Synthesis of bounded Petri Nets from Prime Event Structures with Cutting Context using Wrong Continuations

Robert Lorenz, Johannes Metzger, and Lev Sorokin

Department of Computer Science
University of Augsburg, Germany
`robert.lorenz@informatik.uni-augsburg.de`
`johannes.metzger@informatik.uni-augsburg.de`
`lev.sorokin@informatik.uni-augsburg.de`

Abstract. We consider the problem of synthesizing a bounded place / transition Petri net from a labelled prime event structure with cutting context, which actually is the most expressive notion for the specification of the nonsequential behaviour of concurrent systems allowing for a solution of the Petri net synthesis problem. The existing solution method is mainly of theoretical value and not applicable in practise in general. As a step towards practical application we develop an algorithm solving the synthesis problem using the technique of wrong continuations. We give a characterization of exact solutions, present an implementation of the algorithm and show first experimental results.

Keywords: Synthesis, Region Theory, Petri Net, Prime Event Structure, Cutting Context, Token Flow, Wrong Continuation

1 Introduction

In [9] the authors present a theory for the token flow region based synthesis of bounded place/transition Petri nets from labelled prime event structures (LPES) associated with a cutting context. The presented results generalize earlier results on the synthesis of Petri nets from partial languages [12], since they extend the class of non-sequential behaviour, for which Petri nets can be synthesized.

As usual in region based theory, the places of the synthesized net are computed as non-negative integral solutions of a linear inequation system. As mentioned in [9], it is immediately possible to construct a concrete synthesis algorithm from the presented theory using the so called basis representation of this inequation system. But, as experiences in the context of partial languages show, the basis representation in general produces complex-structured nets and leads to algorithms with inefficient run-time [1–5]. Moreover, it is not possible to decide directly, whether the synthesized net is an exact solution, and there are no parameters which can be used to influence the synthesis result. That means, the framework from [9] is mainly of theoretical value and, in general, not suitable for application in practise.

In [12] also another representation, the separation representation based on so called wrong continuations, is presented. It is shown that the separation representation results

in more efficient synthesis algorithms computing simpler nets. Moreover, it can be deduced directly, whether the synthesized net is an exact solution, and there are several parameters for fine tuning the algorithm.

As a step towards practical application we adapt the technique of wrong continuations from [12] to the synthesis approach from [9] and present an implementation which allows to adjust the computation w.r.t. several parameters. The paper is organized as follows: In section 2 we introduce basic mathematical notions. Then we briefly recall the synthesis approach from [9] in section 3 and define the separation representation based on wrong continuations for this approach in section 4. In section 5 we present the resulting synthesis algorithm together with an implementation. Finally, we show experimental result through applying the algorithm to the unfoldings of several example Petri nets in section 6.

2   Basic Notions

We use $\mathbb{N}$ to denote the set of *non-negative integers*. A *multiset* over a set A is a function $m : A \to \mathbb{N}$. For an element $a \in A$ the number $m(a)$ determines the number of occurrences of a in m. Addition $+$ on multisets is defined by $(m + m')(a) = m(a) + m'(a)$. The relation $\leq$ between multiset is defined through $m \leq m' \iff \exists m''(m + m'' = m')$. In case $m(a) > 0$ we also write $a \in m$. The multiset m satisfying $\forall a \in A : m(a) = 0$ we call *empty multiset*.

Given a binary relation $R \subseteq A \times A$ over A, the symbol R^+ denotes the *transitive closure* of R. A *directed graph* is a tuple $G = (V, \to)$, where V is its set of *nodes* and $\to \subseteq V \times V$ is its set of *arcs*. As usual, given a binary relation $\to$, we write $v \to w$ to denote $(v, w) \in \to$. In this case, v is called *pre-node* of w and w is called *post-node* of v. For $v \in V$ we denote by $^\bullet v = \{w \in V \mid w \to v\}$ the *preset* of v, and by $v^\bullet = \{w \in V \mid v \to w\}$ the *postset* of v.

A *partial order* is a directed graph $(V, <)$, where $< \subseteq V \times V$ is an irreflexive and transitive binary relation. In the context of this paper, a partial order is interpreted as an "earlier than"-relation between events. A node v is called *maximal* if $v^\bullet = \emptyset$, and *minimal* if $^\bullet v = \emptyset$. A subset $W \subseteq V$ is called *left-closed* if $\forall v, w \in V : (v \in W \wedge w < v) \implies w \in W$. For a left-closed subset $W \subseteq V$, the partial order $(W, <_{|W \times W})$ is called *prefix* of $(V, <)$, defined by W. The set $S(W) = \{v \in V \setminus W \mid w < v \implies w \in W\}$ is the set of the direct successors of the prefix defined by W. The *left-closure* of a subset W is given by the set $W \cup \{v \in V \mid \exists w \in W : v < w\}$. 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 $<_1 \subseteq <_2$ (note that a sequentialization in general does not refer to a total order or a sequence; a sequentialization being a total order is called a linearization). By $<_s \subseteq <$ we denote the smallest subset $<_s$ of $<$ which fulfils $(<_s)^+ = <$, called the *skeleton of* $<$. The *distance* $d(u, w)$ between two nodes u, w of partial order $(V, <)$ with $u < w$ is defined by $d(u, w) := \min\{n \mid u = v_0 <_s v_1 \ldots <_s v_n = w\}$.

A *labelled partial order (LPO) over* T is a triple $(V, <, l)$, where $(V, <)$ is a partial order, and $l : V \to T$ is a *labelling function* on V. We use all notations defined for partial orders also for LPOs. For a finite subset $W \subseteq V$, we define the multiset $l(W)$

by $l(W)(x) = |\{v \in W \mid l(v) = x\}|$. LPOs are used to represent partially ordered runs of Petri nets. Such runs are distinguished only up to isomorphism [6].

A *net* is a triple $N = (P, T, F)$, where P is a set of *places*, T is a set of *transitions*, satisfying $P \cap T = \emptyset$, and $F \subseteq (P \cup T) \times (T \cup P)$ is a *flow relation*. Places and transitions are called the *nodes* of N.

Definition 1 (Place/transition-net). *A* place/transition-net *(p/t-net) N is a quadruple* (P, T, F, W)*, where* (P, T, F) *is a net with finite sets of places and transitions, and* $W : F \to \mathbb{N} \setminus \{0\}$ *is a* weight function*. A* marking *of a p/t-net $N = (P, T, F, W)$ is a function $m : P \to \mathbb{N}$. 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*.*

We extend the weight function W to pairs of net elements $(x, y) \in (P \times T) \cup (T \times P)$ satisfying $(x, y) \notin F$ by $W(x, y) = 0$. A multiset of transitions is called a *transition step*. A transition step τ is *enabled to occur* in a marking m of N if $\forall p \in P : m(p) \geq \sum_{t \in T} \tau(t) \cdot W((p, t))$. If τ is enabled to occur in a marking m, then its *occurrence* leads to the new marking m' defined by $m'(p) := m(p) + \sum_{t \in T} \tau(t) \cdot (W((t, p)) - W((p, t)))$ for all $p \in P$.

