How to synthesize nets from languages - a survey

Robert Lorenz, Sebastian Mauser, Gabriel Juhas

Angaben zur Veröffentlichung / Publication details:

Lorenz, Robert, Sebastian Mauser, and Gabriel Juhas. 2007. "How to synthesize nets from languages - a survey." In *2007 Winter Simulation Conference, 9-12 Dec. 2007, Washington, DC, USA*, edited by Shane G. Henderson, Bahar Biller, Ming-hua Hsieh, and John Shortle, 637–47. Piscataway, NJ: IEEE. https://doi.org/10.1109/wsc.2007.4419657.

HOW TO SYNTHESIZE NETS FROM LANGUAGES - A SURVEY

Robert Lorenz
Sebastian Mauser

Department of Computer Science
Catholic University of Eichstätt-Ingolstadt
85072 Eichstätt, GERMANY

Gabriel Juhás

Faculty of Electrical Eng. and Information Technology
Slovak University of Technology Bratislava
Bratislava, SLOVAKIA

ABSTRACT

In this paper we present a survey on methods for the synthesis of Petri nets from behavioral descriptions given as languages. We consider place/transition Petri nets, elementary Petri nets and Petri nets with inhibitor arcs. For each net class we consider classical languages, step languages and partial languages as behavioral description.

All methods are based on the notion of regions of languages. We identify two different types of regions and two different principles of computing from the set of regions of a language a finite Petri net generating this language. For finite or regular languages almost each combination of Petri net class, language type, region type and computation principle can be considered to compute such a net. Altogether, we present a framework for region-based synthesis of Petri nets from languages which integrates almost all known approaches and fills several remaining gaps in literature.

1 INTRODUCTION

Synthesis of Petri nets from behavioral descriptions has been a successful line of research since the 1990s. There is a rich body of nontrivial theoretical results and there are important applications in industry, in particular in hardware design (Cortadella et al. 2000, Josephs and Furey 2002), in control of manufacturing systems (Zhou and Cesare 1993) and recently also in workflow design (van der Aalst et al. 2003, van der Aalst and Günther 2007).

The synthesis problem is the problem to construct, for a given behavioral specification, a Petri net such that the behavior of this net coincides with the specified behavior (if such a net exists). There are many different methods which are presented in literature to solve this problem. They differ mainly in the Petri net class and the model for the behavioral specification considered. On the other hand, all these methods are based on one common theoretical concept, the so called *theory of regions*. In this paper, we present an overview on region-based synthesis methods, which regard languages as behavioral specifications.

A Petri net N (of any Petri net class) has a set of *places* P and a set of *transitions* T. While transitions model events, places define pre-, post- and side-conditions of transitions. That means, places define states of the modeled system in which transitions (events) can occur and determine how the occurrence of a transition changes such states. Through places causal dependencies between transitions may be introduced. Figure 1 shows a place/transition-net, which serves as a running example in this paper.

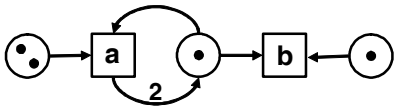

Figure 1: A place/transition-net with transitions a and b and three places. Places carry tokens determining their state. Numbers assigned to arcs from places to transitions specify the number of tokens consumed by the occurrence of a transition. Numbers assigned to arcs from transitions to places specify the number of tokens produced by the occurrence of a transition. Arc weights of value 1 are not shown.

The behavior of a Petri net model N is defined by the *language of executions* $L(N)$ generated by N. Executions are representations of observations of transition occurrences of the Petri net (for example finite sequences of transition occurrences). There are several different (formal) notions of executions of Petri nets depending on the considered net semantics, resulting in different language types. For example, for the place/transition-net shown in Figure 1 we have $L(N) = \{b, a, ab, aab, aba\}$ w.r.t. to the so called *sequential semantics*. Altogether, we formally consider the following synthesis problem w.r.t. different Petri net classes and different language types:

Given: A language L over a finite alphabet A.
Searched: A Petri net N with $L(N) = L$.

That means, we search for an exact solution of the problem. Such an exact a solution may not exist, i.e. not each language L is a *net language*. The idea of region-based synthesis common to all mentioned methods to compute such a net N is as follows: The set of transitions of this net is $T = A$. We first consider the net N having an empty set of places. This net generates each execution in L (i.e. $L \subseteq L(N)$), because there are no places restricting transition occurrences. But it generates much more executions. Since we are interested in an exact solution, we restrict $L(N)$ by adding places.

There are places p, which restrict the set of executions too much in the sense that $L \setminus L(N) \neq \emptyset$, if p is added to N. Such places are called *non-feasible (w.r.t. L)*. We only add so called *feasible* places p satisfying $L \subseteq L(N)$, if p is added to N (Figure 2). The idea of region-based synthesis is to add *all* feasible places to N. The resulting net N_{sat} is called the *saturated feasible net*. On the one hand, N_{sat} has by construction the following very nice property

(min) $L(N_{sat})$ is the smallest net language satisfying $L \subseteq L(N_{sat})$.

This is clear, since $L(N_{sat})$ could only be further restricted by adding non-feasible places. The property (min) directly implies that there is an exact solution of the synthesis problem if and only if N_{sat} is such an exact solution. Moreover, if there is no exact solution, N_{sat} is the best approximation to such a solution.

On the other hand, this result is only of theoretical value, since the set of feasible places is in general *infinite* (Figure 3). Therefore, for a practical solution, a finite subset of the set of all feasible places is defined, such that the net N_{fin} defined by this finite subset fulfills $L(N_{fin}) = L(N_{sat})$. Such a net N_{fin} is called *finite representation* of N_{sat}. In order to construct such a finite representation, in an intermediate step a feasible place is defined through a so called *region* of the given language L.

The described approach is common to all known region-based synthesis methods (see Figure 4). But these methods differ in the definition of regions (of a language) and of the finite representation N_{fin}. In this paper it is for the first time observed, that one can distinguish two types of definitions of regions and two types of definitions of finite representations, covering all known region-based synthesis methods. We show that these different types of definitions can be applied to almost each Petri net class and each language type in all four possible combinations (leading to different nets N_{fin} having the same behavior).

Summarized, the form of the synthesis problem and the solution method can be varied along the following lines: *Petri net class* (p/t-nets, elementary nets, p/t-nets with inhibitor arcs and elementary nets with inhibitor arcs), *language type* (finite or regular classical languages, trace languages and partial languages), *region type* (transition-region, token flow-region) and *finite representation type* (separating representation, basis representation).

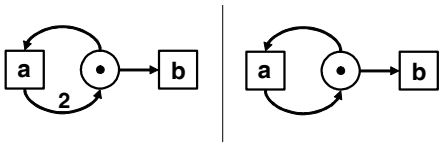

Figure 2: A feasible place (left part) and a non-feasible place (right part) w.r.t. the language $L = \{b, a, ab, aab, aba\}$ (aba is no execution of the right net).

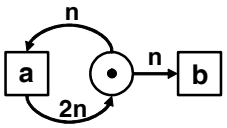

Figure 3: The shown place is feasible w.r.t. $L = \{b, a, ab, aab, aba\}$ for each integer $n \in \mathbb{N}$.

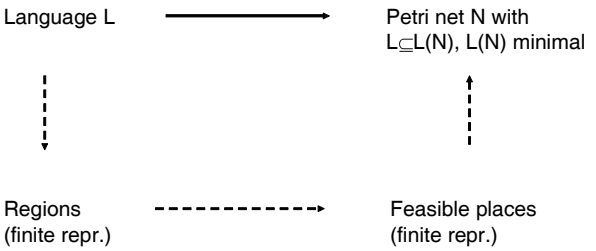

Figure 4: The approach of region-based synthesis.

