	[2]
Best Upper Approx.	-	✓	[9]	✓	-
Best Lower Approx.	-	✓	-	(✓)	-
Version 1	-	✓	-	(✓)	-
Opti. Min. Weights	-	✓	-	-	[20]
Version 1	-	✓	-	-	-
Version 2	-	✓	-	-	[7, 11]

The paper is structured as follows: The classical synthesis problem is explained in our standard settings with both computation principles in Section 2. In Section 3, solution algorithms for the considered variants of the classical problem are developed for separation and basis computation. To support the reviewing procedure, we give additional information in an Appendix which is helpful but not necessary for understanding the paper. For the interested reader, we also provide the Appendix as a technical report on our homepage (http://www.informatik.ku-eichstaett.de/mitarbeiter/mauser/techreports/variants.pdf). In the Appendix, the two algorithms to solve the classical synthesis problem are presented in detailed pseudo-code and for the classical problem as well as for each considered 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.

2 Classical Synthesis Problem

We start 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 $\sigma = t_1 \dots t_n \in T^*$, $n \in \mathbb{N}$, 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 $m_1, \dots, m_n$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} m_n$. Given a marked p/t-net (N, m_0), 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 $t_1 \dots t_n \in \mathcal{L}(N, m_0)$ then each proper *prefix* $t_1 \dots t_i$, $i < n$, (denoted by $t_1 \dots t_i < t_1 \dots t_n$) 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 standard 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 [9, 5] (see the Appendix for an example).

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, in the standard synthesis setting of this paper, we apply the region type called *transition region* for computing places of the synthesized net. A transition region directly specifies the arc weights of arcs connected to the place defined by a region and the initial marking of the place defined by a region [15, 5]. Denoting $T = \{t_1, \dots, t_n\}$, a *region* of $\mathcal{L}$ is a tuple $\mathbf{r} = (r_0, \dots, r_{2n}) \in \mathbb{N}^{2n+1}$ 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 $\mathcal{L}$ [9, 5].

The set of regions can be characterized as the (infinite) set of non-negative integral solutions of a homogenous linear inequality system $\mathbf{A}_\mathcal{L} \cdot \mathbf{r} \geq \mathbf{0}$ [5] 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 $wt \in \mathcal{L}$. Of course such duplicate rows are omitted from $\mathbf{A}_\mathcal{L}$.

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 [9, 5, 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 wrong continuations wt. Of course in this case only one such wt has to be considered. 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 [19]. 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 methods of Khachiyan (ellipsoid method) and Karmarkar (interior point method) need only polynomial runtime [19]. The Simplex algorithm is exponential in the worst case, but probabilistic and experimental results [19] show that it has a fast average runtime.

The final synthesis algorithm **Algorithm 1** to solve Problem 1 works as follows [5] (see the Appendix for detailed pseudo-code): 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 [9, 5] (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, 5]. This basic synthesis procedure is optimized by the following principle: For not yet considered 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.

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 $\lambda_1, \ldots, \lambda_n \geqslant 0$ [19]. 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 is of reasonable size for most practical examples. We call the elements of this set *basis regions*. It is shown in [9, 5] 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 [5]) and test whether $\mathcal{L}(N, m_0) \subseteq \mathcal{L}$ (performance: $|\mathcal{L}(N, m_0)| \approx |\mathcal{L}|$ = size of input), 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 [5] (see the Appendix for detailed pseudo-code): 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.

3 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 can be seen as a large repertoire of synthesis approaches applicable in various practical settings.

3.1 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 (see the Appendix for an example). This variant is useful in practice 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 [8, 11] (see e.g. [10, 16, 4, 11] 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 [8] 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. For example, if $\mathcal{L} = \{a, ab, abc, b\}, \mathcal{L}' = \{a, ab, abc, b, ba\}$, the $\mathcal{L}'$-wrong continuation bac cannot be separated, but its prefix $ba \notin \mathcal{L}$ can be separated.

Version 1: In practical applications an upper bound $\mathcal{L}'$ is often 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 2: 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. Examples for conformance measures are shown in [17]. In the case μ is monotonic for upper approximations of the language in the sense that for $\mathcal{L} \subseteq \mathcal{L}(N, m_0)$, $\mathcal{L} \subseteq \mathcal{L}(N', m_0')$, $\mathcal{L}(N, m_0) \subseteq \mathcal{L}(N', m_0')$ 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.

3.2 Place Bound Variant

A crucial requirement in practice is synthesizing compact, manually interpretable reference models [20]. 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 (see the Appendix for an example).

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 [7]. Although, as stated in [7], the approach in [7] 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: $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$, $i \in \{1, \ldots, b\}$ | $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$, $i \in \{1, \ldots, b\}$, $wt \in WC$ | $\sum_{i=1}^{b} s_{wt,i} \leq b - 1$, $wt \in WC$ | $\mathbf{r}^i \in \mathbb{N}^{2n+1}$, $i \in \{1, \ldots, b\}$ | $k \in \mathbb{N}$ | $s_{wt,i} \in \{0, 1\}$, $i \in \{1, \ldots, b\}$, $wt \in WC$. 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 [19] (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, but in practice they are often fast. There are also heuristics to compute approximate solutions in polynomial time. 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 1: 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 [7], 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: $\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r}^i \geq \mathbf{0}$, $i \in \{1, \ldots, b\}$ | $-k \cdot s_{wt,i} + \mathbf{b}_{wt} \cdot \mathbf{r}^i < \mathbf{0}$, $i \in \{1, \ldots, b\}$, $wt \in WC$ | $\sum_{i=1}^{b} s_{wt,i} \leq b - 1$, $wt \in WC$ | $k \cdot z_i - \sum_{j=0}^{2n} r_j^i \geq 0$, $i \in \{1, \ldots, b\}$ | $\mathbf{r}^i \in \mathbb{N}^{2n+1}$, $i \in \{1, \ldots, b\}$ | $k \in \mathbb{N}$ | $z_i, s_{wt,i} \in \{0, 1\}$, $i \in \{1, \ldots, b\}$, $wt \in WC$ | $min! \sum_{i=1}^{b} z_i$. 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 2: 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 [9]. 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.