An LPO $\mathrm{lpo} = (V, <, l)$ over T is *enabled to occur* in a marking m of N if $m(p) + \sum_{v' \in V'} (W(l(v'), p) - W(p, l(v'))) \geq \sum_{v \in S(V')} W(p, l(v))$ for each prefix $\mathrm{lpo}' = (V', <, l)$ of lpo and each place p. If lpo is enabled to occur in a marking m, then its *occurrence* leads to the new marking m' defined by $m'(p) := m(p) + \sum_{v \in V} (W(l(v), p) - W(p, l(v)))$ for all $p \in P$.

We use labelled prime event structures (LPES) to represent the non-sequential behaviour of p/t-nets. An LPES consists of a set of events labelled with action names, a partial order representing an "earlier than"-relation between events and a set of so-called consistency sets, where left-closed consistency sets represent single runs of a net. Events which are never in the same consistency set are assumed to be in conflict and to belong to alternative runs. Labels of events represent transition names.

Definition 2 (Labelled prime event structure [15]). *A* prime event structure (PES) *is a triple* $\mathrm{pes} = (E, \mathrm{Con}, \prec)$ *consisting of a set E of events, a partial order $\prec$ on E and a set* Con *of finite subsets of E satisfying:*

- $\forall e \in E : \{e' \mid e' \prec e\}$ *is finite.*
- $\forall e \in E : \{e\} \in \mathrm{Con}$.
- $Y \subseteq X \in \mathrm{Con} \Longrightarrow Y \in \mathrm{Con}$.
- $\forall e \in E : ((X \in \mathrm{Con}) \wedge (\exists e' \in X : e \prec e')) \Longrightarrow (X \cup \{e\} \in \mathrm{Con})$.

The elements of Con *are called* consistency sets*. An infinite set X is a* consistent subset *of E if $\forall Y \subseteq X, Y$ finite $: Y \in \mathrm{Con}$. The conflict relation $\#$ between events of* pes *is defined by $e \# e' \Leftrightarrow \{e, e'\} \notin \mathrm{Con}$.*

A tuple $(E, \mathrm{Con}, \prec, l)$, where $(E, \mathrm{Con}, \prec)$ is a PES and $l : E \to T$ is a labelling function on E, is called labelled prime event structure (LPES) *over T.*

As LPOs, LPES are distinguished only up to isomorphism [6]. An LPES, where its whole set of events E forms a consistent set, we interpret as an LPO, i.e. in this case

we omit the set of consistency sets Con. We denote the set of left-closed consistency sets of an LPES lpes $= (E, \text{Con}, \prec, l)$ by $\text{Con}_{pre} \subseteq \text{Con}$. If $C \in \text{Con}_{pre}$ is a left-closed consistency set, then $\text{lpo}_C = (C, \prec_{|C \times C}, l_{|C})$ is an LPO which we interpret as a run given by lpes. We define the *partial language corresponding to* lpes as the sequentialization closure of $\{\text{lpo}_C \mid C \in \text{Con}_{pre}\}$. For every event $e \in E$, the left closure of $\{e\}$ is a finite left-closed consistency set, which is called a *local* consistency set and is denoted by $[e]$.

For a set of events E' and $C \in \text{Con}_{pre}$ we denote by $C \oplus E'$ the set $C \cup E'$, if $C \cup E' \in \text{Con}_{pre}$ and $C \cap E' = \emptyset$. If $E' = \{e\}$, we also write $C \oplus e$ to denote $C \oplus \{e\}$. Such an E' is a *suffix* of C, and $C \oplus E'$ is an *extension* of C. Let $C, D \in \text{Con}_{pre}$ with $C \subseteq D$. The set $S_D(C) = \{v \in D \setminus C \mid w \prec v \implies w \in C\}$ is the set of the direct successors of C in D. For a left-closed subset $F \subseteq E$ and a subset of consistency sets $\text{Con}'_F \subseteq \text{Con}_F := \{C \in \text{Con} \mid C \subseteq F\}$ satisfying the properties of the definition of LPES, the LPES $(F, \text{Con}'_F, \prec_{|F \times F}, l_{|F})$ is called *prefix* of the LPES $(E, \text{Con}, \prec, l)$, defined by F and Con'_F.

In [6] the so-called token flow unfolding of a p/t-net, representing its non-sequential behaviour, is presented. The token flow unfolding is based on an LPES equipped with so-called token flows. Let lpes $= (E, \text{Con}, \prec, l)$ be an LPES and $N = (P, T, F, W, m_0)$ be a marked p/t-net. We interpret lpes as a model of the behaviour of N, where the events in E represent transition occurrences. A *token flow function* $x :\prec \rightarrow \mathbb{N}^P$ is a function assigning multisets of places of N to the arcs of lpes. For an arc (e, e') between transition occurrences e and e' the multiset $x(e, e')$ is intended to represent the token flow between these transition occurrences, that is to represent for each place the number of tokens which are produced by e and then consumed by e'. For a token flow function x, a consistency set $C \in \text{Con}_{pre}$ and an event $e \in C$ we denote

- $IN^x(e) := \sum_{e' \prec e} x(e', e)$ the x-*intoken flow of* e.
- $OUT^x_C(e) := \sum_{e \prec e', \, e' \in C} x(e, e')$ the x-*outtoken flow of* e w.r.t. C.

A prime token flow event structure is an LPES together with a token flow function. Since equally labelled events represent different occurrences of the same transition, they are required to have equal intoken flow. Since not all tokens which are produced by an event are consumed by further events, there is no such requirement for the outtoken flow. It is assumed that there is a unique initial event producing the initial marking.

Definition 3 (Prime token flow event structure [6]). *A* prime token flow event structure over T is a pair (lpes, x), where lpes $= (E, \text{Con}, \prec, l)$ has a unique minimal event e_{init} with $\forall e \neq e_{init} : \ l(e_{init}) \neq l(e)$ and $l(E \setminus \{e_{init}\}) \subseteq T$ and $x :\prec \rightarrow \mathbb{N}^P$ is a token flow function with $\forall e, e' \in E : \ l(e) = l(e') \Rightarrow IN^x(e) = IN^x(e')$.

Two events are called strongly identical (w.r.t. a token flow function), if they are labelled with the same action name and depend on the same events with identical token flow. In [6] it is shown that strongly identical events which are in conflict always lead to isomorphic processes.

