Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language

Robin Bergenthum and Sebastian Mauser

Department of Applied Computer Science,
Catholic University of Eichstätt-Ingolstadt,
firstname.lastname@ku-eichstaett.de

Abstract. In this paper we present two new algorithms that effectively synthesize a finite place/transition Petri net (p/t-net) from a finite set of labeled partial orders (a finite partial language). Either the synthesized p/t-net has exactly the non-sequential behavior specified by the partial language, or there is no such p/t-net. The first algorithm is an improved version of a synthesis algorithm presented in [11], which uses the classical theory of regions applied to the set of step sequences generated by the given partial language. Instead of computing all step sequences, the new algorithm directly works on appropriate prefixes specified in the partial language. The second algorithm is based on the theory of token flow regions for partial languages developed in [13, 12, 11]. While in [12, 11] a so called basis representation is applied, the new algorithm for the first time combines the concept of separation representation with the idea of token flows. We implemented both synthesis algorithms in our framework VipTool. The implementations are used to compare the two new algorithms (also with the two algorithms presented in [11]). The paper provides experimental results.

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 system design [4, 9], and recently also in workflow design [16]. Moreover, there are several synthesis tools that are based on the theoretical results. The most prominent one is Petrify [3].

Originally, synthesis meant algorithmic construction of an unlabeled Petri net from sequential observations. It can be applied to various classes of Petri nets, including elementary nets [6] and place/transition nets (p/t-nets) [1]. Synthesis can start with a transition system representing the sequential behavior of a system or with a step transition system which additionally represents steps of concurrent events [1]. Synthesis can also be based on a language (originally a set of occurrence sequences or step sequences [5]). The *synthesis problem* is to decide whether, for a given behavioral specification, there exists an unlabeled Petri net of the respective class such that the behavior of this net coincides with the specified behavior.

In [11] we developed two algorithms to solve the synthesis problem for p/t-nets where the behavior is given in terms of a finite partial language L, i.e. as a finite set of labeled partial orders (LPOs – also known as *partial words* [7] or *pomsets* [15]).

Partial order behavior of Petri nets truly represents the concurrency of events and is often considered the most appropriate representation of behavior of Petri net models. Both algorithms apply the so called *theory of regions* in the setting of partial languages. A region defines a possible place of the synthesized net structurally w.r.t the given specification.

All approaches to Petri net synthesis based on regions roughly follow the same idea:

- Instead of solving the synthesis problem (is there a net with the specified behavior?) and then – in the positive case – synthesizing the net, first a net is constructed from the given specification.
- The construction starts with the transitions taken from the behavioral specification.
- Its behavior will be restricted by the addition of places and according initial markings and arcs.
- Each region yields a corresponding place in the constructed net. A region is defined in such a way that the behavior of the net with its corresponding place still includes the specified behavior.
- When all, or sufficiently many, regions are identified, all places of the synthesized net are constructed.
- If the behavior of the synthesized net coincides with the specified behavior, then the synthesis problem has a positive solution; otherwise there is no Petri net with the specified behavior and therefore the synthesis problem has a negative solution.

Each region yields a feasible place. The crucial point is that the set of all regions is mostly infinite, whereas in most cases a finite set of regions suffices to represent all relevant dependencies. In region theory there are basically two different approaches to calculate such a finite set of regions, called *basis representation* and *separation representation*. Both methods can be adapted to various region definitions as described in [14].

The two algorithms to synthesize a p/t-net from a finite partial language presented in [11] use the following principles: The first one (Alg. I) uses a region definition based on a translation of the given partial language into a language of step sequences, namely so called *transition regions* of the set of step sequences generated by the partial language are applied. This approach is combined with the principle of separation representation. The second algorithm (Alg. II) used a region definition defined directly on the partial language, so called *token flow regions*, combined with the basis representation approach. We implemented both algorithms and compared their runtime with different examples. The first algorithm Alg. I using the transition regions is faster than Alg. II in examples which do not contain a lot of concurrency. The separation representation used in this algorithm leads to solutions containing far less places than the second approach using the basis representation. In examples containing a lot of concurrency the translation of the partial language into a language of step sequences is too costly. In these examples Alg. II, defining regions directly on the given partial language, is superior to Alg. I, but using the basis representation approach seems to worsen the overall runtime. Computing the complete basis seems to be inefficient.

The aim of this paper is to deduce two new algorithms from the experiences of the results in [11]. The first one is an advanced version of Alg. I using so called *LPO*

transition regions. LPO transition regions are very similar to the transition regions of Alg. I, but it is not necessary to translate the given partial language into a language of step sequences. Instead appropriate prefixes of the partial language are considered. Therefore, this new algorithm performs better then Alg I and this leads to a very fast algorithm solving the synthesis problem given a partial language containing not too much concurrency. But if there is a lot of concurrency, the number of prefixes is very high making this algorithm problematic in runtime. The second new algorithm uses the token flow regions as Alg II. The difference is that we use an adapted separation representation, which is superior to the basis representation in practical use. This leads to an algorithm which performs a lot faster than Alg. II in any example. This new algorithm is a very fast algorithm solving the synthesis problem given a partial language containing much concurrency, since the performance of the algorithm is better if the partial language exhibits much concurrency.

We implemented both algorithms and performed runtime tests. As explained, each of the two new algorithms is superior to one of the algorithms presented in [11]. Thus they are the most promising algorithms to synthesize a p/t-net from a finite partial language developed so far. To examine their relation in detail, we present experimental results comparing the two new algorithms in this paper.

The remainder of the paper is organized as follows: we start with a brief introduction to p/t-nets, LPOs, partial languages and enabled LPOs in Section 2. In Section 3 we consider the problem of synthesizing a p/t-net from a given partial language. In Section 4 we first describe transition regions as developed in [11] and second improve those transition regions to a new advanced region definition of so called LPO transition regions. The definition of LPO transition regions leads to a new advanced algorithm solving the synthesis problem. In Section 5 we recall the definitions and main results from [13]. We develop a second algorithm to solve the synthesis problem using token flow regions as described in [12], but use a better adapted approach to calculate a finite set of places. Finally, we provide experimental results on the time consumption of both new synthesis algorithms.

