Process Mining Based on Regions of Languages

Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser

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

Abstract. In this paper we give an overview, how to apply region based methods for the synthesis of Petri nets from languages to process mining.
The research domain of process mining aims at constructing a process model from an event log, such that the process model can reproduce the log, and does not allow for much more behaviour than shown in the log. We here consider Petri nets to represent process models. Event logs can be interpreted as finite languages. Region based synthesis methods can be used to construct a Petri net from a language generating the minimal net behaviour including the given language. Therefore, it seems natural to apply such methods in the process mining domain. There are several different region based methods in literature yielding different Petri nets. We adapt these methods to the process mining domain and compare them concerning efficiency and usefulness of the resulting Petri net.

1 Introduction

Often, business information systems log all performed activities together with the respective cases the activities belong to in so called event logs. These event logs can be used to identify the actual workflows of the system. In particular, they can be used to generate a workflow definition which matches the actual flow of work. The generation of a workflow definition from event logs is known as *process mining*. Application of process mining and underlying algorithms gained increasing attention in the last years, see e.g. [23] and [22]. There are a number of process mining tools, mostly implemented in the ProM framework [18].

The formal problem of generating a system model from a description of its behaviour is often referred to as synthesis problem. Workflows are often defined in terms of Petri nets [21]. Synthesis of Petri nets is studied since the 1980s [9, 10, 8]. Algorithms for Petri net synthesis have often been applied in hardware design [5, 4].

Obviously, process mining and Petri net synthesis are closely related problems. Mining aims at a system model which has at least the behaviour given by the log and does not allow for much more behaviour. In the optimal case the system has minimal additional behaviour. The goal is to find such a system which is not too complex, i.e., small in terms of its number of components. This is necessary, because practitioners in industry are interested in controllable and interpretable reference models. Apparently, sometimes a trade-off between the size of the model and the additional behaviour has to be found.

One of the main differences in Petri net synthesis is that one is interested in a Petri net representing exactly the specified behaviour. Petri net synthesis was originally assuming a behavioural description in terms of transition systems. For a transition system, sets of nodes called *regions* can be identified. Each region refers to a place of the synthesized net. Analogous approaches in the context of process mining are presented in [24, 20]. Since process mining usually does not start with a transition system, i.e., a state based description of behaviour, but rather with a set of sequences, i.e., a language based description of behaviour, the original synthesis algorithms are not immediately applicable. In [24, 20] artificial states are introduced to the log in order to generate a transition system. Then synthesis algorithms transforming the state-based model into a Petri net, that exactly mimics the behaviour of the transition system, are applied. The problem is that these algorithms include reproduction of the state structure of the transition system, although the artificial states of the transition system are not specified in the log. In many cases this leads to a bias of the process mining result. However, there also exist research results on algorithmic Petri net synthesis from languages [6, 1, 2, 12]. In these approaches, regions are defined on languages. It seems natural to directly use these approaches for process mining, because logs can directly be interpreted as languages. The aim of this paper is to adjust such language based synthesis algorithms to solve the process mining problem.

The idea of language based synthesis algorithms is as follows: The transitions of the constructed net are given by the characters of the language. Adding places restricts the behaviour of the net. Only places not prohibiting sequences of the language are added. Thus the resulting net includes the behaviour specified by the language. This approach is very well suited for process mining. If the language is given by an event log, the constructed net reproduces the log. The algorithmic methods of language based synthesis, deciding which places are added to the net, will turn out to guarantee, that the constructed net does not allow for much more behaviour than shown in the log.

We will present methods for process mining adapted from language based synthesis methods. We compare the methods and give a complete overview of the applicability of regions of languages to the process mining problem.

The process mining algorithms discussed in this paper are completely based on formal methods of Petri net theory guaranteeing reliable results. By contrast, most existing process mining approaches are partly based on heuristic methods, although they borrow techniques from formally developed research areas such as machine learning and grammatical inference [23, 17], neural networks and statistics [23, 3], or Petri net algorithms [7, 24, 20].

The paper is organized as follows. Section 2 provides formal definitions. Section 3 motivates and explains language based synthesis algorithms and defines the process mining problem tackled in this paper more formally. A preliminary solution to the process mining problem defines nets with infinitely many places. In Section 4, two methods for identifying finite (and small) subsets of places which suffice for representing the behaviour of the given event log are presented and compared. Finally, the conclusion completes the overview of the applicability of language based synthesis for process mining and provides a bridge from the more theoretical considerations of this paper to practically useful algorithms.

2 Preliminaries

In this section we recall the basic notions of languages, event logs and place/transition Petri nets. An *alphabet* is a finite set A. The set of all *strings (words)* over an alphabet A is denoted by A^*. The *empty word* is denoted by λ. A subset $L \subseteq A^*$ is called *language over* A. For a word $w \in A^*$, $|w|$ denotes the *length of* w and $|w|_a$ denotes the number of occurrences of $a \in A$ in w. Given two words v, w, we call v *prefix* of w if there exists a word u such that $vu = w$. A language L is *prefix-closed*, if for every $w \in L$ each prefix of w also belongs to L.

The following definition is a formalization of typical log files. Since we focus on the control flow of activities (their ordering), we abstract from some additional information such as originators of events and time stamps of events.