Definition 4 (Strongly identical events [6]). *Let* $((E, \text{Con}, \prec, l), x)$ *be a prime token flow event structure. Two events* $e, e' \in E$ *fulfilling* $(l(e) = l(e')) \wedge (^{\bullet}e = {}^{\bullet}e') \wedge (\forall f \in {}^{\bullet}e : x(f, e) = x(f, e'))$ *are called strongly identical.*

A token flow unfolding of a marked p/t-net is a prime token flow event structure, in which intoken and outtoken flows are consistent with the arc weights resp. the initial marking of the net within each left-closed consistency set. It is also required that the token flow on a skeleton arc may not be zero, that means only real causal dependencies are represented in an unfolding.

Definition 5 (Token flow unfolding [6]). *Let (N, m_0), $N = (P, T, F, W)$, be a marked p/t-net. A token flow unfolding of (N, m_0) is a prime token flow event structure* (lpes, x) *over T,* $\text{lpes} = (E, \text{Con}, \prec, l)$*, satisfying:*

- *(U_{in}):* $\forall e \neq e_{init}, \forall p \in P : IN^x(e)(p) = W(p, l(e))$.
- *(U_{out}):* $\forall C \in \text{Con}_{pre}, \forall e \in C \setminus \{e_{init}\}, \forall p \in P : OUT_C^x(e)(p) \leq W(l(e), p)$.
- *(U_{init}):* $\forall C \in \text{Con}_{pre}, \forall p \in P : OUT_C^x(e_{init})(p) \leq m_0(p)$.
- *(U_{min}):* $\forall (e, e') \in \prec_s: (\exists p \in P : x(e, e')(p) \geq 1)$.
- *(U_{id}): There are no strongly identical events e, e' (w.r.t. x) satisfying $\{e, e'\} \notin \text{Con}$.*

Token flow unfoldings are distinguished only up to isomorphism [6]. There exists a maximal (in general infinite) token flow unfolding $\text{Unf}_{max}(N, m_0)$ (w.r.t. a given prefix relation, see [6] for details), which is unique up to isomorphism. For each finite left-closed consistency set C of Unf_{max}, the LPO lpo_C is a run of N and for each run lpo of N there is a left-closed consistency set C of Unf_{max} with $\text{lpo} = \text{lpo}_C$. For each finite left-closed consistency set C of Unf_{max} the multiset of places

$$\text{Mark}(C)(p) := m_0(p) + \sum_{e \in C}(W(l(e), p) - W(p, l(e))) \quad (p \in P)$$

is a reachable marking of (N, m_0), called the *final marking* of C. Every final marking in Unf_{max} is reachable in (N, m_0), and every reachable marking is a final marking in Unf_{max}.

Since the maximal unfolding is infinite whenever the original net has infinite behaviour, there are approaches for constructing finite and complete prefixes. The essential feature of the existing unfolding algorithms computing finite, complete prefixes is the use of cutoff events, beyond which the unfolding can be truncated without loss of information. In [10] a parametric setup, called *cutting context*, is proposed in which completeness and cutoff events can be discussed in a uniform, general and algorithm-independent way (in [6] the concept of cutting context was adapted to token flow unfoldings). In the following, we briefly recall the notions which are relevant in the context of this paper.

Let Unf_{max} be the maximal token flow unfolding of a marked p/t-net (N, m_0). We denote by Con and Con_{pre} the sets of consistency sets and left-closed consistency sets of Unf_{max}. A *cutting context* is a triple $\Theta = (\approx, \lhd, \{C_e\}_{e \in E})$, where:

1. $\approx$ is an equivalence relation on Con_{pre}, capturing the information which is intended to be retained in a complete prefix. In the standard case $\approx = \approx_{mar}$, this is the set of reachable markings, i.e. $C \approx_{mar} C'$ if $\text{Mark}(C) = \text{Mark}(C')$.
2. $\lhd$ is a so-called *adequate order* on Con_{pre} which refines $\subset$. All $\lhd$-minimal left-closed consistency sets in each equivalence class of $\approx$ will be preserved in a complete prefix (in algorithms, $\subset$ is often used directly).

3. $\approx$ and $\lhd$ are preserved by finite extensions of left-closed consistency sets.
4. $\{C_e\}_{e \in E}$ is a family of subsets of Con_{pre} specifying the set of left-closed consistency sets used to decide whether an event can be designated as a cutoff event. In the standard case, C_e contains the local consistency sets of Unf_{max}.

Roughly spoken, a prefix of Unf_{max} is *complete*, if each equivalence class w.r.t. $\approx$ is represented once in it. Hence, for the relation $\approx_{mar}$, each reachable marking is represented by a left-closed consistency set of a complete prefix.

With these notions, cutoff events can be defined in a "static way" without referring to a specific algorithm building the unfolding. The set CutOff of static cutoff events is defined together with the set Feas of feasible events. *Feasible events* are precisely those events whose causal predecessors are not cutoff events, and as such must be included in the prefix determined by the static cutoff events. An event e is a *static cutoff event*, if it is feasible, and there is $C \in C_e$ such that $C \subseteq \mathrm{Feas} \setminus \mathrm{CutOff}$, $C \approx [e]$, and $C \lhd [e]$. The token flow unfolding Unf_Θ defined by the set of events Feas is called the *canonical prefix of* Unf_{max}. In [6] it is shown that Unf_Θ is complete, finite and uniquely determined by the cutting context Θ, if (N, m_0) is bounded, $\{C_e\}_{e \in E}$ contains all local left-closed consistency sets and $\approx = \approx_{mar}$. As argued in [6], it is also possible to use $\subset$ instead of $\lhd$. Although $\subset$ does not fulfill the third property of a cutting context, there is a canonical prefix w.r.t. $\subset$.

$[3] \approx [1]$
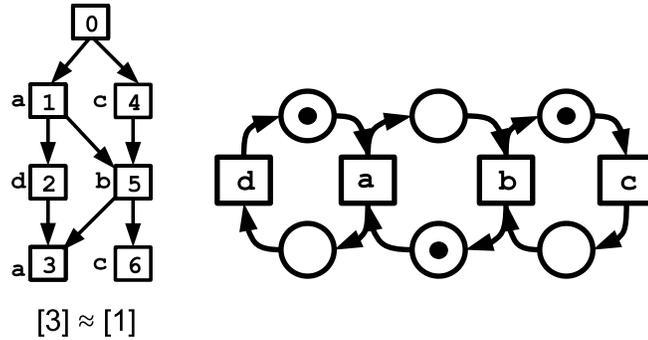
Fig. 1. A p/t-net (right side) together with the LPES underlying the complete finite prefix of its maximal unfolding (left side). Note that the LPES contains no conflicts here.