2 Preliminaries

In this section we first introduce the definitions of *place/transition nets* (p/t-nets), *labeled partial orders* (LPOs), *partial languages* and LPOs *enabled w.r.t. p/t-nets*. We start with basic mathematical notations: by $\mathbb{N}$ we denote the *nonnegative integers*. $\mathbb{N}^+$ denotes the positive integers. The set of all *multi-sets* over a set A is the set $\mathbb{N}^A$ of all functions $f : A \to \mathbb{N}$. Addition $+$ on multi-sets is defined as usual by $(m + m')(a) = m(a) + m'(a)$. We also write $\sum_{a \in A} m(a)a$ to denote a multi-set m over A and $a \in m$ to denote $m(a) > 0$.

Definition 1 (Place/transition net). *A* place/transition-net *(*p/t-net*)* N *is a quadruple* (P, T, F, W)*, where* P *is a (possibly infinite) set of* places*,* T *is a finite set of* transitions *satisfying* $P \cap T = \emptyset$*,* $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation *and* $W : F \to \mathbb{N}^+$ *is a* weight function*.*

We extend the weight function W to pairs of net elements $(x, y) \in (P \times T) \cup (T \times P)$ with $(x, y) \notin F$ by $W(x, y) = 0$. A *marking* of a p/t-net $N = (P, T, F, W)$ is a multi-

set $m : P \to \mathbb{N}$ 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*. Figure 1 shows a marked p/t-net (N, m_0). Places are drawn as circles including tokens representing the initial marking, transitions as rectangles and the flow relation as arcs annotated by the values of the weight function (the weight 1 is not shown).

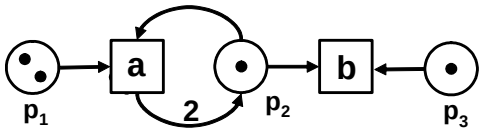

Fig. 1. A marked p/t-net (N, m_0).

A multi-set of transitions $\tau \in \mathbb{N}^T$ is called a *step* (of transitions). A step τ is *enabled to occur* (concurrently) in a marking m if and only if $m(p) \geq \sum_{t \in \tau} \tau(t) W(p,t)$ for each place $p \in P$. In this case, its occurrence leads to the marking $m'(p) = m(p) + \sum_{t \in \tau} \tau(t)(W(t,p) - W(p,t))$. We write $m \xrightarrow{\tau} m'$ to denote that τ is enabled to occur in m and that its occurrence leads to m'. A finite sequence of steps $\sigma = \tau_1 \dots \tau_n,\ n \in \mathbb{N}$, is called a *step 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 $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_n} m_n$. In the marked p/t-net (N, m_0) from Figure 1 only the steps a and b are enabled to occur in the initial marking. In the marking reached after the occurrence of a, the step $a + b$ is enabled to occur.

We use partial orders with nodes (called events) *labeled by transition names* to specify scenarios describing the behavior of Petri nets.

Definition 2 (Labeled Partial order). *A* labeled partial order *(LPO) is a triple $lpo = (V, <, l)$, where V is a finite* set of nodes*, $< \subseteq V \times V$ is an irreflexive and transitive relation over V, called the* set of edges *and $l : V \to T$ is a* labeling function *with* set of labels T.

Two nodes $v, v' \in V$ of an LPO $(V, <, l)$ 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 an LPO $(V, <, l)$ 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 $(V', <', l')$ is a *prefix* of another partial order $(V, <, l)$ if $V' \subseteq V$, $(v' \in V' \wedge v < v') \Longrightarrow (v \in V')$ and $<' = < \cap V' \times V'$.

A partial order $po = (V, <)$ is called *stepwise linear* if co is transitive. Given LPOs $po_1 = (V, <_1, l)$ and $po_2 = (V, <_2, l)$, po_2 is a *sequentialization* of po_1 if $<_1 \subseteq <_2$. If po_2 is stepwise linear, it is called *step linearization* of po_1.

In this paper we consider LPOs only up to isomorphism. Two LPOs $(V, <, l)$ and $(V', <', l')$ are called *isomorphic*, if there is a bijective mapping $\psi : V \to V'$ such that $l(v) = l'(\psi(v))$ for $v \in V$, and $v < w \iff \psi(v) <' \psi(w)$ for $v, w \in V$.

Definition 3 (Partial language). *Let T be a finite set. A set $L \subseteq \{\mathrm{lpo} \mid \mathrm{lpo}$ is an LPO with set of labels $T\}$ is called* partial language over T.

We always assume that each label from T occurs in a partial language over T. Figure 2 shows a partial language given by the set of LPOs $L = \{\mathrm{lpo}_1, \mathrm{lpo}_2\}$, which we will use as a running example.

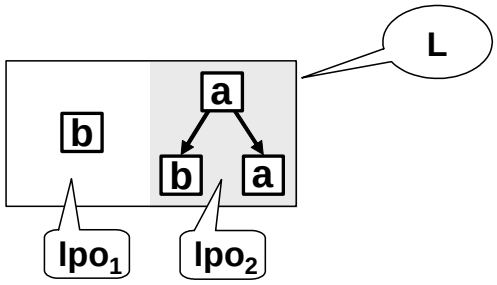

Fig. 2. A partial language.

There are two equivalent formal notions of runs of p/t-nets defining non-sequential semantics based on [10, 17]. We only give the notion of enabled LPOs here: an LPO is enabled w.r.t. a marked p/t-net, if for each cut of the LPO the marking reached by firing all transitions corresponding to events smaller than the cut enables the step (of transitions) given by the cut.