3.3 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 (example in the Appendix). Such information can be integrated in synthesis methods.

Problem 4 (Identifying States Variant)**.** *Given $\mathcal{L}$ and pairwise disjoint $\mathcal{L}_1, \ldots, \mathcal{L}_l \subseteq \mathcal{L}$, decide whether there exists a p/t-net (N, m_0) fulfilling $\mathcal{L}(N, m_0) = \mathcal{L}$ such that the occurrence of each $\sigma \in \mathcal{L}_j$ leads to the same marking for all $j \in \{1, \ldots, l\}$, 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 $j \in \{1, \ldots, l\}$. Then for each $\sigma \neq \sigma_j$, $\sigma \in \mathcal{L}_j$, add two rows $\mathbf{s}_\sigma = (s_{\sigma,0}, \ldots, s_{\sigma,2n})$ 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 $p_\mathbf{r}$:

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

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

Version 1: 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].

3.4 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 having the least net language (w.r.t. set inclusion) larger than the specified language [9] (example in the Appendix). This in particular shows that such least net language exists uniquely [9, 5].

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 $(\forall (N' m_0') : (\mathcal{L}(N, m_0) \setminus \mathcal{L}(N', m_0') \neq \emptyset) \Longrightarrow (\mathcal{L} \nsubseteq \mathcal{L}(N', m_0')))$.*

As shown in Section 2, 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). For example, given $\mathcal{L} = \{b, a, aa, aab\}$, the wrong continuation ab cannot be separated, but the sequences aba and abb can be separated. 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 $\mathcal{L} \subseteq \mathcal{L}(N', m_0')$.

This adaptation of Algorithm 1 of course still solves the classical synthesis problem. Consequently it may be problematic for infinite languages, since there are examples in literature, e.g. general context-free languages [9], where Problem 5 is solvable by computing basis regions, but the classical synthesis problem is undecidable.

3.5 Best Lower Approximation

Although upper approximations are more common, there are also practical 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. A best lower approximation is a lower approximation having maximal behavior in the sense that no other lower approximation includes a larger number of occurrence sequences specified by the language (see the Appendix for an example). 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.

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 and in doing so store all wrong continuations that cannot be separated. The computed net is the starting point. It only remains to separate the stored wrong continuations. For this adapt Algorithm 1 by substituting WC by the set of stored wrong continuations. Apply this adaptation 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 algorithm is more efficient because the set of stored wrong continuations is usually small.

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 1: 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.

3.6 Optimization

As stated before, a major challenge in Petri net synthesis is the creation of concise, readable nets. 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 [7] (example in the Appendix). 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 [7, 5, 20].

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 1: 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 2: 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 [11]. 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 [4, 18, 16].

4 Conclusion

We also examined several further interesting variants of the synthesis problem. We in particular considered the problems listed in the following (see the Appendix for examples). The first five problems are simple modifications of the classical synthesis problem and can be solved straightforwardly. In the case of the last two problems, modularity and simplification of synthesis, so far there are no general satisfying solutions and still theoretical work on these topics has to be done. Together with the open questions concerned with optimization in synthesis (Subsection 3.6) these problems are major challenges for future research in Petri net synthesis. In the view of the application of synthesis in real industrial applications modularity, simplification and optimization determine key success criteria.

Predefined Places/Net: Prior to the actual design of a system model, often some information about certain resources or a complete model of some part of the system (e.g. a plant that has to be controlled or a known part of a business process) are already provided. Formally, this means that some places of the Petri net are already (partly) predefined and have to be regarded by a synthesis algorithm.

Set of Languages: Instead of taking an individual language as input, one can specify a finite set of languages allowing for some tolerance or some uncertain information. Then the problem is to decide whether there is a net having the behavior specified by one of the languages in the set. An interesting generalization of this problem is to consider an infinite set of languages represented by some finite representation yielding a problem similar to [3].

Specific Requirements: Many properties for synthesized nets can be encoded in the inequality systems of the synthesis algorithms (similar as in [7, 20]), 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 (important in the field of business process design). Note that constraints yielding inhomogeneous systems cause problems [5], because on the one hand the standard approach to compute basis regions is no more applicable (but possibly some adapted techniques), and on the other hand rational solvers cannot any more be used but integer solvers have to be applied.

Bounded Nets: In literature often synthesis of bounded nets is considered [1, 14, 5] (also relaxed versions of boundedness [10]). For applications also bounds by certain values may be reasonable, e.g. if the places model resources.

Distributable Nets: Since Petri nets are interesting for the modeling of distributed systems, synthesis of so called distributable nets [10, 8, 11] has practical relevance.

Modularity: A major problem in many applications of synthesis is modularity. Languages occurring in practice can be large, such that the considered inequality systems may become unsolvable large. Therefore, adequate approaches to divide a language into modular parts and to reasonably apply synthesis to the smaller parts are desired [11].

Simplification: Variants simplifying the synthesis procedure are of interest. Such approaches are mostly heuristic and cannot be formulated as exact problems. We are in particular interested in simplifications improving the performance of the algorithms and supporting the generation of clear, small and concise models. The field of process mining [20, 5] is a well-known example where simplified synthesis methods play an important role.

Although major language based synthesis problems are covered by the presented variants, the overview given is incomprehensive. There are of course further possible variants. Most variants discussed in the literature so far, are mentioned in the paper.