The structure of the rest of the paper is as follows. In Section 2 we recall basic mathematical notations and give basic definitions of language types, Petri net classes and executions of Petri nets. In Section 3 we introduce transition-regions and token flow-regions for each mentioned combination of Petri net class and language type. We show in each case that the set of regions equals the set of non-negative integral solutions of a possibly infinite linear system of the form $\mathbf{A}_L \cdot \mathbf{x} \leqslant \mathbf{b}_L$ ($\mathbf{A}_L$ may have infinite many rows and columns). This linear system is finite for finite and for regular languages. In Section 4 we introduce the finite basis representation of the set of solutions of finite linear systems of the form $\mathbf{A}_L \cdot \mathbf{x} \leqslant \mathbf{0}$ and the finite separating representation of the set of solutions of finite linear systems of the form $\mathbf{A}_L \cdot \mathbf{x} \leqslant \mathbf{b}_L$, leading to different nets N_{fin} solving the synthesis problem. In Section 5 we give the references concerning known results stated in the Sections 3 and 4. We integrate all these existing approaches into the presented framework and identify and discuss new alternatives (given by the framework) for solving the synthesis problem. Proofs are omitted due to lack of space. All presented algorithms involve integer programming techniques. Since there are many different such techniques and we did not want to give a survey on these, we omit to give references in this direction.

Due to lack of space, we also did not add an elaboration of applications in practise (there are quite a lot) and the relation to the simulation area. Some brief comments on that can be found in the conclusion.

2 MATHEMATICAL PRELIMINARIES

In this section we briefly introduce basic mathematical notions we use in the paper. In particular, the different considered language types and Petri net classes are defined.

By $\mathbb{N}_0$ we denote the set of *nonnegative integers*, by $\mathbb{N}$ the set of positive integers. Given a function f from X to Y and a subset Z of X we write $f|_Z$ to denote the *restriction* of f to the set Z. Given a finite set X, the symbol $|X|$ denotes the *cardinality* of X. The set of all *multi-sets* over a set X is the set $\mathbb{N}^X$ of all functions $f : X \to \mathbb{N}$. Addition $+$ on multi-sets is defined as usual by $(m+m')(a) = m(a)+m'(a)$. We also write $\sum_{a\in X} m(a)a$ to denote a multi-set m over X and write $a \in m$ if $m(a) > 0$. Given a binary relation $R \subseteq X \times X$ over a set X, the symbol R^+ denotes the *transitive closure* of R. We write aRb to denote $(a,b) \in R$. A *directed graph* is a pair $(V,\to)$, where V is a finite *set of vertices* and $\to \subseteq V \times V$ is a binary relation over V, called the *set of edges* (all graphs considered in this paper are finite).

A *partial order* is a directed graph $\mathrm{po} = (V,<)$, where $<$ is a binary relation on V which is irreflexive ($\forall v \in V : v \not< v$) and transitive ($<=<^+$). Two nodes $v,v' \in V$ of a partial order $(V,<)$ are called *independent* if $v \not< v'$ and $v' \not< v$. By $\mathrm{co} \subseteq V \times V$ we denote the set of all pairs of independent nodes of V. A *co-set* is a subset $C \subseteq V$ fulfilling $\forall x,y \in C : x\,\mathrm{co}\,y$. A *cut* is a maximal co-set. For a co-set C of a partial order $(V,<)$ and a node $v \in V \setminus C$ we write $v < C$, if $v < s$ for an element $s \in C$ and $v\,\mathrm{co}\,C$, if $v\,\mathrm{co}\,s$ for all elements $s \in C$. A partial order $(V',<')$ is a *prefix* of another partial order $(V,<)$ if $V' \subseteq V$ with $(v' \in V' \wedge v < v') \Longrightarrow (v \in V')$ and $<'=<|_{V'\times V'}$. We say that such a prefix is defined by V'. To each cut C, there corresponds a prefix defined by $V' = \{v \in V \mid v < C\}$. Given two partial orders $\mathrm{po}_1 = (V,<_1)$ and $\mathrm{po}_2 = (V,<_2)$, we say that po_2 is a *sequentialization* of po_1 if $<_1 \subseteq <_2$.

Sets of executions of Petri nets are languages. We will consider classical languages, step languages and partial languages. Let A be a finite set of characters. Formally, a *(classical) language over A* is a (possibly infinite) set of finite sequences of characters from A. Such sequences describe sequential executions of Petri nets. An example of a (finite) classical language is $L = \{b,a,ab,aba,aab\}$. A *(concurrent) step over A* is a multi-set over A. A *step language over A* is a (possibly infinite) set of finite sequences of steps over A. Such sequences describe step executions of Petri nets, where actions belonging to a step are considered to be independent. An example of a (finite) step language is $L = \{(2a)b, a(a+b)\}$. For a classical language L and $w \in L$, $|w|_a$ denotes the number of a's occurring in w (for example $|aba|_a = 2$). For a step language L and $w = \alpha_1 \ldots \alpha_m \in L$, $|w|_a = \sum_{i=1}^{m} \alpha_i(a)$ denotes the number of a's occurring in w (for example $|a(a+b)|_a = 2$).

A *partial language over A* is a (possibly infinite) set of non-isomorphic (finite) so called labeled partial orders over A. A *labeled partial order* (LPO) is a triple $\mathrm{lpo} = (V,<,l)$, where $(V,<)$ is a partial order, and $l : V \to A$ is a *labeling function* with *set of labels* A. LPOs describe non-sequential computations of Petri nets. The ordering between events is interpreted as an "earlier than"-relation between events. If two events are not ordered, then they can occur concurrently (in any order and also at the same time, Figure 5).

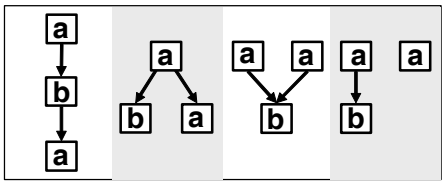

Figure 5: LPOs with three nodes and labels a and b. The first LPO (from left to right) describes the sequential computation aba, the second LPO the step computation $a(a+b)$ and the third LPO the step computation $(2a)b$. The fourth LPO describes the possibility that the sequence ab can occur concurrently to a.

The labeling function l is lifted to a subset Y of V in the following way: $l(Y)$ is the multi-set over A given by $l(Y)(a) = |l^{-1}(a) \cap Y|$ for every $a \in A$.

Classical languages and step languages can be seen as special classes of partial languages. That means a finite sequence σ of characters from A or of (concurrent) steps over A can be represented as a LPO lpo_σ. For $\sigma = a_1 \ldots a_m$, $a_i \in A$ for $i \in \{1,\ldots,m\}$, define $\mathrm{lpo}_\sigma = (V,<,l)$ by $V = \{v_1,\ldots,v_m\}$, $v_i < v_j \Leftrightarrow i < j$ and $l(v_i) = a_i$ for $i \in \{1,\ldots,m\}$. For example, lpo_{aba} is the first LPO in Figure 5. For a finite sequence $\sigma = \alpha_1 \ldots \alpha_m$ of steps over A define $\mathrm{lpo}_\sigma = (V,<,l)$ by $V = \bigcup_{i=1}^{m} V_i$, $l(V_i)(a) = \alpha_i(a)$ for $a \in A$ and $<= \bigcup_{i<j} V_i \times V_j$. For example, $\mathrm{lpo}_{a(a+b)}$ is the second LPO and $\mathrm{lpo}_{(2a)b}$ is the third LPO in Figure 5.

In the following we define different Petri net classes and their different notions of executions. We will not give the classical definitions, instead we introduce different Petri net classes by their different types of places. A Petri net consists of a set of *transitions* T and a set of places P. For any considered Petri net class, the state of a Petri net is given by a *marking* m of places, assigning to each place $p \in P$ a number $m(p) \in \mathbb{N}_0$.

Definition 1 (place) *We introduce places of place/transition-nets and elementary nets (with and without inhibitor arcs). For any Petri net class, each place has assigned an* initial marking $m_0(p)$ *determining the initial state of the Petri net. The other parameters of places*

depend on the considered net class and assign pre-, post- and side-conditions to transitions.

Place/transition nets (p/t-nets)*: A* p/t-net place $p \in P$ *is determined (besides its initial marking) through the number* $W(p,t) \in \mathbb{N}_0$ *of tokens* consumed *by (an occurrence of)* t *in* p *and the number of tokens* $W(t,p) \in \mathbb{N}_0$ produced *by* t *in* p *for each transition* $t \in T$.