Definition 4 (Enabled LPO). *Let (N, m_0) be a marked p/t-net, $N = (P, T, F, W)$. An LPO $\mathrm{lpo} = (V, <, l)$ with $l : V \to T$ is called* enabled (to occur) in (N, m_0) *if $m_0(p) + \sum_{v \in V \wedge v < C}(W(l(v), p) - W(p, l(v))) \geq \sum_{v \in C} W(p, l(v))$ for every cut C of* lpo *and every $p \in P$. Its* occurrence *leads to the* final marking m' *given by $m'(p) = m_0(p) + \sum_{v \in V}(W(l(v), p) - W(p, l(v)))$.*

Enabled LPOs are also called runs*. The set of all LPOs enabled in (N, m_0) is $\mathfrak{Lpo}(N, m_0)$. $\mathfrak{Lpo}(N, m_0)$ is called the* partial language of runs *of (N, m_0).*

There is an equivalent characterization of enabledness using step sequences and their correspondence to stepwise linear LPOs: a stepwise linear LPO $\mathrm{lpo}' = (V, <', l)$ can be represented by the step sequence $\sigma_{lpo'} = \tau_1 \ldots \tau_n$ defined by $V = V_1 \cup \ldots \cup V_n$, $<' = \bigcup_{i<j} V_i \times V_j$ and $\tau_i(t) = |\{v \in V_i \mid l(v) = t\}|$. An LPO $\mathrm{lpo} = (V, <, l)$ is enabled in (N, m_0) if and only if, for each step linearization $lpo' = (V, <', l)$ of lpo, the step sequence $\sigma_{lpo'}$ is enabled in (N, m_0).

Observe that $\mathfrak{Lpo}(N, m_0)$ is always sequentialization and prefix closed, i.e. every sequentialization and every prefix of an enabled LPO is again enabled w.r.t. (N, m_0). Moreover, the set of labels of $\mathfrak{Lpo}(N, m_0)$ is always finite. Therefore, when specifying the non-sequential behavior of a searched p/t-net by a partial language, this partial language must necessarily be sequentialization and prefix closed and must have a finite set of labels. Usually, we specify the non-sequential behavior by a set of concrete LPOs L which is *not* sequentialization and prefix closed and then consider the partial language $\mathcal{L}$ which emerges by adding all prefixes of sequentializations of LPOs in L. In this sense, the partial language L in Figure 2 specifies the non-sequential behavior of a searched p/t-net by extending it to its prefix and sequentialization closure $\mathcal{L} = \{\mathrm{lpo} \mid \mathrm{lpo}$ is a prefix of a sequentialization of an LPO in $L\} = \mathfrak{Lpo}(N, m_0)$. Both LPOs shown in this

Figure are enabled w.r.t. the marked p/t-net (N, m_0) shown in Figure 1. Thus, (N, m_0) solves the synthesis problem w.r.t. L.

3 Synthesis of P/T-Nets

We consider the problem of synthesizing a p/t-net from a partial language specifying its non-sequential behavior. We develop two different algorithms to compute a marked p/t-net (N, m_0) from a given set of LPOs L such that its prefix and sequentialization closure $\mathcal{L}$ satisfies $\mathcal{L} = \mathfrak{Lpo}(N, m_0)$ (if such a net exists). The idea to construct such a net (N, m_0) is as follows: the set of transitions of the searched net is given by the finite set of labels of L. Then each LPO in L is enabled w.r.t. the marked p/t-net consisting only of these transitions (having an empty set of places). We restrict the behavior of this net by adding places. Each place is defined by its initial marking and the weights on the arcs connecting it to each transition (Figure 3).

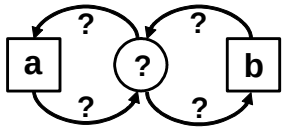

Fig. 3. An unknown place of a p/t-net.

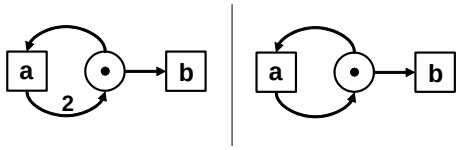

Fig. 4. Left: a feasible place. Right: a place which is not feasible.

Since the specified behavior given by the partial language L should be included in $\mathfrak{Lpo}(N, m_0)$, we only add places which do not exclude specified behavior. Thus we distinguish two kinds of places: in the case that there is an LPO in L which is not a run of the corresponding "one place"-net, this place restricts the behavior too much. Such a place is *non-feasible*. In the other case, the considered place is *feasible*.

Definition 5 (Feasible place). *Let L be a partial language over the finite set of labels T and let (N, m_p), $N = (\{p\}, T, F_p, W_p)$ be a marked p/t-net with only one place p. (N, m_p) is called associated to p. The place p is called* feasible (w.r.t. L)*, if $L \subseteq \mathfrak{Lpo}(N, m_p)$, otherwise* non-feasible (w.r.t. L).

Figure 4 shows on the left side a place which is feasible w.r.t. the partial language L in Figure 2. This is because, after the occurrence of a, the place is marked by two tokens. In this marking the step $a + b$ is enabled to occur (as specified by lpo_2). The place shown on the right side is non-feasible, because, after the occurrence of a, the place is again marked by only one token. In this marking the step $a + b$ is not enabled to occur. Thus lpo_2 is not enabled w.r.t. the one-place-net shown on the right side.

Adding only feasible places to the set of transitions given by the labels of a partial language L results in a p/t-net (N, m_0) for which holds $\mathcal{L} \subseteq \mathfrak{Lpo}(N, m_0)$. If (N, m_0) includes any non-feasible place, $\mathcal{L} = \mathfrak{Lpo}(N, m_0)$ is not possible. Adding places reduces $\mathfrak{Lpo}(N, m_0)$. Therefore, adding all feasible places leads to a p/t-net which is a solution of the synthesis problem or there is no solution. The first problem is to identify feasible places. Feasible places can be found through so called regions. A region is

a function defined on the structure of the language fulfilling certain properties. Every region corresponds to a feasible place. In this paper we will discuss two different definitions of regions of partial languages and discuss their algorithmic applicability. The second problem is that there are always infinitely many feasible places. One possibility to tackle this problem in region theory is a so called separation representation [14]. The idea is to represent non-specified behavior through a finite set of so called *wrong continuations* of $\mathcal{L}$. This set of wrong continuations has the property that if we exclude this behavior from a p/t-net (N, m_0) it holds $\mathfrak{Lpo}(N, m_0) \subseteq \mathcal{L}$. If there exists a solution of the synthesis problem, it is possible to exclude all wrong continuations through a finite set of feasible places. In this case the resulting net is a solution of the synthesis problem.