Figure 1 shows a p/t-net (N, m_0) together with the LPES lpes underlying the complete finite prefix of its maximal unfolding. The LPES lpes contains no conflicts, i.e. the set of all events forms a consistency set. Event numbers are drawn inside and event labels outside nodes. The only cutoff event is e_3 with $[e_3] \approx_{mar} [e_1]$.

In the following section we present a finite notion for the specification of the infinite nonsequential behaviour of concurrent systems. It is based on the idea of complete finite prefixes of the unfolding of bounded Petri nets as recalled above. It uses an LPES together with a set of cutoff events and can be parametrized according to given cutting

26

context. For simplicity, in the rest of the paper we use a cutting context with $\approx=\approx_{mar}$ and $\subset$ instead of a general adequate order $\lhd$. All definitions and theorems can easily be generalized to other choices of $\approx$ and $\lhd$.

3 Synthesis from labelled prime event structures

In this section we briefly recall the synthesis approach from [9]. In [9] the authors follow the traditional region based synthesis approach: Given a specification of the behaviour of a system based on runs over a finite set of action names, the action names are used as the transitions of the searched Petri net. It remains to find a suitable finite set of places, each place with initial marking and connections via arcs and arc-weights to all transitions. Places are found in a three-step-approach:

– First, the set of feasible places is defined. A place is feasible, if it does not prohibit some of the specified behaviour. The set of feasible places usually is infinite.
– Second, so called regions of the specification are defined as non-negative integral solutions of a linear homogenous equation system $\mathbf{A} \cdot \mathbf{x} = \mathbf{0}$, such that each region defines a feasible place and each feasible place is defined by a region. The rows of this equation system we call *constraints*.
– Third, a finite representation, i.e. a finite set of regions representing the infinite set of all regions, is constructed. While in [9] the so-called basis representation is used, in this paper we will employ the so-called separation representation (details later).

In this paper we use the finite representation of the non-sequential behaviour of a concurrent system from [9] as input for the synthesis of p/t-nets. This representation is more expressive and more compact than earlier presented models for this purpose, as for example LPO-terms and partial languages [12]. In particular, it may represent the non-sequential behaviour of arbitrary bounded p/t-nets. The basic idea for this representation is to use a finite LPES representing a finite set of runs together with cutoff events in order to specify repeated behaviour. A cutoff event is a maximal event of the LPES indicating that a repeated marking is reached, which was already seen before.

Definition 6 (LPES with cutoff-list [9]). *An* LPES *with cutoff list* is a finite LPES $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ *together with a finite* cutoff-list $\mathrm{Cut} = (D_i, e_i)_{i \in I}$ *consisting of maximal events $e_i \in E$ and left-closed consistency sets D_i with $D_i \subset [e_i]$ (with some finite index set I). We assume that* lpes *has a unique minimal event e_0 with empty label.*

An example of an LPES with cutoff-list is shown in Figure 1. In general, each LPES underlying the complete finite prefix of the maximal unfolding of a bounded p/t-net together with its list of cutoff events $(e_i)_{i \in I}$ and the set of consistency sets $(D_i)_{i \in I}$ with $[e_i] \approx D_i$ forms an LPES with cutoff list. We will use an LPES with cutoff-list to synthesize a p/t-net having the runs specified by lpes and satisfying $\mathrm{Mark}(D_i) = \mathrm{Mark}([e_i])$ for each i. An LPES with cutoff-list serves as a finite specification of an infinite LPES, a so-called *completion*.

Definition 7 (Completion). *Let* $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ *be an LPES with* cutoff-list $\mathrm{Cut} = (D_i, e_i)_{i \in I}$. *A completion of* $(\mathrm{lpes}, \mathrm{Cut})$ *is an LPES* $\mathrm{lpes}' = (E', \mathrm{Con}', \prec', l')$ *with the following properties:*

1. lpes *is a prefix of* lpes'.
2. *If* $C \in \mathrm{Con}_{pre}$ *such that* $\forall i : e_i \notin C$, *and* $C \oplus e \in \mathrm{Con}'_{pre}$ *for some event e, then* $C \oplus e \in \mathrm{Con}_{pre}$.
3. *Let* $\approx$ *be the smallest equivalence relation on* Con'_{pre} *satisfying:*
   (a) $[e_i] \approx D_i$ *for each* i.
   (b) $\approx$ *is preserved by finite extensions.*
   Then $\approx$ *satisfies:*
   (c) *If* $C' \in \mathrm{Con}'_{pre}$, *then there is* $C \in \mathrm{Con}_{pre}$ *such that* $\forall i : e_i \notin C$ *and* $C' \approx C$.
   (d) *For* $C' \approx C''$ *and each extension* $E' = \{e'\}$ *of* C' *there is an extension* $E'' = \{e''\}$ *of* C'' *with* $l'(e') = l'(e'')$ *(note that this property is symmetric).*

This definition is stronger than the one given in [9] concerning the requirements 2. and 3.(c). We need these additional requirements in order to prove the main theorem giving a characertization of exact solutions (which were not considered in [9]).

As an example, the maximal unfolding of a bounded p/t-net is a completion of its complete finite prefix using $\approx = \approx_{mar}$. The equivalence relation $\approx$ represents the reachable states and it is required that all reachable states are represented in lpes. The cutoff-list $\mathrm{Cut} = (D_i, e_i)_{i \in I}$ specifies that exactly the extensions of D_i should also be possible extensions of $[e_i]$ in a completion. Note that not all maximal events of lpes need to be cutoff events. If a maximal event e is not a cutoff event, then the final marking of $[e]$ is intended not to enable any further transition occurrence. In the rest of the paper we consider the following formal problem statement:

– **Given**: A finite LPES $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ over a finite alphabet of transition names T together with a cutoff-list $\mathrm{Cut} = (D_i, e_i)_{i \in I}$.
– **Searched**: A marked p/t-net (N, m_0) with set of transitions T such that the partial language corresponding to its unfolding is minimal with the property, that it includes the partial language corresponding to a completion of $(\mathrm{lpes}, \mathrm{Cut})$.

In [9] the authors moreover require, that the unfolding of the synthesized net has a strong structural relationship to a completion of $(\mathrm{lpes}, \mathrm{Cut})$ (called *preservation of prefixes and concurrency*), which we do not need to consider here.

In order to present the constraints of the equation system defining a feasible place p (resp. a region), we denote by $E = \{e_0, \ldots, e_n\}$ the list of events, by $\{C_0, \ldots, C_m\}$ the list of left-closed consistency sets which are maximal w.r.t. the subset-relation and by $\{t_0, \ldots, t_l\}$ the list of transitions. The table 1 shows the variables together with their constraints and their interpretation as used in the approach of [9]. In contrast to [9] we use additional variables which directly represent the parameters of the defined place p (initial marking $m_0(p)$, flow weights $W(p,t)$ and $W(t,p)$). The required equations $\mathrm{Mark}(D_i)(p) = \mathrm{Mark}([e_i])(p)$ can be directly encoded using the above variables.