Elementary nets*: An* elementary net place $p \in P$ *is a p/t-net place which satisfies* $m_0(p) \leqslant 1$, $W(p,t) \leqslant 1$ *and* $W(t,p) \leqslant 1$ *(for each* $t \in T$*).*

Nets with inhibitor arcs*: A* inhibitor net place $p \in P$ *is a p/t-net place (resp. elementary net place) which is additionally determined by so called* inhibitor values $I(p,t) \in N_0 \cup \{\omega\}$*, which serve as upper bounds for the number of tokens allowed in* p *for the occurrence of* $t \in T$.

Graphically, places are drawn as circles and transitions as squares. The initial marking $m_0(p)$ is illustrated by drawing $m_0(p)$ tokens inside p, the number $W(p,t)$ is assigned to an arrow from p to t and $W(t,p)$ is assigned to an arrow from t to p. For example the left place p in Figure 2 is defined by $m_0(p) = 1$, $W(p,a) = 1$, $W(p,b) = 1$, $W(a,p) = 2$ and $W(b,p) = 0$. A number $I(p,t)$ is assigned to an arrow from p to t which as a circle as arrowhead.

The definition of executions of Petri nets depends on the *occurrence rule* of transitions, stating in which markings a transition (or a multi-set of transitions) can occur and how these markings are changed by its occurrence.

Definition 2 (occurrence rule) *For all considered Petri net classes the change of a marking by the occurrence of transitions is defined in the same way: If a transition* t *occurs in a marking* m*, then resulting marking* m' *is defined by* $m'(p) = m(p) - W(p,t) + W(t,p)$ *for* $p \in P$*. If a multi-set of transitions* τ *occurs in* m*, then the resulting marking* m' *is defined by* $m'(p) = m(p) - \sum_{t \in \tau} \tau(t) W(p,t) + \sum_{t \in \tau} \tau(t) W(t,p)$ *for* $p \in P$*. We write* $m \xrightarrow{t} m'$ *(*$m \xrightarrow{\tau} m'$*) to denote that* t *(*τ*) can occur in* m *and that its occurrence leads to* m'*. Transitions can occur according to the following conditions:*

P/t-nets*: A transition* $t \in T$ can occur *in a marking* m*, if* $\forall p \in P : m(p) \geqslant W(p,t)$*. A multi-set of transitions* τ can occur *in* m*, if* $\forall p \in P : m(p) \geqslant \sum_{t \in \tau} \tau(t) W(p,t)$.

Elementary nets*: A transition* $t \in T$ can occur *in a marking* m*, if* $\forall p \in P : m(p) \geqslant W(p,t) \wedge m'(p) \leqslant 1$*. A multi-set of transitions* τ can to occur *in* m*, if* $\forall p \in P : m(p) \geqslant \sum_{t \in \tau} \tau(t) W(p,t) \wedge m'(p) \leqslant 1$.

Nets with inhibitor arcs*: There are two different semantics of inhibitor arcs. We only consider the* a-posteriori *semantics here: A transition* $t \in T$ can occur *in a marking* m*, if additionally to the conditions of p/t-nets (elementary nets) it is satisfied* $\forall p \in P : m(p) + W(t,p) \leqslant I(p,t)$*. A multi-set of transitions* τ can to occur *in* m*, if additionally* $\forall p \in P : m(p) + \sum_{t \in \tau} \tau(t) W(t,p) \leqslant I(p,t)$.

The notion of execution depends on the chosen net semantics.

Definition 3 (execution) *We consider the sequential semantics, the step semantics and the partial order semantics of Petri nets:*

Sequential semantics*: A* sequential execution *of a Petri net is a finite sequence of transitions* $\sigma = t_1 \dots t_m$ *such that there are markings* $m_1, \dots, m_n$ *satisfying* $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \xrightarrow{t_m} m_m$.

Step semantics*: A* step execution *of a Petri net is a finite sequence of multi-sets of transitions* $\sigma = \tau_1 \dots \tau_m$ *such that there are markings* $m_1, \dots, m_n$ *satisfying* $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_m} m_m$.

Partial order semantics*: A* partial order execution *of a Petri net is a LPO* lpo *such that each finite sequence of multi-sets of transitions* $\sigma = \tau_1 \dots \tau_m$*, such that* lpo_σ *is a sequentialization of* lpo*, is a step execution. Equivalently, a* partial order execution *of a p/t-net can be defined as a LPO* $\text{lpo} = (V, <, l)$ *such that for each cut* C *of* lpo *and each place* p *there holds:* $m_0(p) + \sum_{v<C} (W(l(v),p) - W(p,l(v))) \geqslant \sum_{v \in C} W(p,l(v))$*. For elementary nets one requires additionally* $m_0(p) + \sum_{v<C \vee v \in C} (W(l(v),p) - W(p,l(v))) \leqslant 1$*. For inhibitor nets one requires additionally* $m_0(p) + \sum_{v<C} (W(l(v),p) - W(p,l(v))) + \sum_{v \in C} W(l(v),p) \leqslant I(p,t)$ *for each* $t \in l(C)$.

Each sequential execution is also a step execution and each sequential or step execution is also a partial order execution. The set of (sequential, step, partial order) executions is always prefix and sequentialization closed. The p/t-net from Figure 1 has the sequential executions b, a, ab, aba, aab and the step executions $b, a, ab, aba, aab, a(a+b)$. Note that $(2a)b$ is not a step execution of this net and that therefore the fourth LPO in Figure 5 is not a partial order execution of this net.

3 REGIONS

As mentioned in the introduction, one can identify two types of regions for the definition of feasible places. We will show that for both types the set of all regions of a given language L equals the set of non-negative integral solutions of a possibly infinite linear system of the form $\mathbf{A}_L \cdot \mathbf{x} \leqslant \mathbf{b}_L$. For finite or regular languages, this representation of regions is used to define computable finite representations of the infinite set of all regions in the next section.

3.1 Transition-regions

A *transition-region* $\mathbf{r}$ can be defined for each mentioned combination of net class and language type. It directly defines a p/t-net- or elementary net-place $p_\mathbf{r}$ by determining

the numbers $m_0(p_\mathbf{r})$ and $W(p_\mathbf{r},t), W(t,p_\mathbf{r})$ for $t \in T$. If $T = \{t_1,\ldots,t_n\}$, then $\mathbf{r}$ is given as a $(2n+1)$-tuple $\mathbf{r} = (r_0,\ldots,r_{2n})$ of non-negative integers. Its components define these numbers via $m_0(p_\mathbf{r}) = r_0$, $W(p_\mathbf{r},t_i) = r_i$ and $W(t_i,p_\mathbf{r}) = r_{n+i}$ for $i \in \{1,\ldots,n\}$. Denoting $t_1 = a$ and $t_2 = b$, then the left place in Figure 2 is defined by $\mathbf{r}_{left} = (1,1,1,2,0)$ and the right place by $\mathbf{r}_{right} = (1,1,1,1,0)$.

For inhibitor net places additionally the numbers $I(p,t)$ for $t \in T$ must be defined. For such nets $\mathbf{r}$ is a $(3n+1)$-tuple $\mathbf{r} = (r_0,\ldots,r_{3n})$ of non-negative integers, where $I(p_\mathbf{r},t_i) = r_{2n+i}$ for $i \in \{1,\ldots,n\}$.

Since a region $\mathbf{r}$ is intended to define a feasible place $p_\mathbf{r}$, it is required to satisfy a property $(f)_L$ ensuring that $p_\mathbf{r}$ is feasible w.r.t. L. We define

Definition 4 (transition-region) *A tuple* $\mathbf{r}$ *as above is called a* transition-region *if it satisfies* $(f)_L$.

The property $(f)_L$ depends on the considered net class and on the type of L. In the following we state $(f)_L$ for several combinations of net class and language type. The following statement is shown in literature for p/t-nets with and without inhibitor arcs and classical languages and step languages. For the other combinations of net class and language type, the theorem is firstly stated in this paper.