4 LPO Transition Regions

A naive approach to synthesize a p/t-net from a finite partial language L is to consider the set of step sequences S_L generated by the LPOs in L (see Figure 5). Following ideas in [8], where regions of trace languages are defined, it is possible to define regions of languages of step sequences [11].

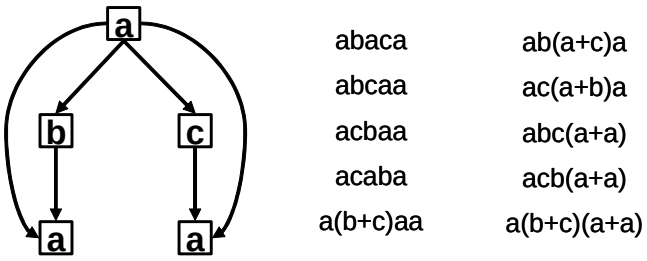

Fig. 5. Set of maximal step sequences generated by an LPO.

A region of a language of step sequences L' is simply a tuple of natural numbers which represents the initial marking of a place and the number of tokens each transition consumes respectively produces in that place, satisfying some property which ensures that no step sequence of the given language L' is prohibited by this place. The set of regions of L' defines the set of feasible places of L' [11]. Such regions, which are directly given by the parameters of a place, are called transition regions [14].

Definition 6 (Transition region). *Denoting $T = \{t_1, \ldots, t_m\}$ the transitions occurring in L', a* transition region *of L' is a tuple $\mathbf{r} = (r_0, \ldots, r_{2m}) \in \mathbb{N}^{2m+1}$ satisfying for every $\sigma = \tau_1 \ldots \tau_n \in L'$ and every $j \in \{1, \ldots, n\}$:*

$$(i) \qquad r_0 + \sum_{i=1}^{m}((\tau_1 + \ldots + \tau_{j-1})(t_i) \cdot r_i - (\tau_1 + \ldots + \tau_j)(t_i) \cdot r_{m+i}) \geq 0.$$

Every transition region $\mathbf{r}$ of L' defines a place p_r via $m_0(p_r) := r_0$, $W(t_i, p_r) := r_i$ and $W(p_r, t_i) := r_{m+i}$ for $1 \leqslant i \leqslant m$. The place p_r is called the corresponding place to $\mathbf{r}$.

In this paper for an algorithmic application we use an advanced approach. Calculating all step sequences from a given LPO is very costly. Furthermore the transition region definition leads to many needless inequations. Many of them are equal and or less restrictive then others. In particular, given a step sequence $\sigma = \tau_1 \dots \tau_n$, a different partitioning of the steps $\tau_1 \dots \tau_{(n-1)}$ does not change the corresponding inequation (i). Thus, it is sufficient to only count the number of transition occurrences in the steps $\tau_1 \dots \tau_{(n-1)}$ to formulate the corresponding inequation. Given a step sequence $\tau_1 \dots \tau_n$ the sum $\pi = \sum_{i=1}^{n-1} \tau_{(i)}$ we call a *prefix step*. Furthermore given a fixed prefix step π, it is sufficient to consider the occurrence of maximal steps τ_n after π as given by the partial language, since maximal steps form more restrictive inequations. Given a partial language L we compute the set of all possible prefix steps as the set of all parikh vectors corresponding to prefixes of L.

Definition 7 (Prefix step, maximal right continuation). *Given a prefix and sequentialization closed partial language $\mathcal{L}$ with the set of labels T. The set $\Pi_L = \{\pi \in \mathbb{N}^T \mid \exists\, lpo = (V, <, l) \in L : \forall t \in T\ \pi(t) = |\{v \in V |\ l(v) = t\}|\}$ contains all multi-sets corresponding to the number of transition occurrences in each LPO in $\mathcal{L}$. $\pi \in \Pi_L$ is called a* prefix step of $\mathcal{L}$.

Given a prefix step π, a multi-set $\tau \in \mathbb{N}^T$ is called a step after π*, if there exists an $lpo = (V, <, l) \in \mathcal{L}$ and a co-set $C \subseteq V$ for which holds: $\forall\ t \in T\ (\tau(t) = |\{v \in C |\ l(v) = t\}|\ \wedge\ \pi(t) = |\{v \in V |\ \exists\ c \in C : v < c,\ l(v) = t\}|)$. If C is a cut, then τ is called* cut after π.

Denote the set of all prefix steps of $\mathcal{L}$ together with all cuts after these prefix steps as $L^{\Pi} = \{(\pi, \tau) |\ \pi \in \Pi_L, \tau$ a cut after $\pi\}$. The set $L^{\Pi}_{max} = \{(\pi, \tau) \in L^{\Pi} |\ \forall (\pi', \tau') \in L^{\Pi} : \pi' = \pi \Rightarrow \tau' \not\geq \tau\}$ is the set of all prefix steps together with its maximal cuts after these prefixes. $(\pi, \tau) \in L^{\Pi}_{max}$ is called a maximal right continuation of $\mathcal{L}$.

Each cut of an LPO in L should be enabled after the occurrence of its prefix. Each cut together with its prefix is a candidate to be a maximal right continuation. For every prefix step π all maximal cuts after π are easy to calculate defining the set of maximal right continuations. Following these ideas leads to the definition of an *LPO transition regions*.