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] E. Badouel and P. Darondeau. The Synthesis of Petri Nets from Path-Automatic Specifications. *Inf. Comput.*, 193(2):117–135, 2004.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

Appendix

Algorithm 1

```
$solvable \leftarrow true$
$\mathbf{A}_{\mathcal{L}} \leftarrow EmptyMatrix$
for all $wt \in \mathcal{L}$ do
    $\mathbf{a}_{wt} \leftarrow EmptyVector$
    for $i = 0$ to $2n$ do
        if $i = 0$ then
            $a_{wt,i} \leftarrow 1$
        end if
        if $i \in \{1, \dots, n\}$ then
            $a_{wt,i} \leftarrow [w](t_i)$
        end if
        if $i \in \{n+1, \dots, 2n\}$ then
            $a_{wt,i} \leftarrow -[wt](t_{i-n})$
        end if
    end for
    if $\mathbf{A}_{\mathcal{L}}.notContainsRow(\mathbf{a}_{wt})$ then
        $\mathbf{A}_{\mathcal{L}}.addRow(\mathbf{a}_{wt})$
    end if
end for
$WC \leftarrow \mathcal{L}.getWrongContinuations()$
$(N, m_0) \leftarrow (\emptyset, \mathcal{L}.getAlphabet(), \emptyset, \emptyset)$
for all $wt \in WC$ do
    if $wt.isOccurrenceSequenceOf((N, m_0))$ then
        $\mathbf{b}_{wt} \leftarrow EmptyVector$
        for $i = 0$ to $2n$ do
            if $i = 0$ then
                $b_{wt,i} \leftarrow 1$
            end if
            if $i \in \{1, \dots, n\}$ then
                $b_{wt,i} \leftarrow [w](t_i)$
            end if
            if $i \in \{n+1, \dots, 2n\}$ then
                $b_{wt,i} \leftarrow -[wt](t_{i-n})$
            end if
        end for
        $\mathbf{r} \leftarrow Solver.getIntegerSolution(\mathbf{A}_{\mathcal{L}} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}, \mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0})$
        if $\mathbf{r} \neq null$ then
            $p_{\mathbf{r}} \leftarrow \mathbf{r}.correspondingPlace()$
            $(N, m_0).addPlace(p_{\mathbf{r}})$
        else
            $solvable \leftarrow false$
        end if
    end if
end for
return $[solvable, (N, m_0)]$
```

Algorithm 1: Solves Problem 1

Algorithm 2

```
$\mathbf{A}_{\mathcal{L}} \leftarrow EmptyMatrix$
for all $wt \in \mathcal{L}$ do
    $\mathbf{a}_{wt} \leftarrow EmptyVector$
    for $i = 0$ to $2n$ do
        if $i = 0$ then
            $a_{wt,i} \leftarrow 1$
        end if
        if $i \in \{1, \dots, n\}$ then
            $a_{wt,i} \leftarrow [w](t_i)$
        end if
        if $i \in \{n+1, \dots, 2n\}$ then
            $a_{wt,i} \leftarrow -[wt](t_{i-n})$
        end if
    end for
    if $\mathbf{A}_{\mathcal{L}}.notContainsRow(\mathbf{a}_{wt})$ then
        $\mathbf{A}_{\mathcal{L}}.addRow(\mathbf{a}_{wt})$
    end if
end for
$BasisRegions \leftarrow \mathbf{A}_{\mathcal{L}}.getBasisRegions()$
$(N, m_0) \leftarrow (\emptyset, T, \emptyset, \emptyset)$
for all $\mathbf{r} \in BasisRegions$ do
    $p_{\mathbf{r}} \leftarrow \mathbf{r}.correspondingPlace()$
    $(N, m_0).addPlace(p_{\mathbf{r}})$
end for
$\mathcal{L}(N, m_0) \leftarrow (N, m_0).getOccurrenceSequences()$
$solvable \leftarrow \mathcal{L}(N, m_0).isIncludedIn(\mathcal{L})$
return $[solvable, (N, m_0)]$
```

Algorithm 2: Solves Problem 1

Example: Problem 1

In the 1990s it was recognized on a broad front that requirements engineering the elicitation, documentation and validation of requirements - is a fundamental aspect of software development and requirements engineering emerged as a field of study in its own right. Scenarios, firstly introduced by Jacobson's use cases, proved to be a key concept for writing system requirement specifications. Important advantages of using scenarios in requirements engineering include the view of the system from the viewpoint of users, the possibility to write partial specifications, the ease of understanding, short feedback cycles, the possibilities to directly derive test cases and the possibility to derive scenarios from log files recorded by information systems. Modeling software systems by means of scenarios received much attention over the past years. Several methodologies to bridge the gap between the scenario view of a system and state-based system models, which are closer to design and implementation, have been proposed (see e.g. [13]). Using scenarios is not only useful in software engineering to specify the requirements of a system but also in other application domains. The main reason for modeling scenarios of a system is that the instance level of scenarios is the simplest and most intuitive modeling concept. We assume that in many settings the domain experts know scenarios of the system to be modeled better than the system as a whole.

Therefore, a typical situation is that the behavior of some software system, hardware system or business process is specified by scenarios. In this general setting, an interesting, fully-automated approach to derive an integrated state-based model of the system from the given specification is language based Petri net synthesis. The classical language based synthesis problem asks whether there is a net having the behavior given by the specified scenarios and in the positive case it requires the construction of such net. In the simplest setting a scenario is now given by a finite sequence of events, where each event models the occurrence of some action given by a transition of the net (actions may occur more than once).

Given the possible actions a, b, c, d, e, it may be specified the set of scenarios $\{abbe, acde, adce\}$. Since the set of occurrence sequences of a Petri net is always prefix closed, the resulting specification is given by the prefix closure $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$. Then, the first question is whether there is a p/t-net (N, m_0) having $\mathcal{L}$ as its set of occurrence sequences. Since this is true, a synthesis algorithm also has to compute a net (N, m_0) with $\mathcal{L}(N, m_0) = \mathcal{L}$. Such net is for instance given by $(N, m_0) = (\{p_1, p_2, p_3, p_4\}, \{a, b, c, d, e\}, W, m_0)$, where the non-zero values of W are given by $W(p_1, a) = 1$, $W(a, p_2) = 2$, $W(p_2, b) = 1$, $W(p_2, c) = 2$, $W(a, p_3) = 2$, $W(p_3, b) = 1$, $W(p_3, d) = 2$, $W(b, p_4) = 1$, $W(c, p_4) = 1$, $W(d, p_4) = 1$, $W(p_4, e) = 2$ and $m_0(p_1) = 1$, $m_0(p_2) = m_0(p_3) = m_0(p_4) = 0$. But this solution is of course not unique. The place p_2 could for example be replaced by the two places p_5, p_6 with $W(p_5, b) = 1$, $W(p_5, c) = 2$, $m_0(p_5) = 2$, $W(a, p_6) = 1$, $W(b, p_6) = 1$, $W(p_6, b) = 1$, $W(p_6, c) = 1$, $m_0(p_6) = 0$ or by the place p_7 given by $W(a, p_7) = 4$, $W(p_7, b) = 2$, $W(p_7, c) = 4$, $m_0(p_7) = 0$. A net containing all these places is also a solution. In fact, there are infinitely many nets solving Problem 1. In particular, each linear combination of feasible places yields a feasible place which can be added to a solution net, e.g. $p_7 = 2 * p_2$.