Theorem 1 *A tuple* $\mathbf{r}$ *satisfies* $(f)_L$ *if and only if* $p_\mathbf{r}$ *is feasible w.r.t. L.*

That means the regions of L exactly define feasible places. Remember that $p_\mathbf{r}$ is feasible w.r.t. L if the net resulting from adding $p_\mathbf{r}$ still generates L. The property $(f)_L$ is defined as follows:

3.1.1 P/T-nets

Classical languages: For each $w = w'a_m \in L$ it holds $r_0 - \sum_{i=1}^n |w|_{t_i} r_i + \sum_{i=1}^n |w'|_{t_i} r_{n+i} \geqslant 0$. This is the case if and only if $\mathbf{a}_w \cdot \mathbf{r} \leqslant 0$ for $\mathbf{a}_w = (a_{w,0},\ldots,a_{w,2n})$ defined by

$$\mathbf{a}_{w,j} \quad = \quad \begin{cases} -1 & \text{if } j = 0, \\ |w|_{t_j} & \text{if } j \in \{1,\ldots,n\}, \\ -|w'|_{t_{j-n}} & \text{if } j \in \{n+1,\ldots,2n\}. \end{cases}$$

Denoting $t_1 = a$ and $t_2 = b$, for the right place of Figure 2 defined by $\mathbf{r}_{right} = (1,1,1,1,0)$ there holds for $w = aba$: $\mathbf{a}_w = (-1,2,1,-1,-1)$ and $\mathbf{a}_w \cdot \mathbf{r}_{right} = 1 > 0$. Thus $\mathbf{r}_{right}$ is not a region of $L = \{aba\}$ and this place is non-feasible w.r.t. L. On the other side, for the left place of Figure 2 defined by $\mathbf{r}_{left} = (1,1,1,2,0)$ there holds for $w = aba$: $\mathbf{a}_w \cdot \mathbf{r}_{left} = 0$. Thus $\mathbf{r}_{left}$ is a region of L and this place is feasible w.r.t. L.

Step language: For each $w = w'\alpha_m \in L$ it holds $r_0 - \sum_{i=1}^n |w|_{t_i} r_i + \sum_{i=1}^n |w'|_{t_i} r_{n+i} \geqslant 0$. This is the case if and only if $\mathbf{a}_w \cdot \mathbf{r} \leqslant 0$ for $\mathbf{a}_w = (a_{w,0},\ldots,a_{w,2n})$ by

$$\mathbf{a}_{w,j} \quad = \quad \begin{cases} -1 & \text{if } j = 0, \\ |w|_{t_j} & \text{if } j \in \{1,\ldots,n\}, \\ -|w'|_{t_{j-n}} & \text{if } j \in \{n+1,\ldots,2n\}. \end{cases}$$

Consider the step language $L = \{(2a), a(a+b)\}$. Then both place from Figure 2 are non-feasible w.r.t. L. This can be verified by similar computations as above: $\mathbf{a}_{(2a)} = (-1,2,0,0,0)$, $\mathbf{a}_{a(a+b)} = (-1,2,1,-1,-1)$ and $\mathbf{a}_{(2a)} \cdot \mathbf{r}_{left} = \mathbf{a}_{a(a+b)} \cdot \mathbf{r}_{right} = 1 > 0$.

Partial language: For each lpo $\in L$ it holds for each cut C of lpo: $r_0 + \sum_{i=1}^n l(V')(t_i)(r_{n+i} - r_i) - \sum_{i=1}^n l(C)(t_i) r_i \geqslant 0$, where $V' = \{v \in V \mid v < C\}$. This is the case if and only if $\mathbf{a}_{\text{lpo},C} \cdot \mathbf{r} \leqslant 0$ for $\mathbf{a}_{\text{lpo},C} = (a_{C,0},\ldots,a_{C,2n})$ defined by:

$$\mathbf{a}_{C,j} \quad = \quad \begin{cases} -1 & \text{if } j = 0, \\ l(V' \cup C)(t_j) & \text{if } j \in \{1,\ldots,n\}, \\ -l(V')(t_{j-n}) & \text{if } j \in \{n+1,\ldots,2n\}. \end{cases}$$

Let lpo be the fourth LPO from Figure 5 and $L = \{\text{lpo}\}$. The both places from Figure 2 are non-feasible w.r.t. L. This can be verified as follows: Let C_{left} be the cut of lpo consisting of the two a-labeled events and C_{right} be the cut of lpo consisting of one a- and one b-labeled node (there is only one such possibility). Then: $\mathbf{a}_{C_{left},\text{lpo}} = (-1,2,0,0,0)$, $\mathbf{a}_{C_{right},\text{lpo}} = (-1,2,1,-1,-1)$ and $\mathbf{a}_{C_{left},\text{lpo}} \cdot \mathbf{r}_{left} = 1 > 0$, $\mathbf{a}_{C_{right},\text{lpo}} \cdot \mathbf{r}_{right} = 1 > 0$.

In this case, Theorem 1 can be shown by a straightforward computation using the definition of partial order executions and the definition of places $p_\mathbf{r}$.

Altogether, $\mathbf{r}$ is a transition-region of some language L w.r.t. a p/t-net if and only if $\mathbf{r}$ is a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{r} \leqslant 0$ for an appropriately defined matrix $\mathbf{A}_L$ (which possibly has infinite many rows).

3.1.2 Elementary Nets

It is additional required that $\forall i \in \{0,\ldots,2n\} : r_i \leqslant 1$. This can be expressed by $\mathbf{e}_i \cdot \mathbf{r} \leqslant 1$ for $i \in \{0,\ldots,2n\}$, where the i-th component of e_i equals 1 and all other components equal 0.

Moreover, each marking reachable by transition occurrences from $m_0(p)$ is required to be less or equal to 1 (this directly follows from the definition of the occurrence rule of elementary nets). This second requirement formally reads as follows for the different language types:

Classical languages: For each $w \in L$ it holds $r_0 - \sum_{i=1}^n |w|_{t_i} r_i + \sum_{i=1}^n |w|_{t_i} r_{n+i} \leqslant 1$. This is the case if and only if $\mathbf{e}_w \cdot \mathbf{r} \leqslant 1$ for the vector $\mathbf{e}_w = (e_{w,0},\ldots,e_{w,2n})$ defined by

$$\mathbf{e}_{w,j} \quad = \quad \begin{cases} 1 & \text{if } j = 0, \\ -|w|_{t_j} & \text{if } j \in \{1,\ldots,n\}, \\ |w|_{t_{j-n}} & \text{if } j \geqslant n+1. \end{cases}$$

Step languages: For each $\alpha_1 \dots \alpha_m \in L$ it holds $r_0 - \sum_{i=1}^{n} |\alpha_1 \dots \alpha_m|_{t_i} r_i + \sum_{i=1}^{n} |\alpha_1 \dots \alpha_m|_{t_i} r_{n+i} \leqslant 1$. This is the case if and only if $\mathbf{e}_w \cdot \mathbf{r} \leqslant 1$ for the vector $\mathbf{e}_w = (e_{w,0}, \dots, e_{w,2n})$ defined by

$$\mathbf{e}_{w,j} = \begin{cases} 1 & \text{if } j = 0, \\ -|w|_{t_j} & \text{if } j \in \{1, \dots, n\}, \\ |w|_{t_{j-n}} & \text{if } j \geqslant n+1. \end{cases}$$

Partial languages: For each lpo $\in L$ it holds for each cut C of lpo: $r_0 + \sum_{i=1}^{n} l(V' \cup C)(t_i)(r_{n+i} - r_i) \leqslant 1$, where $V' = \{v \in V \mid v < C\}$. This is the case if and only if $\mathbf{e}_{\text{lpo},C} \cdot \mathbf{r} \leqslant 1$ for the vector $\mathbf{e}_{\text{lpo},C} = (e_{C,0}, \dots, e_{C,2n})$ defined by

$$\mathbf{e}_{C,j} = \begin{cases} 1 & \text{if } j = 0, \\ -l(V' \cup C)(t_j) & \text{if } j \in \{1, \dots, n\}, \\ l(V' \cup C)(t_{j-n}) & \text{if } j \geqslant n+1. \end{cases}$$

Altogether, $\mathbf{r}$ is a transition-region of L w.r.t. an elementary net if and only if $\mathbf{r}$ is a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{b}_L$.