Definition 8 (LPO transition region). *Let L represent $\mathcal{L}$ and denote $T = \{t_1, \dots, t_m\}$ the transitions occurring in L. A* LPO transition region *of L is a tuple $\mathbf{r} = (r_0, \dots, r_{2m}) \in \mathbb{N}^{2m+1}$ satisfying for every maximal right continuation (π, τ) of $\mathcal{L}$:*

$$(ii) \qquad r_0 + \sum_{t \in T} ((\pi)(t) \cdot r_i - (\pi + \tau)(t) \cdot r_{m+i}) \geq 0.$$

Like transition regions every LPO transition region $\mathbf{r}$ of L defines a place p_r via $m_0(p_r) := r_0$, $W(t_i, p_r) := r_i$ and $W(p_r, t_i) := r_{m+i}$ for $1 \leqslant i \leqslant m$. Again, the place p_r is called the corresponding place to $\mathbf{r}$.

Theorem 1. *Given a partial language L, each LPO transition regions defines a feasible place and each feasible place is defined by an LPO transition region.*

Proof. Let $\mathcal{L}$ be a partial language and T the transitions occuring in $\mathcal{L}$.

Given an LPO transition region $\mathbf{r}$ and p_r its corresponding place. For each $lpo = (V, <, l) \in \mathcal{L}$ and each cut C of lpo there exists a maximal right continuation (π, τ) of $\mathcal{L}$ fulfilling $\forall\ t \in T\ (\tau(t) \geq |\{v \in C|\ l(v) = t\}|\ \wedge\ \pi(t) = |\{v \in V|\ \exists\ c \in C : v < c,\ l(v) = t\}|)$. It holds: $r_0 + \sum_{t\in T}((\pi)(t) \cdot r_i - (\pi + \tau)(t) \cdot r_{m+i}) \geq 0$ and $r_0 + \sum_{t\in T}((\pi)(t) \cdot r_i - (\pi + \tau)(t) \cdot r_{m+i}) = m_0(p_r) + \sum_{t\in T}((\pi)(t) \cdot W(t, p_r) - (\pi + \tau)(t) \cdot W(p_r, t) \leq m_0(p_r) + \sum_{v\in V\wedge v<C} W(l(v), p_r) - \sum_{v\in V\wedge v<C} W(p_r, l(v)) - \sum_{v\in C} W(p_r, l(v))$ such that $m_0(p_r) + \sum_{v\in V\wedge v<C} W(l(v), p_r) - \sum_{v\in V\wedge v<C} W(p_r, l(v)) \geq \sum_{v\in C} W(p_r, l(v))$. Thus lpo is enabled w.r.t p_r. p_r is feasible w.r.t. $\mathcal{L}$.

Given a feasible place p, define a tuple $\mathbf{r}$ by $r_0 := m_0(p)$, $r_i := W(t_i, p)$ and $r_{m+i} := W(p, t_i)$ for $1 \leq i \leq m$. For each maximal right continuation (π, τ) there exists a $lpo \in \mathcal{L}$ and a Cut C of lpo fulfilling $\forall\ t \in T\ (\tau(t) = |\{v \in C|\ l(v) = t\}|\ \wedge\ \pi(t) = |\{v \in V|\ \exists\ c \in C : v < c,\ l(v) = t\}|)$. Since each lpo is enabled w.r.t. the one place net containing only the place p, it holds: $m_0(p) + \sum_{v\in V\wedge v<C} W(l(v), p) - \sum_{v\in V\wedge v<C} W(p, l(v)) - \sum_{v\in C} W(p, l(v)) \geq 0$ and $m_0(p) + \sum_{v\in V\wedge v<C} W(l(v), p) - \sum_{v\in V\wedge v<C} W(p, l(v)) - \sum_{v\in C} W(p, l(v)) = r_0 + \sum_{t\in T}((\pi)(t) \cdot r_i - (\pi + \tau)(t) \cdot r_{m+i})$ such that $r_0 + \sum_{t\in T}((\pi)(t) \cdot r_i - (\pi + \tau)(t) \cdot r_{m+i}) \geq 0$. r is an LPO transition region defining p.

The set of LPO transition regions of L (resp. transition regions of L') can be characterized as the set of non-negative integral solutions of a homogenous linear inequation system $\mathbf{A}_L^{(ii)} \cdot \mathbf{r} \geq \mathbf{0}$ (resp. $\mathbf{A}_L^{(i)} \cdot \mathbf{r} \geq \mathbf{0}$.). Every inequation (ii) (resp. (i)) as given by Definition 8 (resp. Definition 6) defines a row of the inequation system. By this approach the set of all feasible places is computable. This is shown in Theorem 1 (resp. in [11]).

The ideas in [1, 5] to get an effective synthesis algorithm is to prohibit non-specified behavior by feasible places. In the original algorithm based on Definition 6 we try to calculate a region for each step sequence not specified in L' such that the corresponding place guarantees that this step sequence is not enabled. If $\tau_1 \dots \tau_n$ is not enabled then also $\tau_1 \dots \tau_n\tau_{n+1}$ and $\tau_1 \dots \tau'_n$ with $\tau_n \leq \tau'_n$ are not enabled. Therefore, it is sufficient to consider so called wrong continuations as defined in [11] instead of the set of all step sequences not in L'. The same principle holds, if we consider prefix steps and steps after prefix steps of a given partial language. Therefore, we only have to consider all possible prefix steps π and all minimal steps not being a step after π. We call this a *wrong continuation*:

Definition 9 (Wrong continuation). *Denote $L_{co}^{\Pi} = \{(\pi, \tau) \mid \pi$ a prefix step of $\mathcal{L}$, τ a step after $\pi\}$. The set of wrong continuations of L is defined by $L_{wrong} = \{(\pi, \tau') \notin L_{co}^{\Pi} \mid \exists(\pi, \tau) \in L_{co}^{\Pi},\ t \in T : \tau' = \tau + t\}$.*

In order to compute a feasible place which prohibits a wrong continuation (π, τ) of L, one defines so called *separating LPO transition regions* defining such places:

Definition 10 (Separating LPO transition region). *Let (π, τ) be a wrong continuation of L. An LPO transition region $\mathbf{r}$ of L is a* separating LPO transition region w.r.t. (π, τ) *if*

$$(iii) \qquad r_0 + \sum_{i=1}^{m}(\pi(t_i) \cdot r_i - (\pi + \tau)(t_i) \cdot r_{m+i}) < 0.$$

A separating region $\mathbf{r}$ w.r.t. (π, τ) can be calculated (if it exists) as a non-negative integer solution of a homogenous linear inequation system with integer coefficients of the form

$$\mathbf{A}_L^{(ii)} \cdot \mathbf{r} \geq \mathbf{0}$$
$$\mathbf{b}_{\pi\tau}^{(iii)} \cdot \mathbf{r} < \mathbf{0}.$$

The vector $\mathbf{b}_{\pi\tau}^{(iii)}$ is defined in such a way that $\mathbf{b}_{\pi\tau} \cdot \mathbf{r} < \mathbf{0} \Leftrightarrow (iii)$.