If we shorten the third scenario and consider the set $\{abbe, acde, adc\}$, i.e. $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc\}$, we get the situation that there is no p/t-net (N, m_0) with $\mathcal{L}(N, m_0) = \mathcal{L}$. In this case a synthesis algorithm not necessarily computes a net.

Finally, it remains to mention that the considerations on the classical synthesis problem presented in this paper are restricted to finite languages. Synthesis from finitely represented infinite languages requires some additional considerations. In the case of languages of occurrence sequences considered in our example synthesis setting, an overview how to deal with certain classes of infinite languages is described in [9]. Namely, semi-linearity of the so called "right derivatives" of a class of infinite languages can be exploited to enable the computation of a finite set of basis regions similarly as in Algorithm 2. This semi-linearity property is e.g. fulfilled by regular and context-free languages. For a separation computation in the case of an infinite language usually some kind of basis computation has to be performed in advance to get finite inequality systems. Semi-linearity of the so called "complements of the right derivatives" then potentiates considering only finitely many wrong continuations. This semi-linearity property is e.g. fulfilled by regular and deterministic context-free languages. Similar considerations are also valid for step languages. Concerning languages of partially ordered runs, we show in [6] a way to deal with infinite languages represented by "regular LPO-expressions". Namely, it is shown that an iteration operator can be encoded by a finite number of inequalities. More general infinite languages of partially ordered runs are discussed in the workshop paper "Towards Synthesis of Petri Nets from General Partial Languages" presented by Lorenz at the AWPN 2008 in Rostock. The principles sketched in this paragraph can basically be used for an adaptation of synthesis algorithms for finite languages to respective classes of infinite languages. But there are still several open problems concerned with synthesis from infinite languages which is a field of our current research.

Example: Problem 2

When specifying the scenario behavior of a system, sometimes this is not exactly possible. Such situation may arise due to design alternatives or design latitude for the system, due to incomplete information about the system or due to uncertainty w.r.t. the behavior of the system. This is particularly relevant in early design phases when the information about the behavior of the system is often vague and one aims to model a prototype of the system. In these cases, it may be desirable to specify some range of tolerance for the behavior of the system. That means, one is interested in specifying to the best of ones knowledge the minimal and the maximal possible behavior of the system by two language specifications $\mathcal{L} \subseteq \mathcal{L}'$. It is searched for a system model exhibiting at least the scenarios of $\mathcal{L}$ and at most the scenarios of $\mathcal{L}'$.

It is stated in [8] that this bound problem is relevant in theory and practice of *controller synthesis*, since this is a typical situation where there is some latitude for the behavior of the target system. In the simplest case the language bounds specify the minimal and maximal expected behavior of a plant given by a net (N_p, m_p) fulfilling $\mathcal{L} \subseteq \mathcal{L}(N_p, m_p)$. It is searched for a net (N_c, m_c) controlling this plant such that $\mathcal{L} \subseteq \mathcal{L}((N_p, m_p) \times (N_c, m_c)) \subseteq \mathcal{L}'$, where $(N_p, m_p) \times (N_c, m_c)$ is the net obtained by amalgamating (N_p, m_p) and (N_c, m_c) on transitions. This problem is solved by solving the bound variant of the synthesis problem. If it has a positive answer, the computed net (N, m_0) is an appropriate controller, otherwise there exists no controller.

If we for example consider $\mathcal{L} = \{a, ab, abb, ac, acd, acde, ad, adc\} \subseteq \mathcal{L}' = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ ($\mathcal{L}'$ is the language from the example to Problem 1), the bound variant of course has a positive answer, since Problem 1 for $\mathcal{L}'$ has a positive answer. All nets solving Problem 1 for $\mathcal{L}'$ including e.g. the net (N, m_0) from the example to Problem 1 are solution nets to the considered bound problem. But the bound problem has additional solution nets. If we for instance omit (i.e. set weight to zero) the arc from transition b to the place p_4 in the net (N, m_0), the resulting net has $\mathcal{L} \cup \{adce\} \subset \mathcal{L}'$ as its set of occurrence sequences and thus also solves the bound problem. Note that there is no p/t-net having $\mathcal{L}$ or $\mathcal{L} \cup \{abbe\} \subset \mathcal{L}'$ as its set of occurrence sequences.

If we consider the case that $\mathcal{L}$, $\mathcal{L}'$ specify the minimal and maximal expected behavior of a plant (N_p, m_p) fulfilling $\mathcal{L} \subseteq \mathcal{L}(N_p, m_p)$, then to control the plant we just have to add the places of a solution net for the respective bound problem to (N_p, m_p), e.g. the places p_1, p_2, p_3 and p_4 of (N, m_0). This works for every plant, but depending on (N_p, m_p) some of the places may be unnecessary, e.g. if (N_p, m_p) already contains p_2 or contains p_5, p_6 (see the example to Problem 1), adding p_2 is redundant.

Example: Problem 3

Important aims in many application fields of formal modeling with Petri nets are automatical analysis of the models by algorithms, simulation of the models by respective engines or even final implementation of the models e.g. by translating them to program code or hardware building blocks. Due to performance issues in these cases the size of the models must typically not exceed a certain upper limit or range. There are also applications in which Petri net models are not only intended to be automatically processed, but also manual inspection and analysis of the models is required. In such situation a very important goal is to generate models which are not too complex, i.e. restricted in terms of its number of components, because practitioners and analysts in industry are interested in controllable and interpretable reference models, which can quickly be understood also by domain experts or managers unexperienced in modeling. In particular, if the models are automatically generated by synthesis, readability is crucial to allow fine tuning and maintenance of the models by hand.