3.1.3 Inhibitor Nets

A transition-region w.r.t. an inhibitor net is a $(3n+1)$-tuple. We set $a_{w,j} = a_{C,j} = e_j = e_{w,j} = e_{C,j} = 0$ for $j \in \{2n+1, \dots, 3n\}$. Then we additionally require for the different language types:

Classical languages: For each $w = w' a_m \in L$ and $t_k = a_m$ it holds $r_0 - \sum_{i=1}^{n} |w'|_{t_i} r_i + \sum_{i=1}^{n} |w|_{t_i} r_{n+i} - r_{2n+k} \leqslant 0$ in the *a-posteriori* semantics. This is the case if and only if $\mathbf{c}_w \cdot \mathbf{r} \leqslant 0$ for the vector $\mathbf{c}_w = (c_{w,0}, \dots, c_{w,3n})$ defined by

$$\mathbf{c}_{w,j} = \begin{cases} 1 & \text{if } j = 0, \\ -|w'|_{t_j} & \text{if } j \in \{1, \dots, n\}, \\ |w|_{t_{j-n}} & \text{if } n+1 \leqslant j \leqslant 2n, \\ -1 & \text{if } j = 2n+k, \\ 0 & \text{else} \end{cases}$$

Step languages: For each $w = w' \alpha_m \in L$ and each $t_k \in \alpha_m$ it holds $r_0 - \sum_{i=1}^{n} |w'|_{t_i} r_i + \sum_{i=1}^{n} |w|_{t_i} r_{n+i} - r_{2n+j} \leqslant 0$ in the *a-posteriori* semantics. This is the case if and only if $\mathbf{c}_w \cdot \mathbf{r} \leqslant 0$ for the vector $\mathbf{c}_w = (c_{w,0}, \dots, c_{w,3n})$ defined by

$$\mathbf{c}_{w,j} = \begin{cases} 1 & \text{if } j = 0, \\ -|w'|_{t_j} & \text{if } j \in \{1, \dots, n\}, \\ |w|_{t_{j-n}} & \text{if } n+1 \leqslant j \leqslant 2n, \\ -1 & \text{if } j = 2n+k, \\ 0 & \text{else} \end{cases}$$

Partial languages: For each lpo $\in L$, for each cut C of lpo and for each $t_k \in l(C)$ it holds $r_0 + \sum_{i=1}^{n} l(V')(t_i)(r_{n+i} - r_i) + \sum_{i=1}^{n} l(C)(t_i) r_{n+i} - r_{2n+j} \leqslant 0$ in the *a-posteriori* semantics (where $V' = \{v \in V \mid v < C\}$). This is the case if and only if $\mathbf{c}_{\text{lpo},C} \cdot \mathbf{r} \leqslant 0$ for the vector $\mathbf{c}_{\text{lpo},C} = (c_{C,0}, \dots, c_{C,3n})$ defined by

$$\mathbf{c}_{C,j} = \begin{cases} 1 & \text{if } j = 0, \\ -l(V')(t_j) & \text{if } j \in \{1, \dots, n\}, \\ l(V' \cup C)(t_{j-n}) & \text{if } n+1 \leqslant j \leqslant 2n, \\ -1 & \text{if } j = 2n+k, \\ 0 & \text{else} \end{cases}$$

Altogether, $\mathbf{r}$ is a transition-region of L w.r.t. an inhibitor net if and only if $\mathbf{r}$ is a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{0}$ in the p/t-net case and of $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{b}_L$ in the elementary net case.

3.2 Token Flow-regions

Also a *token flow-region* $\mathbf{r}$ can be defined for each combination of net class and language type. It defines a place $p_{\mathbf{r}}$ indirectly by determining the token flow (in this place) between transition occurrences, i.e. by determining the number of tokens produced by a transition which are consumed by a subsequent transition. For this, it is appropriate to consider the different language types as special classes of partial languages (as introduced in the last paragraph of Section 2). Then an execution is given as a LPO and such token flows can be assigned to its edges. In the definition also token flows of tokens consumed from the initial marking and of tokens which are produced but not further consumed are considered. If $W = \bigcup_{(V,<,l) \in L} V$ is the set of nodes of LPOs in L and $E = \bigcup_{(V,<,l) \in L} <$ is the set of edges of LPOs in L, then $\mathbf{r}$ is given as a tuple $\mathbf{r} = (r_i)_{i \in W \times \{in,out\} \cup E}$ of non-negative integers (with possibly infinite many components). Its components define

- the number of tokens, an event $v \in W$ consumes from the initial marking by $r_{v,in}$.
- the number of tokens, which are produced by an event v and not consumed by a subsequent event by $r_{v,out}$.
- the number of tokens, which are produced by an event v and consumed by an event w for $e = (v,w) \in E$ by r_e.

Consider the partial language L given in Figure 6. Figure 7 illustrates the token flow-region $\mathbf{r}$ defined by $r_{v_1,in} = 1$, $r_{v_2,in} = 1$, $r_{v_3,in} = 0$, $r_{v_4,in} = 0$ and $r_{v_1,out} = 0$, $r_{v_2,out} = 0$, $r_{v_3,out} = 0$, $r_{v_4,out} = 2$ and $r_{v_2,v_4} = 1$, $r_{v_2,v_3} = 1$.

Such a region $\mathbf{r}$ defines a p/t-net-place $p_{\mathbf{r}}$ as follows: $m_0(p_{\mathbf{r}}) = \sum_{v \in V} r_{v,in}$ for some LPO $(V,<,l) \in L$ – the sum is called *initial token flow* of the LPO; $W(p_{\mathbf{r}},t) = r_{v,in} + \sum_{e=(u,v) \in <} r_e$ for some LPO $(V,<,l) \in L$ and $v \in V$ with $l(v) = t$ – the sum is called *intoken flow* of v; $W(t,p_{\mathbf{r}}) = r_{v,out} + \sum_{e=(v,u) \in <} r_e$ for some LPO $(V,<,l) \in L$ and $v \in V$

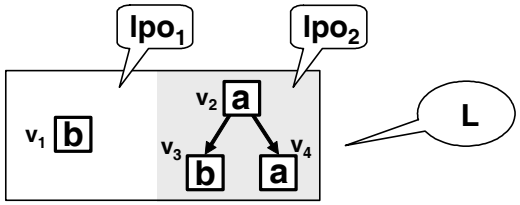

Figure 6: A partial language $L = \{\text{lpo}_1, \text{lpo}_2\}$ with $W = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_2, v_4), (v_2, v_3)\}$.

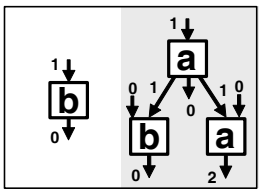

Figure 7: A token flow-region defining the left place from Figure 2. The intoken flow of both a-labeled nodes equals 1, the intoken flow of both b-labeled nodes equals 1, the outtoken flow of both a-labeled nodes equals 2, the outtoken flow of both b-labeled nodes equals 0 and the initial token flow of both LPOs equals 1.

with $l(v) = t$ – the sum is called *outtoken flow* of v (compare Figure 7). This construction is still dependent on the choice of $(V, <, l) \in L$ and $v \in V$, thus $p_{\mathbf{r}}$ is not well-defined. Therefore, we require $\mathbf{r}$ to fulfill a property $(wd)_L$ which makes $p_{\mathbf{r}}$ well defined. The property $(wd)_L$ states the following:

The initial token flows of different LPOs are equal: $\sum_{v \in V} r_{v,in} = \sum_{v' \in V'} r_{v',in}$ for $(V, <, l), (V', <', l') \in L$. This property can be expressed as linear equation system $\mathbf{A}_{L,a} \cdot \mathbf{r} = \mathbf{0}$ as follows: Let $L = \{\text{lpo}_1, \text{lpo}_2, \ldots\}$ and $\text{lpo}_n = (V_n, <_n, l_n)$. Then for each n the matrix $\mathbf{A}_{L,a}$ has a row $\mathbf{a}_n = (a_{n,i})_{i \in W \times \{in,out\} \cup E}$ defined by

$$\mathbf{a}_{n,i} = \begin{cases} 1 & \text{if } i = (v, in) \text{ for } v \in V_n, \\ -1 & \text{if } i = (v', in) \text{ for } v' \in V_{n+1} \\ 0 & \text{else.} \end{cases}$$