4   Separation representation and wrong continuations

In this section we adapt the notions of wrong continuations and separation representation w.r.t. partial language based specifications [12] to LPES based specifications as

Table 1. List of variables and constraints for the computation of regions according to [9].

variables	interpretation	constraints
$r_{i,k}$	tokens produced by e_i and not consumed by events in C_k for $e_i \in C_k$	
$a_{i,j}$	tokens produced by e_i and consumed by e_j for each edge $e_i \prec e_j$	
in_j	tokens consumed by e_j	$in_j = \sum_{e_i \prec e_j} a_{i,j}$
$out_{i,k}$	tokens produced by e_i w.r.t. C_k for $e_i \in C_k$	$out_{i,k} = r_{i,k} + \sum_{e_j \in C_k, \, e_i \prec e_j} a_{i,j}$
m	initial marking of the place ($m = m_0(p)$)	$m = out_{0,k}$ for each k
pt_h	tokens consumed by t_h ($pt_h = W(p, t_h)$)	$pt_h = in_j$ for each j with $l(e_j) = t_h$
tp_h	tokens produced by t_h ($tp_h = W(t_h, p)$)	$tp_h = out_{i,k}$ for each i with $l(e_i) = t_h$ and k with $e_i \in C_k$
	the marking after execution of $[e_i]$ equals the marking after execution of D_i for each pair $(D_i, e_i) \in \mathrm{Cut}$	$\mathrm{Mark}(D_i)(p) = \mathrm{Mark}([e_i])(p)$

considered in this paper. As the main result of this paper, we give a characterization of exact solutions of the synthesis problem.

In the following we denote by $\mathbf{x}$ a region as defined in the previous section (that means an integral non-negative solution of the equation system presented in the previous section) and by $p_{\mathbf{x}}$ the place defined by $\mathbf{x}$. An idea to get a finite representation is to separate behavior specified by $(\mathrm{lpes}, \mathrm{Cut})$ from behavior not specified by $(\mathrm{lpes}, \mathrm{Cut})$ by a finite set of regions (see [12] in the context of partial languages). The resulting representation is called *separation representation*. To derive a separation representation, an appropriate finite set $\{\mathrm{lpo}_1, \ldots, \mathrm{lpo}_o\}$ of LPOs with the following properties is defined:

- The LPOs lpo_i are not runs specified by $(\mathrm{lpes}, \mathrm{Cut})$
- Each LPO lpo_i extends a run specified by $(\mathrm{lpes}, \mathrm{Cut})$ by one event.

Then for each lpo_i one tries to find a region $\mathbf{x}$ such that $p_{\mathbf{x}}$ prohibits lpo_i (that means lpo_i is not a run w.r.t. $p_{\mathbf{x}}$). If such a region exists, $p_{\mathbf{x}}$ is added to the separation representation. The LPOs lpo_i are called *wrong continuations*. The aim is to define them in such a way that an exact solution of the synthesis problem exists if and only if each wrong continuation can be prohibited by a place. A solution (N, m_0) is an exact solution, if it does not have runs which are not specified by $(\mathrm{lpes}, \mathrm{Cut})$.

Definition 8 (Exact solution). *A solution (N, m_0) of the considered synthesis problem is called* exact solution*, if the partial language corresponding to its maximal unfolding equals the partial language corresponding to a completion of $(\mathrm{lpes}, \mathrm{Cut})$.*

For example, a bounded p/t-net is an exact solution w.r.t. the specification given by the complete finite prefix of its maximal unfolding. Formally we split a wrong continuation into a prefix belonging to the partial language of lpes, a subsequently enabled step of transitions and an additional transition which should be prohibited.

Definition 9 (Wrong continuation). *Let* $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ *be a finite LPES over a finite alphabet of transition names T together with a cutoff-list* $\mathrm{Cut} = (D_i, e_i)_{i \in I}$.
A wrong continuation *of* $(\mathrm{lpes}, \mathrm{Cut})$ *is a triple* (C, τ, t), *where*

- *C is a left-closed consistency set C of* lpes *such that* lpo_C *does not contain a cutoff event ($\forall i : e_i \notin C$).*
- *There is a maximal consistency set D of* lpes *with $C \subseteq D$, $\tau \leq l(S_D(C))$ and $\tau(t) = l(S_D(C))(t)$ (τ may be the empty multiset).*
- *There is no maximal consistency set D' of* lpes *with $C \subseteq D'$, $\tau \leq l(S_{D'}(C))$ and $\tau(t) < l(S_{D'}(C))(t)$.*

We call lpo_C *the* prefix *and τ *the* follower step *of the wrong continuation.*

According to the specification $(\mathrm{lpes}, \mathrm{Cut})$, the prefix lpo_C of a wrong continuation (C, τ, t) leads to marking enabling τ, but not enabling $\tau + t$. Some of the wrong continuations of $(\mathrm{lpes}, \mathrm{Cut})$ shown in Figure 1 are: $w_1 = (\{e_0\}, \{\}, b)$ (prohibiting b in the initial marking), $w_2 = (\{e_0\}, \{\}, d)$ (prohibiting d in the initial marking) and $w_3 = (\{e_0\}, \{e_1\}, a)$ (prohibiting $2a$ in the initial marking).

Theorem 1. *A solution (N, m_0) of the considered synthesis problem is an exact solution if and only if for each wrong continuation (C, τ, t) of $(\mathrm{lpes}, \mathrm{Cut})$ the prefix lpo_C leads to marking not enabling $\tau + t$.*

Proof. Let (lpes_u, x) be the maximal token flow unfolding of (N, m_0) and lpes_c be a completion of $(\mathrm{lpes}, \mathrm{Cut})$ such that the partial language corresponding to lpes_u includes the partial language corresponding to lpes_c.

"if": Assume that (N, m_0) is not an exact solution. Let $\mathrm{lpo} = (V \cup \{e\}, \prec, l)$ belong to the partial language corresponding to lpes_u, but not to the partial language corresponding to lpes_c, such that $(V, \prec, l)$ belongs to the partial language corresponding to lpes_c. Let $\approx$ be the equivalence relation on lpes_c from the definition of completions. Using the properties 3.(a),3.(b),3.(d) of the definition of completions and the properties of places defined by regions, it can be proven iteratively, that $C \approx C' \Longrightarrow \mathrm{Mark}(C) = \mathrm{Mark}(C')$. By property 3.(c) there is a left-closed consistency set V' of lpes with $V' \approx V$ and containing no cutoff-events. Since lpo does not belong to lpes_c and from property 2. of the definition of completions, there is a wrong continuation $(C, \tau, l(e))$ with $C \subseteq V'$ and $l(C) + \tau = l(V')$. From $\mathrm{Mark}(V') = \mathrm{Mark}(V)$ we deduce that the transition $l(e)$ is enabled in $\mathrm{Mark}(V')$, i.e. lpo_C leads to a marking enabling $\tau + l(e)$.