As a consequence, a typical requirement for the synthesis of Petri net models is an upper bound for the size of the models. While the number of transitions of the model is given by the number of activities in the behavioral specification, i.e. in the language, a natural possibility is to restrict the number of places of the model. That means, besides the language specification, also an upper bound b for the number of places is specified. Such bound may often depend on the size of the system to be modeled, which can e.g. be measured by the number of activities of the system (occurring in the specified language) or by the size of the specified language.

If we consider the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ from the example to Problem 1 and the bound $b = 4$, the bound problem is solvable and the net (N, m_0) from the example to Problem 1 is a solution net having 4 places. Replacing in (N, m_0) the place p_2 by the two places p_5, p_6 (see the example to Problem 1) yields a net which solves the classical synthesis problem w.r.t. $\mathcal{L}$ but not the bound problem w.r.t. $\mathcal{L}$ and $b = 4$, because it has 5 places. If we consider the language $\mathcal{L}$ and the bound $b = 3$, the bound problem has a negative answer, i.e. there is no net with 3 or less places having the set of occurrence sequences $\mathcal{L}$.

Example: Problem 4

Many kinds of specification techniques allow to model explicit information about system states (see e.g. UML). A language only allows to specify the scenario-behavior of a system. Scenarios describe the observable behavior of a system, i.e. the occurrence of activities of the system, on the simplest and most obvious level of single instances of system executions (without considering branching behavior or system states). Therefore, scenarios represent the most intuitive and clear view on a system by users and can easily be specified. A motivation given in the example to Problem 1 for considering language based synthesis is to support the step of translating a user oriented scenario specification to an implementation oriented state based model of a system. A typical situation here is that a user not only knows the scenarios of the system but a user often also has partial implementation information about states of the system. Then, the user specification serving as the input for a synthesis algorithm comprises both a language $\mathcal{L}$ and information about states which can be given by indicating which elements of the language yield the same states, i.e. by specifying pairwise disjoint subsets $\mathcal{L}_1, \ldots, \mathcal{L}_l \subseteq \mathcal{L}$.

Note that if two occurrence sequences v, w of $\mathcal{L}$ are specified to entail the same state ($v, w \in \mathcal{L}_i$ for some i), then the follow-up behavior of a system fulfilling the specification is the same for both occurrence sequences, since it only depends on the reached system state. That means, the identifying states problem can only have a solution if within $\mathcal{L}$ the residual behavior after each of the two occurrence sequences v, w is specified to be equal. The residual behavior of w within $\mathcal{L}$ is given by $\{x \mid wx \in \mathcal{L}\}$.

A typical example for specifying a system state is to determine a unique final state of the system. Given the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ from the example to Problem 1 one can specify that the three maximal occurrence sequences $\mathcal{L}_1 = \{abbe, acde, adce\}$ all lead to the same state. This identifying states problem is solvable and the net (N, m_0) from the example to Problem 1 is a solution net. After the occurrence of each of the occurrence sequences in $\mathcal{L}_1$ all places of N are empty, i.e. the three sequences yield the same marking. Adding the place p_6 to (N, m_0) (see the example to Problem 1) yields a net which solves the classical synthesis problem w.r.t. $\mathcal{L}$ but not the identifying states problem w.r.t. $\mathcal{L}$ and $\mathcal{L}_1$, because after the occurrence of $abbe$ there is one token in p_6, while after the occurrence of $acde$ and $adce$ the place p_6 is unmarked. If we consider the language $\mathcal{L}$ and want to identify the states of the occurrence sequences $\mathcal{L}_1$ and additionally the occurrence sequences $\mathcal{L}_2 = \{ac, ad\}$, the identifying states problem has a negative answer, since the specified residual behavior of ac is $\{d, de\}$ and the specified residual behavior of ad is $\{c, ce\}$. Such different residual behavior is not possible if both occurrence sequences yield the same marking.

Example: Problem 5

The synthesis problems considered so far have been decision problems. If the problems have a positive answer, the solution algorithms presented compute witness nets. In the case the problems have a negative answer, nothing is required for the computed nets. But for practical applications of synthesis algorithms the main focus usually lies in the construction of a system model from a given specification, not in the decision of the synthesis problem. That means, applications typically require the computation of a net, whether or not some synthesis problem has a positive answer. The previous synthesis problems do not require the computation of a net being a reasonable system model, if the problems have a negative answer. Actually, in these cases they do not even require computing a net at all. But specifications which cannot exactly be fulfilled are not seldom in realistic settings. The question here is which kind of net should be computed if a synthesis problem has a negative answer. This may depend on a concrete application context. An appropriate possibility to also compute a reasonable system model in the case an exact synthesis solution is not possible is to consider (best) approximations. Regarding the classical synthesis problem, Problem 1, a natural synthesis question focusing on computing a reasonable net (without posing a decision question) in all possible situations is to construct a Petri net whose set of occurrence sequences in some sense is a best approximation to the given language (among all Petri net languages). Then, if the classical synthesis problem has a positive answer still an exact solution is computed and in the negative case an in some sense best solution candidate is computed.

In Petri net theory (best) upper approximations have nice properties. The infinite saturated feasible net represents a best upper approximation by a Petri net to the given language. We consider the problem of generating a finite net having the least net language w.r.t. set inclusion larger than the specified language (such language is unique), i.e. a finite best upper approximation to the language by a Petri net. Synthesizing an upper approximation from a behavioral specification given by a language is in many cases reasonable. If the language is explicitly specified by an expert, one can assume that it usually contains only real desired behavior of the target system, since in this case there is no danger of noise in the specification and through reviewing the specification it should be possible to avoid the specification of wrong behavior. Thus, the specified language should definitely be included in the language of the synthesized model. The construction of a best upper approximation ensures that only such additional behavior is added to the synthesized net, which is necessary to create a Petri net model. A best upper approximation can be seen as a natural completion of the specified language by a Petri net. Generating such a net with more behavior than specified by the language is usually a reasonable approach. The reason for this is that in practice even in high quality system specifications some desired system behavior may have been forgotten. This can have several reasons and can in general not be detected by reviewing or similar inspections of the language. That means, the specified language may often be incomplete. In some application fields there are additional reasons justifying approximate solutions and in particular upper approximations, e.g. in process mining [20, 5]. Besides, also from a pure theoretical point of view, the synthesis of best Petri net approximations makes sense, because important decision problems are decidable for Petri nets.