If there exists no non-negative integer solution of the system $\mathbf{A}_L^{(ii)} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{b}_{\pi\tau} \cdot \mathbf{r} < \mathbf{0}$, there exists no separating region w.r.t. (π, τ) and thus no feasible place prohibiting (π, τ). If there exists a non-negative integer solution of the system, any such solution defines a feasible place prohibiting (π, τ). If we choose one arbitrary separating region $\mathbf{r}_{\pi\tau}$ for each wrong continuation (π, τ) for which such a region exist, then we call the finite set of all these regions a *separation representation* (of the set of all regions). A place corresponding to each separating region of the separation representation is added to the synthesized net (N, m). Algorithmically, the places are introduced step by step according to a fixed ordering of the wrong continuations. If a wrong continuation is already prohibited by previously introduced places, it is not searched for a respective separating region.

If we denote the synthesized net by (N, m), it holds the following theorem:

Theorem 2. *There is a solution of the synthesis problem for the partial language $\mathcal{L}$ (defined by L) if and only if $\mathfrak{Lpo}(N, m) = \mathcal{L}$.*

Proof. It is only necessary to prove the *only if*-part. Assume there is a solution (N', m') of the synthesis problem for the partial language $\mathcal{L}$ (defined by L) and $\mathfrak{Lpo}(N, m) \neq \mathcal{L}$. This implies $\mathfrak{Lpo}(N, m) \supsetneq \mathcal{L}$, because we know $\mathfrak{Lpo}(N, m) \supseteq \mathcal{L}$ (Theorem 1).

We can distinguish two cases: either the set of maximal right continuations of the set of LPOs enabled in (N, m) coincides with L_{max}^{Π} or not. If the set of maximal right continuations of the set of LPOs enabled in (N, m) does not coincide with L_{max}^{Π}, then for some wrong continuation $(\pi, \tau) \in L_{wrong}$ there does not exist a separating region. In this case (π, τ) is given by an LPO not specified by L, which is enabled in (N', m'). Otherwise, (N', m') would have a place prohibiting this LPO, and this place would correspond to a separating region. This is a contradiction.

Let the set of maximal right continuations of the set of LPOs enabled in (N, m) coincide with L_{max}^{Π} and let $\mathrm{lpo} \notin \mathcal{L}$ be enabled in (N, m). This means each right continuations (π, τ) given by lpo is enabled in (N, m) (firing all transitions in π enables step τ). Since the set of enabled maximal right continuations of (N, m) coincides with L_{max}^{Π}, we conclude there exists $(\pi, \tau')L_{max}^{\Pi}$ with $\tau \leq \tau'$. Thus all right continuations of lpo are also enabled in (N', m'). Therefore also $\mathrm{lpo} \notin \mathcal{L}$ is enabled in (N', m') and thus $\mathfrak{Lpo}(N', m') \neq \mathcal{L}$. This is a contradiction.

We newly implemented the advanced algorithm to synthesize a p/t-net from a partial language using LPO transition regions in our tool Viptool [2] and compared the runtime to the version computing step sequences and using transition regions of step sequences as described in [11] (optimized by also only considering Parikh images of $\tau_1 \dots \tau_{n-1}$

in the inequation system $\mathbf{A}_L^{(i)} \cdot \mathbf{r} \geq \mathbf{0}$.). As expected the method using directly LPO transition regions was superior to the approach generating step sequences first and then using transition regions.

5 LPO Transition Regions vs. Token Flow regions

The main problem of considering LPO transition regions is the possible exponential number of cuts of a partial language compared to the number of events. Loosely speaking, the more parallelism in a given partial language, the worse is the runtime of the synthesis algorithm using LPO transition regions. To tackle this problem, we devolved another notion of regions so called *token flow regions* [13, 12, 11]. Again every token flow region corresponds to a feasible place. The idea of defining token flow regions of a given partial language L is as follows: assign to every edge (x, y) of an LPO in L a natural number representing the number of tokens which are produced by the occurrence of $l(x)$ and consumed by the occurrence of $l(y)$ in the place to be defined. Then the number of tokens consumed overall by a transition $l(y)$ in this place is given as the sum of the natural numbers assigned to ingoing edges of y. This number can then be interpreted as the weight of the arc connecting the new place with the transition $l(y)$. Similarly, the number of tokens produced overall by a transition $l(x)$ in this place is given as the sum of the natural numbers assigned to outgoing edges of x, and this number can then be interpreted as the weight of the arc connecting the transition $l(x)$ with the new place. Moreover, transitions can also consume tokens from the initial marking of the new place: In order to specify the number of such tokens, we extend an LPO by an *initial event* v_0 representing a transition producing the initial marking. The sum of the natural numbers assigned to outgoing edges (v_0, y) of the initial event v_0 can be interpreted as the initial marking of the new place. Transitions can produce tokens in the new place which remain in the final marking. In order to specify the number of such tokens, we extend an LPO by a *final event* v_{max} representing a transition consuming the final marking. Such an extended LPO by an initial and a final event we call a $\star$-extension of an LPO.

According to the above explanation, we can define a token flow region r by assigning in each LPO a natural number $r(x, y)$ to each edge (x, y) of all $\star$-extension of LPOs in L.

- The sum of the natural numbers assigned to ingoing edges of a node y we call the *in-token flow* of y.
- The sum of the natural numbers assigned to outgoing edges of a node x we call the *out-token flow* of x.
- The sum of the natural numbers assigned to outgoing edges of the initial node of an LPO lpo we call the *initial token flow* of lpo.