The intoken flows of equally labeled events are equal: $r_{v,in} + \sum_{e=(u,v) \in <} r_e = r_{v',in} + \sum_{e=(u,v') \in <'} r_e$ for $v \in V$, $v' \in V'$, $(V, <, l), (V', <', l') \in L$ and $l(v) = l'(v')$. This property can be expressed as linear equation system $\mathbf{A}_{L,b} \cdot \mathbf{r} = \mathbf{0}$ as follows: Let $W_t = \{v \in W \mid l(v) = t\} = \{v_1^t, v_2^t, \ldots\}$ be the set of all t-labeled events for $t \in T$. Then for each t and each n the matrix $\mathbf{A}_{L,b}$ has a row $\mathbf{b}_n^t = (b_{n,i}^t)_{i \in W \times \{in,out\} \cup E}$ defined by

$$\mathbf{b}_{n,i}^t = \begin{cases} 1 & \text{if } i = (v_n^t, in) \vee i = (u, v_n^t), \\ -1 & \text{if } i = (v_{n+1}^t, in) \vee i = (u, v_{n+1}^t) \\ 0 & \text{else.} \end{cases}$$

The outtoken flows of equally labeled events are equal: $r_{v,out} + \sum_{e=(v,u) \in <} r_e = r_{v',out} + \sum_{e=(v',u) \in <'} r_e$ for $v \in V$, $v' \in V'$, $(V, <, l), (V', <', l') \in L$ and $l(v) = l'(v')$. This property can be expressed as linear equation system $\mathbf{A}_{L,c} \cdot \mathbf{r} = \mathbf{0}$ as follows: For each t and each n the matrix $\mathbf{A}_{L,c}$ has a row $\mathbf{c}_n^t = (c_{n,i}^t)_{i \in W \times \{in,out\} \cup E}$ defined by

$$\mathbf{c}_{n,i}^t = \begin{cases} 1 & \text{if } i = (v_n^t, out) \vee i = (v_n^t, u), \\ -1 & \text{if } i = (v_{n+1}^t, out) \vee i = (v_{n+1}^t, u) \\ 0 & \text{else.} \end{cases}$$

It can be shown that the place $p_{\mathbf{r}}$ then is feasible w.r.t. L by construction, that means there is no need for a property $(f)_L$ as for transition-regions. As argued, there is a matrix $\mathbf{A}_L$ such that $\mathbf{r}$ is a token flow-region of L w.r.t. a p/t-net if and only if $\mathbf{r}$ is a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{r} = \mathbf{0}$ ($\mathbf{A}_L$ possibly has infinite many rows and columns)

To define elementary net-places, $\mathbf{r}$ has to satisfy an additional requirement $(elem)_L$. Property $(elem)_L$ reflects as in case of transition-regions that all numbers defining a place are less or equal to 1, and that each marking reachable by transition occurrences from $m_0(p)$ is less or equal to 1. The first condition can be encoded by requiring that all initial token flows, intoken flows and outtoken flows are less or equal to 1 (which can be written as linear inhomogenous inequations). The second condition can be encoded by requiring that for each cut C the sum of token flows on edges from events $v < C$ to events $v \not< C$, from the initial marking to events $v \not< C$ and from events $v < C$ to the final marking is less or equal to 1 (which also can be written as a linear inhomogenous inequation). We omit to give the additional rows of the linear system.

To define inhibitor places from token flow-regions, we add new components to regions defining inhibitor net places. As in the case of transition regions, we add such a component for each transition t defining by this component the value $I(p,t)$ for the place p defined by the region. Then an additional property $(inh)_L$ is required to define token flow-regions for inhibitor nets. This property states that for each cut C and each $w \in C$ the sum of token flows on edges from events $v < C$ to events $v \not< C$, from the initial marking to events $v \not< C$, from events $v < C$ to the final marking and from events $v \in C$ to other events or the final marking is less or equal to the component defining $I(p, l(w))$. This condition can be written as a linear homogenous inequation. We omit to give the additional rows of the linear system.

Altogether, we have the following definition of token flow-regions:

Definition 5 (token flow-region) *A tuple* $\mathbf{r}$ *as above is called a* token flow-region *w.r.t. p/t-nets if it satisfies* $(wd)_L$. *It is called a token flow region w.r.t. elementary nets, if it additionally satisfies* $(elem)_L$. *It is called a token flow region w.r.t. inhibitor nets, if it additionally satisfies* $(inh)_L$.

The following statement is shown in literature for p/t-nets with and without inhibitor arcs and partial languages (including classical languages and step languages). For elementary nets, the theorem is firstly stated in this paper.

Theorem 2 *A tuple* $\mathbf{r}$ *as above is a token flow-region if and only if* $p_{\mathbf{r}}$ *is feasible w.r.t.* L.

4 COMPUTING FINITE REPRESENTATIONS

As shown in the last paragraph, each of the properties $(f)_L$, $(wd)_L$, $(elem)_L$ and $(inh)_L$ can in each case be encoded by a system of linear equations or inequations of the form $\mathbf{A}_L \cdot \mathbf{r} = \mathbf{0}$, $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{0}$ or $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{b}_L$ for a matrix $\mathbf{A}_L$ and a vector $\mathbf{b}_L$. Each row of $\mathbf{A}_L$ reflects one condition of the considered property, that means $\mathbf{A}_L$ has in general infinite many rows and in the case of token flow-regions, $\mathbf{A}_L$ also has infinite many columns.

But there are several cases, when $\mathbf{A}_L$ can be chosen with finite many rows and columns. If L is finite then this is obviously the case. If L is infinite, then there are finite representations of L allowing to chose $\mathbf{A}_L$ to be finite. This is the case for example if $L = L(\alpha)$ is a regular language given by a regular expression α (combining characters by $+$-operation (choice), ;-operation (sequence) and $*$-operation (iteration)). Then there can be constructed a finite subset $L' \subseteq L(\alpha)$ to define the mentioned equations and inequations. Additional equations ensure that parts of the executions (defined by the $*$-operation in α) can be iterated. This construction can be generalized to regular step languages and regular partial languages, where characters in α are interpreted as multi-sets of events resp. as LPOs. There is also an extension by the $\|$-operation for concurrent composition. These constructions are too involved to be presented here (here we refer the reader to existing publications, Section 5).

Altogether, in these cases it is possible to define a finite matrix $\mathbf{A}_L$ such that $\mathbf{r}$ is a region if and only if $\mathbf{r}$ is a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{r} = \mathbf{0}$, $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{0}$ or $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{b}_L$. In other words, the set of regions can be computed as the set of non-negative integer solutions of such a system. There are two possibilities to define a finite representation of such a set of solutions.

4.1 Basis Representation

For systems of the form $\mathbf{A}_L \cdot \mathbf{r} = 0$ or $\mathbf{A}_L \cdot \mathbf{r} \leqslant 0$ there is a so called *basis representation* of the set of all non-negative solutions. That means there are non-negative basis-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 $\lambda_1, \ldots, \lambda_n \geqslant 0$. In the case that all values in $\mathbf{A}_L$ are integral (this is the case here) also the values of $\mathbf{y}_1, \ldots, \mathbf{y}_n$ can be chosen integral. If p_i is the place defined by $\mathbf{y}_i$ and N_{fin} is the net containing exactly the places $p_1, \ldots, p_n$, then it is shown that $L(N_{fin}) = L(N_{sat})$. That means N_{fin} is the best approximation to a solution of the synthesis problem generating L. Since from the construction it is not clear whether N_{fin} is an exact solution of the synthesis problem, it is finally necessary to test whether $L(N_{fin}) = L$ or not. Such tests are briefly discussed at the end of this section. N_{fin} is called *basis representation* of N_{sat}.

The property $L(N_{fin}) = L(N_{sat})$ is shown for p/t-nets, token flow-regions and finite partial languages. Since classical languages and step languages are special classes of partial languages, this also holds for p/t-nets, token flow-regions and such languages (which are finite). Moreover, this statement carries over to transition-regions and it is possible to show it also for p/t-inhibitor net places.

For systems of the form $\mathbf{A}_L \cdot \mathbf{r} = 0$, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ can be chosen as so called *minimal solutions*. That means, the numbers $m_0(p_i), W(p_i,t), W(t,p_i)$ determined by $\mathbf{y}_i$ are minimal. This is the case for token flow-regions and p/t-nets with or without inhibitor nets. In the worst case n can be exponential in the number of rows of $\mathbf{A}_L$, but in practise it is often small. There are effective algorithms to compute $\mathbf{y}_1, \ldots, \mathbf{y}_n$. Unfortunately, it is not possible to define a basis representation for elementary net-regions. This is because the linear inequation system defining regions is then not longer homogenous.