"only if": Assume that there is a wrong continuation (C, τ, t) of $(\mathrm{lpes}, \mathrm{Cut})$, such that after occurrence of the prefix lpo_C the transition step $\tau + t$ is enabled in (N, m_0). Let $\mathrm{lpo}_C = (C, \prec, l)$ and $W \subseteq S_D(C)$ be a subset of direct successors of lpo_C in a maximal consistency set D of lpes such that $l(W) = \tau$ and $C \oplus W \in \mathrm{Con}$ (such W exists by the definition of wrong continuations). The LPO $\mathrm{lpo}_{C \oplus W}$ belongs to the partial language corresponding to lpes and, by assumption, its occurrence leads to a marking enabling t. That means, there is an extension $C \oplus W \oplus e$ with $l(e) = t$ such that $\mathrm{lpo}_{C \oplus W \oplus e}$ belongs to the partial language corresponding to lpes_u. On the other side, according to the property 2. of the definition of completions, $\mathrm{lpo}_{C \oplus W \oplus e}$ does not belong to the partial language corresponding to lpes_c. That means (N, m_0) is not an exact solution. $\square$

Consider a wrong continuation (C, τ, t). Our aim is to compute a region $\mathbf{x}$ such that after the occurrence of the prefix lpo_C the marking of $p_\mathbf{x}$ does not enable the transition step $\tau + t$. This can directly be expressed by a linear constraint using the variables of the linear equation system defining regions:

Table 2. Constraint for the computation of a region w.r.t. a wrong continuation.

variables	interpretation	constraints
	lpo_C does not enable $\tau + t$ for the wrong continuation (C, τ, t)	$\mathrm{Mark}(C)(p) < \sum_{h=0}^{l}(\tau+t)(t_h) \cdot pt_h$

In order to compute such a region $\mathbf{x}$ w.r.t. a wrong continuation (C, τ, t), we add the above constraint to the linear equation system defining regions. For each wrong continuation a separate constraint leading to a separate linear inequation system needs to be considered. Considering for example Figure 1, the wrong continuation $w_1 = (\{e_0\}, \{\}, b)$ is prohibited by the place in $c^\bullet \cap {}^\bullet b$.

5 Synthesis algorithm and implementation

Before formulating the synthesis algorithm and presenting its implementation we consider two parameters influencing the solution net resulting from a separation representation [12].

5.1 Using a target function

Considering the linear inequation system corresponding to a wrong continuation, a non-negative integral solution can be computed which minimizes a given linear target function ϕ. Such a target function can be used to produce "simple" places [12].

In the context of this paper, for example, it is possible to minimize the token flows $a_{i,j}$ along edges and the residual token flows $r_{i,k}$. In our case, a target function has the following general form

$$\phi = \alpha_1 \cdot \sum_{i,k} \gamma_{i,k} \cdot r_{i,k} + \alpha_2 \cdot \sum_{i,j} \beta_{i,j} \cdot a_{i,j},$$

with $\alpha_i, \gamma_{i,k}, \beta_{i,j} \in \mathbb{R}_0^+$. The use of different target functions leads to different solutions. Our experiments have shown that the simple target function $\phi_{rest} = \sum_{i,k} r_{i,k}$ is a good choice in most cases in order to compute "simple" places, since minimizing the residual token flows $r_{i,k}$ produces places prohibiting many wrong continuations at once. In several cases also the target function $\phi_{penalty} = \sum_{i,j} \beta_{i,j} \cdot a_{i,j}$, where $\beta_{i,j} = 1$ if $d(e_i, e_j) = 1$ and $\beta_{i,j} = 50$ else, yields good results, since it leads to places representing only local dependencies. It is also possible to distinguish between edges with different distances using $\beta_{i,j} = d(e_i, e_j)$, or to combine these target functions.

31

5.2 Determining the order of wrong continuations

In general, a place does not prohibit only one wrong continuation. If a wrong continuation w is prohibited by a place, which was already computed previously, it is not necessary to compute a separate solution for w. Consider two wrong continuations w_1, w_2. It is possible that the solution p_1 prohibiting w_1 also prohibits w_2, but that the solution p_2 prohibiting w_2 does not prohibit w_1. In such a case, it is better to consider w_1 first, since this order leads to a net with less places. That means, the order in which wrong continuations are considered, the so-called *wct-ordering*, has an influence on the synthesis result [12].

Unfortunately, there is still no theory on optimizing the wct-ordering in the above sense. Therefore, in our experiments we considered several different orderings $\leq_{wct}$ based on the following definitions for wrong continuations $w_1 = (C_1, \tau_1, t_1)$, $w_2 = (C_2, \tau_2, t_2)$:

- $w_1 <_C w_2 :\Leftrightarrow |C_1| < |C_2|$ (can be used to consider small prefixes before big prefixes, or vice versa)
- $w_1 <_{dC} w_2 :\Leftrightarrow d(C_1) < d(C_2)$, where $d(C) := max\{d(e_0, e_i) \mid e_i \in C\}$ (can be used to consider short prefixes before long prefixes, or vice versa)
- $w_1 <_\tau w_2 :\Leftrightarrow |\tau_1| < |\tau_2|$ (can be used to consider small follower steps before big follower steps, or vice versa)

These components can be combined in different ways to derive orderings of wrong continuations for the synthesis algorithm. In our experiments we identified the following wct-ordering as a good choice:

$$\leq_{wct} = <_{dC} \cup (=_{dC} \cap <_C) \cup (=_C \cap =_{dC} \cap <_\tau)$$

Note that this is a partial order. Unordered wrong continuations are ordered randomely by the algorithm. As a reference, the tool also offers the possibility to use the completely random order $\leq_{wct}^{rnd}$.

5.3 Synthesis algorithm and implementation

We are now able to present the synthesis algorithm. First we construct the equation system defined by the basic constraints from table 1 (line 1) and start with an empty set S of solutions (line 2). In the next step we create the list of wrong continuations $\{w_1, \ldots, w_n\}$ ordered w.r.t. $\leq_{wct}$ (line 3; prefixes are computed using a breadth-first approach; for the follower step we consider the direct successors of a prefix, see Definition 9). Now we can iterate over this list (lines 4 - 19). In the i-th iteration we consider the wrong continuation w_i:

- first we test, whether w_i is prohibited by one of the already computed solutions in S (lines 6 - 11).
- if this is the case, we consider the next wrong continuation (lines 12 - 13).
- if this is not the case (lines 14 - 17), we add the constraint corresponding to w_i (table 2) to the inequation system (line 15), add the computed solution to S (if one exists; line 16), and remove the constraint (line 17).