The value $r(x, y)$ we call the *token flow* between x and y. Since equally labeled nodes formalize occurrences of the same transition, this is well-defined only if equally labeled events have equal in-token flow (property IN) and equal out-token flow (property OUT). In particular all LPOs must have the same initial token flow (property INIT). Each such function r fulfilling (IN), (OUT) and (INIT) on L defines a feasible place p_r. We call p_r *corresponding place* of r.

Definition 11 (Token Flow Region). *Let L be a partial language and $E_L^\star$ the set of edges of the $\star$-extensions of all LPOs in L. A* token flow region *of L is a function $r : E_L^\star \to \mathbb{N}$ fulfilling (IN), (OUT) and (INIT) on L.*

It was shown in [13] that the set of places corresponding to token flow regions of a partial language equals the set of feasible places w.r.t. this partial language.

Theorem 3. *Given a partial language L. Each token flow region defines a feasible place and each feasible place is defined by a token flow region.*

The set of token flow regions can be computed as the set of non-negative integer solutions of a homogenous linear equation system with integer coefficients $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$. To compute a token flow region r, we need to assign a value $r(x, y)$ to every edge $e = (x, y)$ in the finite set of edges of the $\star$-extensions of the LPOs in the given partial language L. The vector $\mathbf{x}$ contains a variable x_i for each edge $e_i \in E_L^\star$ representing $r(e_i)$. We encode the properties (IN), (OUT) and (INIT) in the sense that r fulfills (IN), (OUT) and (INIT) on L if and only if $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$. This can be done by defining for pairs of equally labeled nodes a row of $\mathbf{A}_L$ counting the token flow on ingoing edges of one node positively and of the other node negatively. Similarly, another row of $\mathbf{A}_L$ counts the token flow on outgoing edges of one node positively and of the other node negatively can be defined. It is enough for each label t to ensure that the intoken (outtoken) flow of the first and second node with label t are equal that the intoken (outtoken) flow of the second and third node with label t are equal, and so on. The property (INIT) is assured just like the property (OUT), but considering the initial nodes of the LPOs in L.

By the above considerations the set of token flow regions r is in one-to-one correspondence to the set of non-negative integer solutions $\mathbf{x} = (x_1, \ldots, x_n)$ of $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$ via $r(e_i) = x_i$. This means, every feasible place can be computed by such a solution. The place corresponding to a solution $\mathbf{x}$ we denote by $p_\mathbf{x}$. Note that the number of rows of $\mathbf{A}_L$ linearly depends on the number of nodes of the LPOs [12]. The number of columns of $\mathbf{A}_L$ is equal to the number of edges $E_L^\star$.

To calculate a p/t-net which is a solution of our synthesis problem (if such a net exists) we use the wrong continuations given in the previous section. Again for each wrong continuation $(\pi, \tau) \in L_{wrong}$ we calculate a feasible place $p_\mathbf{x}$ as a non-negative integer solution of $\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$ which prohibits (π, τ). Therefore we need to translate the separating inequation (iii) into the new notion of token flow regions. For every transition t_i of the given language L we choose an example node x labeled by t_i. The set of all ingoing (outgoing) edges corresponds directly to the weight of the arc $W(t, p_\mathbf{x})$ ($W(p_\mathbf{x}, t)$). Given a fixed numbering of the edges of the LPOs in L and given a fixed example node x_{t_i} for each label t_i, we define sets T_i^{in} containing the numbers of ingoing edges into x_{t_i} and the sets T_i^{out} containing the numbers of outgoing edges of x_{t_i}. The set T^{init} contains the numbers of outgoing edges of the initial example node.

Definition 12 (Separating token flow region). *Let (π, τ) be a wrong continuation of a given partial language L. A token flow region $\mathbf{r}$ of L is a* separating region w.r.t. (π, τ) *if*

$$(iv) \qquad \sum_{j \in T^{init}} r(e_j) + \sum_{i=1}^{m} \Big(\sum_{j \in T_i^{out}} \pi(t_i) \cdot r(e_j) - \sum_{j \in T_i^{in}} (\pi + \tau)(t_i) \cdot r(e_j) \Big) < 0.$$

A separating token flow region $\mathbf{r}$ w.r.t. (π, τ) can be calculated (if it exists) as a non-negative integer solution of a homogenous linear inequation system with integer coefficients of the form

$$\mathbf{A}_L \cdot \mathbf{x} = \mathbf{0}$$
$$\mathbf{b}_{\pi\tau}^{(iv)} \cdot \mathbf{x} < \mathbf{0}.$$

The vector $\mathbf{b}_{\pi\tau}^{(iv)}$ is defined in such a way that $\mathbf{b}_{\pi\tau}^{(iv)} \cdot \mathbf{x} < \mathbf{0} \Leftrightarrow (iv)$.

Calculating for each wrong continuation a separating token flow region leads to the second algorithm described in this paper. As in the last section, feasible places corresponding to separating token flow regions are introduced step by step. If a wrong continuation is already prohibited by a previously added place, it is not searched for a respective separating region. If we denote the synthesized net by (N, m) and following the same arguments used in Theorem 2 it holds:

Theorem 4. *There is a solution of the synthesis problem for the partial language $\mathcal{L}$ (defined by L) if and only if $\mathfrak{Lpo}(N, m) = \mathcal{L}$.*

Again we implement this algorithm in our tool VipTool. As indicated by our experiences in [11], tests showed that the algorithm is much faster than the algorithm using a basis representation of token flow regions presented in [12].

The implementation of the new algorithm using separating token flow regions is used to compare the method with the algorithm using LPO transition regions. We ran three test series. A first one increasing the number of considered LPOs (e_1, e_2, e_3) (see Figure 6), a second one increasing the number of nodes in three concurrent sequences of nodes in one LPO (abc_n) (see Figure 7) and a third one increasing the number of sequential nodes in three LPOs (a_n, b_n, c_n) (see Figure 7).