If we consider the specification $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc\}$ (see the example to Problem 1) the classical synthesis problem has no exact solution. The net (N, m_0) from the example to Problem 1 is a best upper approximation by a Petri net to $\mathcal{L}$ having one additional occurrence sequence $adce$. Each net having the same behavior as (N, m_0) is also a best upper approximation and each best upper approximation is given by such net. Given $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$, the net (N, m_0) exactly solves the classical synthesis problem and therefore is a best upper approximation.

Example: Problem 6

In the example to Problem 5 it is explained that in general the synthesis of a best approximation to a specified language is reasonable. Besides upper approximations we can also consider lower approximations which are nets having only behavior specified by the given language. A best lower approximation is a lower approximation having a maximal number of occurrence sequences (such set of occurrence sequences in general is not unique). That means, the number of occurrence sequences specified by the language but not included in the behavior of the net is minimal among all lower approximations. Then again, if the classical synthesis problem has a positive answer a best lower approximation is an exact solution and in the negative case it represents an in some sense best solution candidate.

There are realistic settings in which a best lower approximation is particularly desirable, e.g. compared to a best upper approximation. This is the case if it is more problematic to compute a system model which allows behavior not specified by the language than to leave specified behavior from the synthesized model. That means, the system specification is complete in the sense that all behavior not specified in the language is faulty behavior which has to be prohibited under any circumstances. Such situation may occur in safety critical systems, such as medic systems or (embedded) traffic systems, where behavior not explicitly allowed by the specification can have fatale consequences. Also, in systems having only few (but possibly long) executions it may be very likely that all desired behavior is considered in a specification and all not specified behavior causes errors. If in such cases an implementation of the specification (by a Petri net) is not directly possible it is better to leave some of the desired behavior of the specification, i.e. to restrict the functionality of the system, than to add faulty behavior to the system. A best lower approximation reduces the desired behavior of the specification as few as possible. In general, it can even be useful to compute lower approximations, i.e. to omit some behavior of a specification from a system model, since due to human failability manual specifications may include unnecessary or even wrong behavior in particular if the specification has been developed under pressure of time and due to disturbing influences automatically generated specifications may contain noise.

If we consider the specification $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc\}$ from the example to Problem 5 the classical synthesis problem has no exact solution. The net resulting from omitting the arcs between c resp. d and p_4 from (N, m_0) (see example to Problem 1) is a best lower approximation by a Petri net to $\mathcal{L}$ having $\mathcal{L} \setminus \{acde\}$ as its set of occurrence sequences. The net resulting from adding (loop) arcs in both directions between c and p_3, i.e. $W(c, p_3) = 1$ and $W(p_3, c) = 1$, to (N, m_0) is also a best lower approximation by a Petri net to $\mathcal{L}$ having $\mathcal{L} \setminus \{adc\}$ as its set of occurrence sequences. The set of all best lower approximations is given by the set of nets having the same behavior as one of the two previous nets. Given $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$, the net (N, m_0) exactly solves the classical synthesis problem and therefore is a best lower approximation.

Example: Problem 7

As explained in the example to Problem 3, a crucial challenge for applying Petri net synthesis in industrial settings is the creation of concise, clearly readable nets. Thereby, for readability, the complexity of the single components is often more important than the number of components. If the functionality of each single component gets clear on the first glance, usually a model can nicely be analyzed and modularly looking at local parts of interest within the model is supported. Since in the Petri net synthesis approach places are the determinant for the behavior of the system, the aim is to synthesize in some sense simple places. An interesting approach here is to minimize the arc weights and the initial markings of places. On the one hand smaller numbers of tokens and arc weights are simpler to understand and on the other hand this supports zero arc weights reducing the branching of places. That means, the synthesized places exhibit few complexity in the sense that complicated token ratios are avoided and few transitions are interrelated with one place. To achieve this the construction of the places of the synthesized net is guided by a respective objective function.

Considering the specification $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$, the net (N, m_0) from the example to Problem 1 solves the minimal weights problem, since each of the four places p_1, p_2, p_3 and p_4 cannot be replaced by a combination of simpler places each having a smaller sum of arc weights and initial marking. The optimization approach ensures that such simple net as (N, m_0) is computed and no unnecessary complicated places are introduced. For instance, multiplicities of places such as p_7 instead of p_2 and other too complex places such as p_7 with the weight $W(p_7, c)$ changed from 4 to 3 or the place p_4 extended with an additional arc from e to p_4, i.e. $W(e, p_4) = 1$, are ruled out. Given the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad\}$, the net resulting from adding (loop) arcs in both directions between c and p_3, i.e. $W(c, p_3) = 1$ and $W(p_3, c) = 1$, to (N, m_0) solves the classical synthesis problem (see Example to Problem 7) but it is not optimal. The sum of the arc weights and the initial marking of the modified p_3 is 7. The same behavior can be generated by replacing the modified p_3 by the original p_3 together with the additional place p_8 defined by $W(a, p_8) = 1$, $W(c, p_8) = 1$, $W(p_8, c) = 1$, $W(p_8, d) = 1$. Both places are simpler, since the sum of the arc weights and the initial marking of p_3 is 5 and the sum of p_8 is 4. The resulting net solves the minimal weights problem. Note that the minimal weights problem does not regard the complexity of nets arising from a huge number of places, e.g. in the first example $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ the net resulting from replacing p_2 by p_5 and p_6 in (N, m_0) is also optimal and in the second example $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad\}$ the optimal net has more places (and even a higher overall sum of initial markings and arc weights of all places) than the initially proposed net. This problem is tackled in Version 2 of Problem 7.