4.2 Separating Representation

Another idea is to separate behavior specified in L from behavior not specified in L by a finite set of regions. For this, one defines a finite set L^c with $L \cap L^c = \emptyset$ satisfying that $L(N) \cap L^c = \emptyset \Longrightarrow L(N) = L$ for each net N. Then for each $w \in L^c$ one tries to find a region $\mathbf{r}_w$ such that w is not an execution of the net having the place $p_{\mathbf{r}_w}$, i.e. a region which *separates L from w*. If such a region exists, then the corresponding place is added to the net N_{fin} called *separating representation* of N_{sat}. There is an exact solution of the synthesis problem if and only if for each $w \in L^c$ there is such a region $\mathbf{r}_w$. In case L is a net language, it holds $L(N_{fin}) = L(N_{sat}) = L$ – that means N_{fin} is a possible solution. In case L is not a net language, it holds $L(N_{sat}) \subseteq L(N_{fin})$ but in general not $L(N_{sat}) = L(N_{fin})$. Thus, the separating representation in general is not the best approximation to a solution of the synthesis problem generating L (in opposite to the basis representation). On the other hand, it can be defined and computet not only for p/t-nets and inhibitor nets, but also for elementary nets.

It is easy to define and compute a separating representation for finite languages. The definition of the finite set L^c depends on the considered language type. The elements of L^c are called *wrong continuations*. In each case, a wrong continuation is an execution in L extended by some subsequent event.

Classical languages: The finite set $L^c = \{wt \mid w \in L, t \in T, wt \notin L\}$ satisfies $L^c \cap L = \emptyset$ and $L(N) \cap L^c = \emptyset \Longrightarrow L(N) = L$ for each net N. For $L = \{b, a, ab, aba, aab\}$ there holds $L^c = \{ba, bb, abb, abab, abaa, aabb, aaba\}$.

Step languages: The finite set $L^c = \{w(\alpha + t) \mid w\alpha \in L, t \in T, w(\alpha + t) \notin L\}$ satisfies $L^c \cap L = \emptyset$ and $L(N) \cap L^c = \emptyset \Longrightarrow L(N) = L$ for each net N. For $L = \{b, a, ab, aba, aab, a(a + b)\}$ there holds $L^c = \{ba, bb, abb, abab, abaa, aabb, aaba, (2b), (b + a), (2a), a(2b), ab(2a), ab(a + b), aa(2b), aa(a + b), a(a + 2b), a(2a + b)\}$.

Partial languages: The finite set L^c of LPOs $\text{lpo}_{C,t} \notin L$, which are constructed from some prefix of a sequentialization lpo of a LPO in L by adding a t-labeled event subsequent to some co-set C (C is allowed to be empty), satisfies $L^c \cap L = \emptyset$ and $L(N) \cap L^c = \emptyset \Longrightarrow L(N) = L$ for each net N. For L given in Figure 6 some LPOs in L^c are shown in Figure 8.

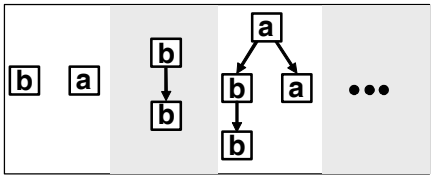

Figure 8: Some LPOs in L^c for L given in Figure 6. The most left LPO is constructed from the empty co-set C.

A region separating L from some $w \in L^c$ is computed as an arbitrary solution of an adequate linear inequation system. Such a system consists of the equations/inequations given by $\mathbf{A}_L$ defining regions and an additional row $\mathbf{d}_w$. The row is defined in such a way that $\mathbf{d}_w \cdot \mathbf{r} < 0$ if and only if w is not an execution of the net having the place $p_\mathbf{r}$. There are polynomial algorithms to compute a non-negative integer solution of a system $\mathbf{A}_L \cdot \mathbf{r} \leqslant \mathbf{b}_L$, $\mathbf{d}_w \cdot \mathbf{r} < 0$.

For transition-regions and p/t-nets, the row $\mathbf{d}_w$ is defined in a similar way as the rows of $\mathbf{A}_L$. For each $w \in L$ there is a row $\mathbf{a}_w$ of $\mathbf{A}_L$ such that w is an execution of a net N if and only if all places $p_\mathbf{r}$ of N satisfy $\mathbf{a}_w \cdot \mathbf{r} \leqslant \mathbf{0}$. That means, w is *not* an execution of a net N if and only if there is a place $p_\mathbf{r}$ of N satisfying $\mathbf{a}_w \cdot \mathbf{r} > \mathbf{0}$. Thus $\mathbf{d}_w$ can be defined as $\mathbf{d}_w = -\mathbf{a}_w$ for $w \in L^c$. In the case of elementary nets and inhibitor nets additional possibilities to separate L from some $w \in L^c$ must be considered. These can be deduced from the additional conditions regions satisfy for such nets.

For token flow-regions, $\mathbf{d}_w$ counts intoken flows and outtoken flows of events of prefixes of LPOs. If w is a wrong continuation, extending the execution $v \in L$ by an event e, then $\mathbf{d}_w$ is defined in such a way that $\mathbf{d}_w \cdot \mathbf{r}$ counts the number of tokens remaining in $p_\mathbf{r}$ after the execution of v minus the number of tokens consumed by e (as it is the case for transition-regions).

There is an approach to define L^c and compute a separating representation of transition-regions for regular classical languages which is too involved to be presented here. This approach was not generalized to regular step languages or regular partial languages so far, neither for transition-regions nor for token flow-regions.

Finally, we want to mention that it is possible to develop a test for $L(N_{fin}) = L$ from L^c, if N_{fin} is a basis representation: It is simply enough to test whether no LPO in L^c is an execution of N_{fin}. This can be done for each LPO by the polynomial algorithm given in Juhás, Lorenz, and Desel (2005).

5 DISCUSSION

In Darondeau (1998), Badouel and Darondeau (1996a), and Badouel and Darondeau (1996b), transition regions are defined for a parametric definition of p/t-nets, so called type of nets, and regular classical languages. With types of nets, all net classes considered in this paper can be instantiated. There is derived a separating representation for types of nets. This representation coincides with the presented one for finite classical languages (we did not present in this paper the solution for regular languages), although we used a different terminology here. In the case of p/t-nets, the separating representation can be computed in polynomial time, if the regular language is given by a regular expression in a special form. Computing this special form (from an arbitrary one) needs exponential size in the worst case.

In Lorenz et al. (2007), Lorenz and Juhás (2006), and Lorenz, Bergenthum, and Mauser (2007), token flow regions are introduced for p/t-nets with and without inhibitor arcs and partial languages. For finite partial languages, a basis representation is deduced. In Lorenz et al. (2007) a test of $L(N_{fin}) = L$ for a basis representation N_{fin} is developed through defining a set L^c as in the last section. As argued at the end of the last section, it is possible to define from L^c a separating representation. The presentation of token flow-regions in this paper follows essentially the lines presented in Lorenz et al. (2007) and Lorenz, Bergenthum, and Mauser (2007).

In Hoogers, Kleijn, and Thiagarajan (1995) transition-regions for trace languages (step languages) and p/t-nets are introduced. The authors do not consider the subclasses of finite or regular step languages and therefore only define an infinite separating representation (involving infinite many constraints corresponding to the constraints given in Section 3)

There are many other publications considering (step) transition systems instead of languages as behavioral model (e.g., Ehrenfeucht and Rozenberg 1989a, Ehrenfeucht and Rozenberg 1989b, Mukund 1992, Pietkiewicz-Koutny 1999, Busi and Pinna 1997, Pietkiewicz-Koutny 2002, Darondeau 2003). These approaches are restricted to transition-regions and separating representations and yield polynomial synthesis algorithms in case of p/t-nets. They do not guarantee in

general that the computed net has the smallest net behavior including the specified behavior.