The table shows the results of the tests. The first column describes the partial language given in each test. The columns 2-4 contain overall runtime, number of calculated places and the runtime for the calculation of one separating region for the algorithm using LPO transition regions. The columns 5-7 contain the same for the algorithm using token flow regions. The last column contains the runtime used to calculate the wrong continuations of the given language which is equal in both algorithms. In the first test series, given in the first three rows of the table, the algorithm using token flow regions performs slightly better than the algorithm using LPO transition regions. The languages represented by the LPOs contain a lot of concurrency. That means the number of arcs is relatively small and the number of possible prefixes is relatively high. This results in a small number of columns in the inequation systems solved by the algorithm using token flow regions (the number of rows is always linear in the number of nodes), and a huge number of rows in the inequation systems solved by the algorithm using LPO transition region (the number of columns is always linear in the number of transitions). The same but more distinctive holds for the second test series. In the third series it is the other way round. There is no concurrency allowed by the given languages. The algorithm using LPO transition regions is much faster than the algorithm using token flow regions.

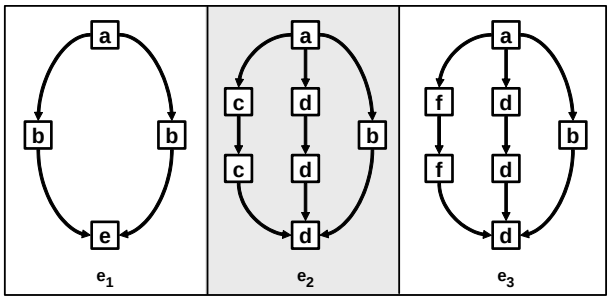

Fig. 6. Example LPOs (e_1, e_2, e_3).

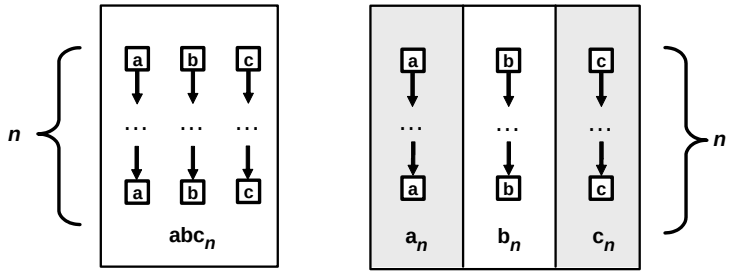

Fig. 7. Example LPOs. Left: One LPO modelling three concurrent sequences (abc_n). Right: Three LPOs modelling three alternative sequences (a_n, b_n, c_n).

Language	**Synthesis LPO transition regions**			**Synthesis Token flow regions**			
	complete runtime (ms)	number of places	runtime per equation system	complete runtime (ms)	number of places	runtime per equation system	building wrong continuations
$\{e_1\}$	46	3	1.8	43	4	0.6	40
$\{e_1, e_2\}$	90	11	3.4	62	11	0.8	53
$\{e_1, e_2, e_3\}$	250	14	13.4	129	14	4.8	62
$\{abc_2\}$	77	6	5.7	46	6	0.6	43
$\{abc_3\}$	529	6	79	60	6	1	53
$\{abc_4\}$	7166	6	1183	76	6	1	65
$\{a_5, b_5, c_5\}$	62	4	2.3	92	10	3.9	53
$\{a_6, b_6, c_6\}$	66	4	2.7	159	10	10.4	55
$\{a_7, b_7, c_7\}$	73	4	3	264	10	20.3	61

6 Conclusion

Both presented algorithms performed better then their previous versions described in [11]. The first algorithm using LPO transition regions does not have to translate the given partial language into a language of step sequences any more. The second algorithm using token flow regions benefits from using the adapted wrong continuation representation approach. The two algorithms developed in this paper use the two most promising combinations of a region definition together with a finite representation approach for the set of all feasible places. The algorithm using token flow regions performs fast if there is a lot of concurrency given in the partial language, the algorithm using LPO transition regions performs fast in the converse case.

References

1. E. Badouel and P. Darondeau. On the synthesis of general petri nets. Technical Report 3025, Inria, 1996.
2. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Viptool-homepage., 2008. http://www.informatik.ku-eichstaett.de/projekte/vip/.
3. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. of Informations and Systems*, E80-D(3):315–325, 1997.
4. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Hardware and petri nets: Application to asynchronous circuit design. In *ICATPN, LNCS 1825*, pages 1–15, 2000.
5. P. Darondeau. Deriving unbounded petri nets from formal languages. In *CONCUR, LNCS 1466*, pages 533–548, 1998.
6. 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.
7. J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4(2):428–498, 1981.
8. P. Hoogers, H. Kleijn, and P. Thiagarajan. A trace semantics for petri nets. *Information and Computation*, 117(1):98–114, 1995.
9. M. B. Josephs and D. P. Furey. A Programming Approach to the Design of Asynchronous Logic Blocks. In *Concurrency and Hardware Design, LNCS 2549*, pages 34–60, 2002.
10. A. Kiehn. On the interrelation between synchronized and non-synchronized behaviour of petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 24(1/2):3–18, 1988.
11. R. Lorenz, R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri Nets from finite partial languages. *Fundamenta Informaticae*, to appear, 2009.
12. R. Lorenz, R. Bergenthum, S. Mauser, and J. Desel. Synthesis of petri nets from finite partial languages. In *Proceedings of ACSD 2007*, 2007.
13. R. Lorenz and G. Juhás. Towards synthesis of petri nets from scenarios. In *ICATPN, LNCS 4024*, pages 302–321, 2006.
14. R. Lorenz, G. Juhás, and S. Mauser. How to Synthesize Nets from Languages - a Survey. In *Proceedings of the Wintersimulation Conference (WSC)*, 2007.
15. V. Pratt. Modelling concurrency with partial orders. *Int. Journal of Parallel Programming*, 15:33–71, 1986.
16. W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.
17. W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets.*, volume 625 of *Lecture Notes in Computer Science*. Springer, 1992.