Example: Conclusion

Predefined Places/Net: Besides a behavioral specification of a system by a language, there may already be fixed parts of a system model. This for instance happens if a system is not designed from scratch but a prior system model has to be updated or if some components of a system are already known. Then, for an actual synthesis algorithm not only the transitions are a priori given by the language but also some places (defined by their arc weights and their initial marking) are predefined. To account for this it is only necessary to check whether all predefined places are feasible. In the positive case the usual synthesis algorithms Algorithm 1 and 2 are applicable, whereas concerning Algorithm 1 the predefined places can be used to separate wrong continuations. Otherwise the problem has no solution. If we consider the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ from the example to Problem 1, there may for instance some of the places of (N, m_0) be predefined. They are feasible and it is only necessary to add the remaining places to get a synthesis solution. If a non-feasible place such as p_2 with the weight $W(p_2, b)$ changed to 2 is predefined, there is no solution net. It is also possible that some places are only partly predefined. That means one specifies a certain place, but not all parameters (arc weights, initial marking) are established. Additionally, certain properties may be constituted, e.g. the marking of the place after certain net behavior. It is searched for one respective feasible place. For this, one has to find one solution of the usual inequality system defining regions whereas some parameters are no more variables, but already fixed, and additional properties are integrated by additional inequalities. An example for such situation is some place modeling a warning lamp in a production process. Given $\mathcal{L}$, one may require a feasible place which is marked when the process is finished, i.e. after each of the maximal occurrence sequences $abbe, acde, adce$, and unmarked otherwise. That means, the place has no predefined arc weights, but it is demanded that the place is marked in every final marking of certain occurrence sequences triggering the warning and unmarked otherwise. The place p_9 given by $W(e, p_9) = 1$ and $m_0(p_9) = 0$ is such place. It is a final place of the process. Requiring such place is also interesting when interpreting $\mathcal{L}$ as the behavior of a business process.

Set of Languages: To allow for some tolerance or to consider some uncertain information in synthesis applications one can specify a finite set of alternative languages each modeling possible behavior of the target system. This problem is a refinement of the bound problem and therefore can be motivated in the same way (see example to Problem 2). In order to decide whether there is a net having the behavior specified by one of the languages in the given set, one can apply the synthesis algorithms Algorithm 1 and 2 to each of the languages. Given the set $\{\mathcal{L}, \mathcal{L}'\}$ of the two languages $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ and $\mathcal{L}' = \{a, ab, abb, abbe, ac, acd, acde, ad, adc\}$ from the example to Problem 1, the set of solution nets is given by all nets having $\mathcal{L}$ as its set of occurrence sequences, including e.g. (N, m_0), since there is no net having the behavior specified by $\mathcal{L}'$.

Specific Requirements: To increase the expressivity of language specifications one may allow to not only specify a language but additionally certain properties for the system to be modeled. A lot of such properties can be regarded by the inequality systems of the synthesis algorithms Algorithm 1 and 2. There are many requirements useful in practice, e.g. structural restrictions to nets of certain types. For example restricting arc weights by certain values can be achieved by adding inequalities limiting variables of the regions. Restricting the marking in certain system states, where a state is given by the final marking of an element of the language, can be achieved by adding inequalities limiting respective final markings given as linear combinations of the variables of the regions. A transition invariant can be defined by an equation setting an appropriate linear combination of the variables of the regions to zero. For instance, in order to synthesize a sound workflow net from $\mathcal{L} = \{a, ab, abe, ac, acd, acde, ad, adc, adce\}$, one has to require that all arc weights are at most one, one has to guarantee the construction of an initial and a final place w.r.t. the maximal occurrence sequences $abe, acde, adce$ as shown in the predefined places problem, and for each place except the initial and final place it has to be required that the final marking of the empty and the maximal occurrence sequence is zero. This synthesis problem is solvable, e.g. by the net $(N, m_0) = (\{p_1, p_2, p_3, p_4, p_5, p_6\}, \{a, b, c, d, e\}, W, m_0)$, where the non-zero values of W are given by $W(p_1, a) = 1$, $W(a, p_2) = 1$, $W(p_2, b) = 1$, $W(p_2, c) = 1$, $W(a, p_3) = 1$, $W(p_3, b) = 1$, $W(p_3, d) = 1$, $W(b, p_4) = 1$, $W(c, p_4) = 1$, $W(p_4, e) = 1$, $W(b, p_5) = 1$, $W(d, p_5) = 1$, $W(p_5, e) = 1$, $W(e, p_6) = 1$ and $m_0(p_1) = 1$, $m_0(p_2) = m_0(p_3) = m_0(p_4) = m_0(p_5) = m_0(p_6) = 0$.

Bounded Nets: An important requirement for nets in Petri net theory is boundedness. Boundedness is relevant in theory and application to avoid overflow of places, to allow the application of analysis methods for bounded nets and sometimes also for system design purposes. Also, k-boundedness of places by certain values k is useful in some settings. In our finite case, if the synthesis problem has a positive answer, the resulting nets are always bounded. In the negative case, due to the best upper approximation property, the net resulting from Algorithm 2 is bounded. It is easy to introduce boundedness also in Algorithm 1: Add for every transition t 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 $t' \in T$, $m_0(p) = max\{[w](t) \mid w \in \mathcal{L}\}$. To consider boundedness by a certain value (e.g. one-safe nets), the bound has to be integrated to the inequality systems by an additional inequality for each state defined by a final marking of an occurrence sequence of the language (as described in the last paragraph). The net (N, m_0) from the example to Problem 1 realizes the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ and is bounded by $k = 2$. There is no one-safe net having the set of occurrence sequences $\mathcal{L}$.