32

Require: LPES with cutoff list (lpes, Cut), target function ϕ, ordering $\leq_{wct}$
1: $litok \leftarrow getFlowConstraints(\text{lpes}, \text{Cut})$
2: $S \leftarrow \emptyset$
3: $\{w_1, \ldots, w_n\} \leftarrow getWCTs((\text{lpes}, \text{Cut}), \leq_{wct})$
4: **for** i = 1 **to** n **do**
5:    $redundant \leftarrow false$
6:    **for** $s \in S$ **do**
7:       **if** $wct_i.satisfied(s)$ **then**
8:          $redundant \leftarrow true$
9:          $break$
10:       **end if**
11:    **end for**
12:    **if** $redundant$ **then**
13:       continue
14:    **else**
15:       $litok.add(wct_i)$
16:       $S \leftarrow S \cup lpsolve(litok, \phi)$
17:       $litok.remove(wct_i)$
18:    **end if**
19: **end for**
20: $(N, m_0) \leftarrow extractPNet(S)$
21: **return** (N, m_0)

Algorithm 1: Synthesis algorithm

Finally we retrieve the solution net from set of solutions S (line 20). All wrong continuations, which cannot be prohibited, are reported (this is not shown in the algorithm for a compact representation).

We use the following optimizations (adapted from [12]):

- If (C, τ, t) is a wrong continuation and a place p prohibits the step $\tau + t$ after the execution of lpo_C, then p also prohibits each step $\tau' + t$ after the execution of lpo_C for $\tau \leq \tau'$. In this case, the wrong continuations of the form (C, τ', t) need not be considered. This feature is implemented through considering small follower steps first in the used wct-ordering.
- For two prefixes lpo_C and $\text{lpo}_{C'}$ with $l(C) = l(C')$, there holds: after the execution of lpo_C, a step $\tau + t$ can be prohibited if and only if it can be prohibited after the execution of $\text{lpo}_{C'}$. Considering several wrong continuations with such prefixes, their follower steps can be combined. This feature is implemented through comparing $l(C)$ for each wrong continuation with previously considered wrong continuations.

Altogether, the runtime of the algorithm is exponential in the number of events of the LPES in general, since there are exponentially many prefixes to consider. The runtime can be reduced using heuristics and by grouping wrong continuations, but this is a topic of further research.

The synthesis algorithm is implemented in JAVA as a command line tool. For computing solutions of linear inequation systems we used the free solver lp_solve [7] and the framework Java ILP [14]. The tool can be downloaded together with a readme file

and the examples presented in the next section [13]. The readme file explains installation and usage of the tool. For the input we use text files specifying $(\mathrm{lpes}, \mathrm{Cut})$, ϕ and $\leq_{wct}$ using a simple text format. For the output we use the PNML-format [8]. The resulting p/t-net can be visualized with ePNK [11] in Eclipse.

6 Experimental results

We now briefly present two kinds of experiments evaluating the presented tool. First we considered several p/t-nets and constructed the complete finite prefix of their maximal unfolding. This complete finite prefix then served as input LPES with cutoff-list for the synthesis of a new p/t-net, which we compared with the original net. We used nets which were drawn as simple and intuitive as possible. That means, our goal was to test, whether the used synthesis algorithm was able to reproduce the original net and to analyse the differences.

Figures 1 and 2 show two of the considered p/t-nets together with the complete finite prefix of their maximal unfoldings.
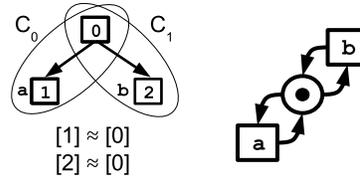
Fig. 2. A p/t-net together with the complete finite prefix of its maximal unfolding as input specification $(\mathrm{lpes}, \mathrm{Cut})$ for synthesis. The LPES contains two maximal consistency sets C_0, C_1.

It turned out that using Φ_{rest} as target function and the wct-ordering $\leq_{wct}$ (from the last section) it was possible to exactly rediscover the original p/t-net in both presented examples and also for several other small nets. Note here that for the specification in Figure 1 there is no other existing notion, which can be used for synthesis. In particular, this specification cannot be expressed by an LPO-term [12].

In case of bigger nets with arc weights and more complicated dependency realtions, some additional and / or different places were computed due to a still not optimized combination of target function and wct-ordering (which is a topic of future research). For example, we considered the net in part (a) of Figure 3, constructed the complete finite prefix of its maximal unfolding and got the following result using different target functions:

– Part (b): This net was computed using the target function Φ_{rest} and the wct-ordering $\leq_{wct}$. It has one additional and unnessecary gray-coloured place. It prohibits the transition step $2c$ after the occurrence of a, while there is another place in the original net which prohibits the transition steps $2b$, $b+c$ and $2c$ after the occurrence of a.
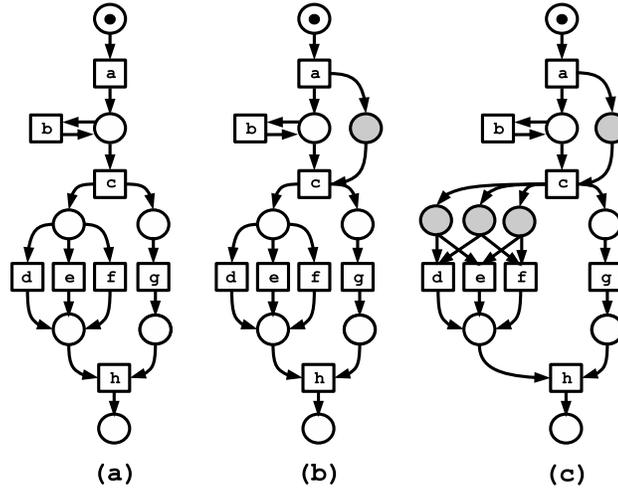
Fig. 3. A p/t-net (part (a)) and two synthesis result from the complete finite prefix of its maximal unfolding (parts (b) and (c)) with additional places depending on the used target function.

That means, computing these places in a different order would result in the original net. In order to avoid such unnessecary places it is nessecary to find additional local rules for determining the wct-ordering.

– Part (c): This net was computed using the target function $\Phi_{penalty}$ and the wct-ordering $\leq_{wct}$. It has three more additional and unnessecary gray-coloured places. These places represent the conflict between the three transitions d, e, f. The original place representing this conflict is non-optimal w.r.t. $\Phi_{penalty}$, since it has more connections to transitions. On the other side, the three gray places are non-optimal w.r.t. Φ_{rest}, since tokens remain in some of them after ocurrence of d, e or f.