There are some approaches starting from languages, we did not mention yet. One is the generalization of of regular to deterministic context-free classical languages in Darondeau (1998). Another is the generalization of partial languages to stratified languages, which describe the behavior of inhibitor nets w.r.t. the so called *a-priori semantics*, see Lorenz, Bergenthum, and Mauser (2007).

In this paper we developed new synthesis algorithms from transition regions applied to partial languages, computing a basis representation or a separating representation. The other way round, we combined for the first time token flow-regions and basis representations with classical languages. Moreover, we introduced the first synthesis algorithms for step languages. Up to now, there are no synthesis algorithms starting from regular step languages or regular partial languages.

Finally, there is some speciality concerning the synthesis from descriptions of sequential behavior, we did not mention yet. In the beginning of research in synthesis of Petri nets, one proposed methods to synthesize elementary nets without loops from descriptions of sequential behavior. Because of the absence of loops, the synthesized net has maximal concurrency among transitions in this case (from the two sequences ab and ba there would be synthesized a net with two concurrent transitions a and b, which do not share a common resource in form of a loop place). If loops a allowed, one gets a net with minimal concurrency among transitions, because such loop places are always feasible w.r.t. sequential behavior. Of course this is not the case if one considers non-sequential behavior. It should be clear to the reader how to modify the inequation system for computing feasible places in order to avoid loops for both types of regions.

Concerning performance, it seems that solving the synthesis problem for languages needs in general exponential time. One exception are finite languages, where separating representations can be computed in polynomial time for classical and for step languages. For partial languages, all presented algorithm need exponential time. However, although computing a basis representation is exponential in worst case, since there can be exponentially many basis solutions, in practise the set of basis solutions is often small.

The presented framework offers several different synthesis algorithms which can be applied in a concrete setting. Each of these algorithms yields a different resulting net. In depends on the application area, which method is finally most appropriate. In general, the computation of basis representations and token flow-regions seems to be more time consuming than of separating representations and transitions-regions. On the other hand, only the basis-representation guarantees that the computed net has the minimal net behavior including the given language. Token flow-regions lead to minimal solutions and therefore "nice" places.

6 CONCLUSION

In this paper we presented a survey on region-based synthesis methods from languages. These approaches are based on the definition of the infinite saturated feasible net which has minimal net behavior including the given language. We identified two types of regions and two types of finite representations of the saturated feasible net and showed, that these can be arbitrarily combined in order to solve the synthesis problem for many Petri net classes and language types. The presented framework integrates almost all known region-based methods and completes these existing approaches by several new algorithms. It is common to all methods that they involve integer programming methods.

An actual application area of synthesis methods is the area of process mining (van der Aalst and Günther 2007, Bergenthum et al. 2007). There one tries to compute system models from event logs (generated by some information system). These event logs can be thought of as observations of some real system. A topic of future research could be to synthesize in a similar way system models from trajectories of simulation.

ACKNOWLEDGMENTS

This work was supported by the German Academic Exchange Service (DAAD) within the project SAMANET and by the German Research Council (DFG) within the project SYNOPS.

REFERENCES

Badouel, E., and P. Darondeau. 1996a. On the synthesis of general petri nets. Technical Report 3025, Inria.

Badouel, E., and P. Darondeau. 1996b. Theory of regions. In *Petri Nets*, ed. W. Reisig; G. Rozenberg, Volume 1491 of *Lecture Notes in Computer Science*, 529–586: Springer.

Bergenthum, R., J. Desel, R. Lorenz, and S. Mauser. 2007. Process mining based on regions of languages. In *Proceedings of BPM 2007*.

Busi, N., and G. M. Pinna. 1997. Synthesis of nets with inhibitor arcs. In *CONCUR*, ed. A. W. Mazurkiewicz; J. Winkowski, Volume 1243 of *Lecture Notes in Computer Science*, 151–165: Springer.

Cortadella, J., M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. 2000. Hardware and petri nets: Application to asynchronous circuit design. In *ICATPN*, ed. M. Nielsen; D. Simpson, Volume 1825 of *Lecture Notes in Computer Science*, 1–15: Springer.

Darondeau, P. 1998. Deriving unbounded petri nets from formal languages. In *CONCUR*, ed. D. Sangiorgi; R. de Simone, Volume 1466 of *Lecture Notes in Computer Science*, 533–548: Springer.

Darondeau, P. 2003. Unbounded petri net synthesis. In *Lectures on Concurrency and Petri Nets*, ed. J. Desel, W. Reisig, and G. Rozenberg, Volume 3098 of *Lecture Notes in Computer Science*, 413–438: Springer.

Ehrenfeucht, A., and G. Rozenberg. 1989a. Partial (set) 2-structures. part I: Basic notions and the representation problem. *Acta Inf.* 27 (4): 315–342.

Ehrenfeucht, A., and G. Rozenberg. 1989b. Partial (set) 2-structures. part II: State spaces of concurrent systems. *Acta Inf.* 27 (4): 343–368.

Hoogers, P., H. Kleijn, and P. Thiagarajan. 1995. A trace semantics for petri nets. *Information and Computation* 117 (1): 98–114.

Josephs, M. B., and D. P. Furey. 2002. A programming approach to the design of asynchronous logic blocks. In *Concurrency and Hardware Design*, Volume 2549 of *Lecture Notes in Computer Science*, 34–60.

Juhás, G., R. Lorenz, and J. Desel. 2005. Can I execute my scenario in your net? In *ICATPN*, ed. G. Ciardo and P. Darondeau, Volume 3536 of *Lecture Notes in Computer Science*, 289–308: Springer.

Lorenz, R., R. Bergenthum, and S. Mauser. 2007. Theory of regions for the synthesis of inhibitor nets from scenarios. In *Proceedings of ICATPN 2007*.

Lorenz, R., R. Bergenthum, S. Mauser, and J. Desel. 2007. Synthesis of petri nets from finite partial languages. In *Proceedings of ACSD 2007*.

Lorenz, R., and G. Juhás. 2006. Towards synthesis of petri nets from scenarios. In *ICATPN*, ed. S. Donatelli and P. S. Thiagarajan, Volume 4024 of *Lecture Notes in Computer Science*, 302–321: Springer.

Mukund, M. 1992. Petri nets and step transition systems. *Int. J. Found. Comput. Sci.* 3 (4): 443–478.

Pietkiewicz-Koutny, M. 1999. The synthesis problem for elementary net systems with inhibitor arcs. *Fundam. Inform.* 40 (2-3): 251–283.

Pietkiewicz-Koutny, M. 2002. Synthesising elementary net systems with inhibitor arcs from step transition systems. *Fundam. Inform.* 50 (2): 175–203.

van der Aalst, W., and C. Günther. 2007. Finding structure in unstructered processes: The case of process mining. In *Proceedings of ACSD 2007, invited paper*.

van der Aalst, W. M. P., B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. 2003. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.* 47 (2): 237–267.

Zhou, M., and F. D. Cesare. 1993. *Petri net synthesis for discrete event control of manufacturing systems.* Kluwer.

AUTHOR BIOGRAPHIES

ROBERT LORENZ is an assistant professor in computer science at the Catholic University of Eichsttt-Ingolstadt in Germany. In 2006 he finished his postdoctoral lecture qualification with the habilitation thesis "Scenario based verification and synthesis of Petri net models." Actual he is leader of the DFG project SYNOPS (Synthesis of Petri nets from Scenarios) concerning the efficiency of Petri net synthesis methods and their applicability in several application areas (for example process mining). Since this year he is member of the program committee of the conference series "Petri Nets."

SEBASTIAN MAUSER finished his Diplom (German university degree) in business mathematics at the Catholic University Eichstt-Ingolstadt in 2006. Then he started his postgraduate studies as a research assistant at the department of Applied Computer Science in Eichsttt.

JUHÁS GABRIEL is full professor in computer science at the Slovak University of Technology in Bratislava, Solvakia. Since several years he is member of the program committee of the conference series "Petri Nets." In 2007 he organizes the 7th International Conference on Application of Concurrency to System Design (ACSD). Actual he leader of the slovakian side of the DAAD project SAMANET (Scenario based approaches for misbehavior detection in ad hoc wireless networks) concerning the application of Petri net verification and synthesis methods to misbehavior detection in ad hoc wireless networks.