Distributable Nets: Petri nets allow modeling of true concurrency. Thus, an important application field of Petri nets is modeling of highly-concurrent or distributed systems. For such systems distributed architectures of Petri nets are applied. The basic Petri net modeling formalism for distributed systems are distributable nets. Distributable nets consist of distributed components, also called locations, where transitions in one component may only consume tokens from places in the same component, but they may produce tokens also in places of other components for communication. Since competitions for tokens are local, distributable nets consist of local components communicating by asynchronous message passing. Given a partition of the set of transitions of a language to distributed components, the set of feasible places of one component can be defined by ensuring the formulated restrictions for places of this component through leaving variables representing forbidden arcs (to transitions of other components) in the inequality systems. Then, the places of the different components can be computed separately using standard methods. If we consider the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ from the example to Problem 1 and consider the distributed components given by the sets of transitions $\{a\}$, $\{b, c, d\}$ and $\{e\}$, the net (N, m_0) is a distributable net solving the synthesis problem. The place p_1 belongs to the component $\{a\}$, the places p_2, p_3 to the component $\{b, c, d\}$ and the place p_4 to the component $\{e\}$. Dividing the component $\{b, c, d\}$ to two components $\{b\}$ and $\{c, d\}$ is not possible, because to model the conflict between b and c, d a place having an arc to both b and c resp. d is necessary, but such place would then belong to both components.

Modularity: In practice there are examples of very large language specifications. Automatically generated specifications can typically become very large, but in huge projects also manually developed system specifications may be large. Regarding synthesis from such languages the algorithms may run into performance problems or troubles w.r.t. the complexity of the generated nets. An approach to solve such problems is to consider smaller problem instances by dividing a language into modular parts. Basically there are three possibilities to divide a language: First, the set of transitions can be partitioned. Then, each set of the partition defines a smaller language by projecting the original language to the considered transitions. Second, the language can simply be partitioned. Third, the maximal occurrence sequences of the language can be divided. For example each maximal occurrence sequence can be decomposed to a fixed number of sequential parts. Then all first parts, all second parts, etc each define a smaller language. Heuristics to achieve a reasonable division of the language are important in all cases. The generated modular parts of the language either represent alternative behavior (second case), or sequential parts of the complete behavior (third case), or preferably concurrent components of the behavior, that have to be synchronized by additional communication (first case). Of course, also a combination of the three cases is possible. The modular parts either have disjoint transition sets (first case) or overlapping transition sets (second and third case). Given modular parts of a language, the simplest approach is to synthesize separately a net for each part. If the transition sets of the parts are overlapping, this causes label splitting, but only in a restricted way according to the modular division. If the parts model alternative behavior, a non-deterministic choice at the beginning is introduced leading to the initial marking of one of the modular nets. If the parts model sequential behavior, the first modular net builds the starting point. Then, the other modular nets have to be sequentially appended. For each possible final marking of a prior modular net, there is a non-deterministic choice to change the marking to the initial marking of the next modular net (this may cause problems with sub-states). Often adding additional connecting places to combine the sequential components is necessary here. If the parts model concurrent behavior, the modular nets are independently arranged. Possibly some communication has to be introduced, which is a challenging task. The problem of modularity has not yet received much attention (outside supervisory control [10]) and there are a lot of open questions. As an example assume the division of the maximal occurrence sequences $abbe, acde, adce$ of the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ by truncating the final e in each case. Then the first language is given by $\mathcal{L}' = \{a, ab, abb, abb, ac, acd, acd, ad, adc, adc\}$ and the second language is given by $\mathcal{L}'' = \{e\}$. Let the net synthesized from $\mathcal{L}'$ be the net (N, m_0) from the example to Problem 1 without transition e, which means that the place p_4 is replaced by a final place p'_4 with two tokens in the final marking given by $W(b, p'_4) = 1$, $W(c, p'_4) = 1$, $W(d, p'_4) = 1$ and $m_0(p'_4) = 0$. Let the net synthesized from $\mathcal{L}''$ be given by $(\{p''_4\}, \{e\}, W', m'_0)$, where $W'(p''_4, e) = 1$ and $m'_0(p''_4) = 1$. Then, to sequentially combine the two nets the initial marking is removed from the second net ($m'_0(p''_4) = 0$) and a transfer transition t, given by $W(p'_4, t) = 2$, $W'(t, p''_4) = 1$ is introduced to allow to change the final marking of the first net to the initial marking of the second net.

Simplification: Approaches to improve the performance of synthesis algorithms and to support the generation of clear, non "spaghetti"-like models are of particular relevance. These two problems are the main obstacles for applying synthesis in industrial settings interested in formal modeling with Petri nets. Since in many applications, as explained in several of the previous examples, exact synthesis is not essential, approaches simplifying the synthesis procedure w.r.t. the above problems are of interest. We only sketch some possible examples for such approaches here. Examples include: Simply neglecting some (less relevant) words or transitions of the language or simplifying cyclic behavior within the language. Adding additional concurrency to the nets (which may be reasonable according to some external information source) by modifying the inequality systems. Applying heuristics to only add the most "important" places or to neglect too complex places (having high arc weights); this is especially reasonable in combination with optimization algorithms. Simplifying places by reducing arc weights to one or by completely deleting some arcs connecting relatively independent transitions. Also, pre-processing of the language before applying a synthesis algorithm and post-processing of the synthesized net is important in this context to possibly simplify the synthesis procedure and to improve the resulting net. Concerning the language $\mathcal{L} = \{a, ab, abb, abbe, ac, acd, acde, ad, adc, adce\}$ from the example to Problem 1, one can for example leave all occurrence sequences containing b, if the action b is not so important. The resulting behavior can be reproduced by the very simple net without arc weights (larger than one) $(N, m_0) = (\{p_1, p_2, p_3, p_4, p_5\}, \{a, c, d, e\}, W, m_0)$, where the non-zero values of W are given by $W(p_1, a) = 1$, $W(a, p_2) = 1$, $W(p_2, c) = 1$, $W(a, p_3) = 1$, $W(p_3, d) = 1$, $W(c, p_4) = 1$, $W(p_4, e) = 1$,$W(d, p_5) = 1$, $W(p_5, e) = 1$ and $m_0(p_1) = 1$, $m_0(p_2) = m_0(p_3) = m_0(p_4) = m_0(p_5) = 0$. If b should not completely be omitted, we can simplify the cyclic behavior of b by deleting its repetitions. The resulting language $\mathcal{L} = \{a, ab, abe, ac, acd, acde, ad, adc, adce\}$ can be realized by the net without arc weights given by adding the transition b and connections $W(p_2, b) = 1$, $W(p_3, b) = 1$, $W(b, p_4) = 1$, $W(b, p_5) = 1$ to (N, m_0).