Another example is shown in Figure 4. Part (a) shows the original net and part (b) the synthesis result using the target function Φ_{rest} and the wct-ordering $\leq_{wct}$. It has one additional and unnessecary gray-coloured place which prohibits the transition step e in the initial marking, but it is easy to see that also some other places of the original net probit transition step e in the initial marking. That means (again) that computing these places in a different order would result in the original net. Note that the gray place is non-optimal w.r.t. $\Phi_{penalty}$, but on the other side, also several places of the original net are non-optimal w.r.t. $\Phi_{penalty}$. The net was already considered in [1] where the synthesis result was the same using a synthesis algorithm based in sequences.

These first experiments show that it is possible to synthesize simple and intuitive nets using the presented approach, if the local dependency relations between events are simple. For several more complex situations the choice of the target function and the wct-ordering influences the synthesis result and it depends on the local dependency relations, which choice is the better on. The local choice of target function and wct-ordering will be a topic of further research.
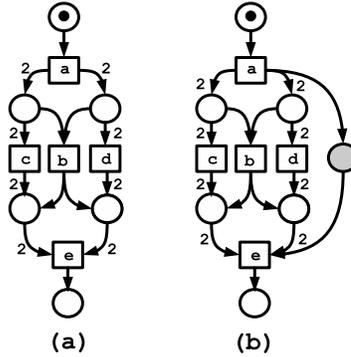
Fig. 4. A p/t-net (part (a)) and a synthesis result from the complete finite prefix of its maximal unfolding (parts (b)) with an additional place.

Second we evaluated the runtime of the tool using several parametrized p/tnets and their unfolding. Figure 5 shows one of these nets, where the parameter is the number n of tokens in the place in ${}^\bullet a \cap {}^\bullet b$. The net has 2^n different runs and can be seen as one of the "worst cases" for the presented synthesis approach. Our algorithm was able to exactly rediscover the original p/t-net. Of course runtime increased rapidly when increasing the parameter n due to the amount of prefixes and wrong continuations which have to be computed.
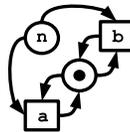
Fig. 5. A p/t-net having the language $(a + b)^n$ and 2^n different runs.

The following table shows the runtime for different choices of n using Φ_{rest} as target function and the wct-ordering $\leq^{rnd}_{wct}$. The average runtime was computed over several runs of the algorithm using a system with i7-6500U CPU having 2.50 GHz.

n	WCT constraints	Average runtime
1	8	≈ 40 ms
2	24	≈ 50 ms
3	64	≈ 130 ms
4	160	≈ 700 ms
5	384	≈ 5200 ms
6	896	≈ 76000 ms

This example shows that there is still work to do, but there are already first approaches to improve such situations (not implemented yet). For example, the above situation can be first considered as an infinite iteration (easy to solve, see above), and then the number of iterations can be restricted by an appropriate place.

7 Conclusion and future work

In this paper we characterized exact solutions of the region based synthesis of bounded p/t-nets from LPES with cutoff-list using wrong continuations and presented an implementation of the according synthesis algorithm. Experimental results are promising, showing that simple dependency relations among transitions can be rediscovered exactly. For more complicated dependency situations there are two parameters of the algorithm - target function and wct-ordering - which can be adjusted. The local choice of these parameters is a topic of further research.

In order to improve runtime, we plan to develop heuristics for the grouping of wrong continuations (that means, a solution is searched prohibiting a group of several wrong continuation at once) and for the consideration of a small subset of the set of all wrong continuations (which is likely to "represent" the whole set). As a basis for both aspects we are working on a theory concerning an optimal ordering of wrong continuations.

Concerning practical applications, we are developing a two-step approach in the area of process mining using our synthesis framework:

– First step (preprocessing): Construct an LPES with cutoff-list from an event log through detecting loops, causality, concurrency and noise. Since LPES with cutoff-list is a more general model than p/t-nets, causal structures can be detected with less loss of information than with approaches using directly Petri net based models. That means an LPES with cutoff-list has a low representational bias in the context of process mining.
– Second step (synthesis): Synthesize a Petri net from the result of the first step.

References

1. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer, 2007.
2. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. *Fundam. Inform.*, 88(4):437–468, 2008.
3. R. Bergenthum, J. Desel, and S. Mauser. Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. *T. Petri Nets and Other Models of Concurrency*, 3:216–243, 2009.
4. R. Bergenthum, J. Desel, S. Mauser, and R. Lorenz. Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. *Fundam. Inform.*, 95(1):187–217, 2009.
5. R. Bergenthum and S. Mauser. Synthesis of Petri Nets from Infinite Partial Languages with VipTool. In N. Lohmann and K. Wolf, editors, *AWPN*, volume 380 of *CEUR Workshop Proceedings*, pages 81–86. CEUR-WS.org, 2008.
6. R. Bergenthum, S. Mauser, R. Lorenz, and G. Juhás. Unfolding Semantics of Petri Nets Based on Token Flows. *Fundam. Inform.*, 94(3-4):331–360, 2009.

7. M. Berkelaar. lp_solve. http://lpsolve.sourceforge.net/5.5/.

8. L. d'informatique de Paris 6 / Paris Nord. PNML - Framework, 2009. http://pnml.lip6.fr.

9. G. Juhás and R. Lorenz. Synthesis of bounded Petri Nets from Prime Event Structures with Cutting Context. In W. M. P. van der Aalst, R. Bergenthum, and J. Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016 Satellite event of the conferences: 37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and 16th International Conference on Application of Concurrency to System Design ACSD 2016, Torun, Poland, June 20-21, 2016.*, volume 1592 of *CEUR Workshop Proceedings*, pages 58–77. CEUR-WS.org, 2016.

10. V. Khomenko, M. Koutny, and W. Vogler. Canonical Prefixes of Petri Net Unfoldings. *Acta Inf.*, 40(2):95–118, 2003.

11. E. Kindler. The ePNK: An Extensible Petri Net Tool for PNML. In *Applications and Theory of Petri Nets - 32nd International Conference, PETRI NETS 2011, Newcastle, UK, June 20-24, 2011. Proceedings*, pages 318–327, 2011.

12. R. Lorenz, J. Desel, and G. Juhás. Models from Scenarios. In *Transactions on Petri Nets and Other Models of Concurrency VII*, pages 314–371. Springer Berlin Heidelberg, 2013.

13. R. Lorenz, J. Metzger, and L. Sorokin. Prosyn, 2016. https://www.informatik.uni-augsburg.de/lehrstuehle/inf/projekte/prosyn/.

14. M. Lukasiewycz. Java ILP, 2008. http://javailp.sourceforge.net/.

15. G. Winskel. Event Structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.