Definition 1 (Event log). *Let T be a finite set of* activities *and C be a finite set of* cases. *An* event *is an element of $T \times C$. An* event log *is an element of $(T \times C)^*$.*

Given a case $c \in C$ we define the function $p_c : T \times C \to T$ by $p_c(t, c') = t$ if $c = c'$ and $p_c(t, c') = \lambda$ else. Given an event log $\sigma = e_1 \dots e_n \in (T \times C)^*$ we define the process language $L(\sigma)$ of σ *by* $L(\sigma) = \{p_c(e_1) \dots p_c(e_i) \mid i \leq n, c \in C\} \subseteq T^*$.

Observe that the process language of an event log is finite and prefix closed. It represents the control flow of the activities given by the log. Each case of the log adds one word (drawn in italic in the following example) over the set of activities together with its prefixes to the process language. Of course several cases may add the same words to the process language (e.g. the word $abba$ in the following example). Therefore in real life, the control flow given by an event log is a bag of words. In this paper we do not distinguish words w.r.t. their frequencies. Therefore, the process language is defined as a set of words. The following example log will serve as a running example.

event log (activity,case):
(a,1) (b,1) (a,2) (b,1) (a,3) (d,3) (a,4) (c,2) (d,2) (e,1) (c,3) (b,4) (e,3) (e,2) (b,4) (e,4)
process language:
a ab abb *abbe* ac acd *acde* ad adc *adce*

Example 1.

A *net* is a triple $N = (P, T, F)$, where P is a (possibly infinite) set of *places*, T is a finite set of *transitions* satisfying $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. Let $x \in P \cup T$ be an element. The *preset* $\bullet x$ is the set $\{y \in P \cup T \mid (y, x) \in F\}$, and the *post-set* $x\bullet$ is the set $\{y \in P \cup T \mid (x, y) \in F\}$.

Definition 2 (Place/transition-net). *A* place/transition-net *(*p/t-net*) is a quadruple $N = (P, T, F, W)$, where (P, T, F) is a net, 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 function $m : P \to \mathbb{N}_0$ 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*. As usual, places are drawn as circles including tokens representing the initial marking, transitions are depicted as rectangles and the flow relation is shown by arcs which have annotated the values of the weight function (the weight 1 is not shown).

A transition $t \in T$ is *enabled to occur in a marking* m of a p/t-net N if $m(p) \geq W(p,t)$ for every place $p \in \bullet t$. If a transition t is enabled to occur in a marking m, then its *occurrence* leads to the new marking m' defined by $m'(p) = m(p) - W(p,t) + W(t,p)$ for every $p \in P$. That means t *consumes* $W(p,t)$ tokens from p and *produces* $W(t,p)$ tokens in p. We write $m \xrightarrow{t} m'$ to denote that t is enabled to occur in m and that its occurrence leads to m'. A finite sequence of transitions $w = t_1 \dots t_n,\ n \in \mathbb{N}$, is called an *occurrence sequence enabled in* m *and leading to* m_n if there exists a sequence of markings $m_1, \dots, m_n$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} m_n$. In this case m_k $(1 \geq k \geq n)$ is given by $m_k(p) = m(p) + \sum_{i=1}^{k}(W(t_i,p) - W(p,t_i))$ for $p \in P$. The set of all occurrence sequences enabled in the initial marking m_0 of a marked p/t-net (N, m_0) forms a language over T and is denoted by $L(N, m_0)$. Observe that $L(N, m_0)$ is prefix closed. $L(N, m_0)$ models the (sequential) behaviour of (N, m_0). There is the following straightforward characterization of $L(N, m_0)$:

Lemma 1. *Let* (N, m_0) *be a marked p/t-net. Then* $w = t_1 \dots t_n \in T^*$, $n \in \mathbb{N}$, *is in* $L(N, m_0)$ *if and only if for each* $1 \leqslant k \leqslant n$ *and each* $p \in P$ *there holds:*
$m_0(p) + \sum_{i=1}^{k-1}(W(t_i,p) - W(p,t_i)) \geq W(p,t_k)$.
Let $w = t_1 \dots t_n \in L(N, m_0)$ *and* $t \in T$. *Then* $wt \notin L(N, m_0)$ *if and only if for one* $p \in P$ *there holds:* $m_0(p) + \sum_{i=1}^{n}(W(t_i,p) - W(p,t_i)) < W(p,t)$.

Figure 1 shows a marked p/t-net having exactly the process language of the running example as its language of occurrence sequences. That means this Petri net model is a process model describing the process given by the event log.

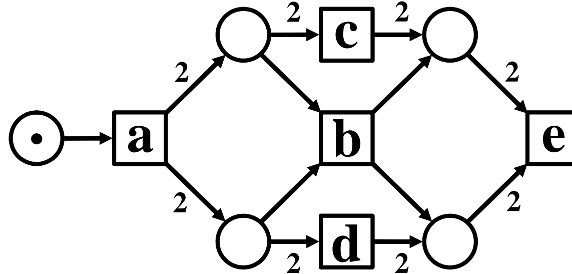

Fig. 1. Petri net model describing the event log of the running example.

3 Theory of Regions Applied to Process Mining

In this section we formally define the process mining problem and show how the classical language based theory of regions can be adjusted to solve this problem. The regions

definition introduced in this section is an adaption of the definition in [6, 1] to the setting of process mining. In [6, 1] languages given by regular expressions instead of finite languages given by event logs and pure nets instead of p/t-nets are considered. In the following section we will develop concrete algorithms from the considerations presented in this section.

Process mining aims at the construction of a process model from an event log which is able to reproduce the behaviour (the process) of the log, and does not allow for much more behaviour than shown in the log. Moreover, as argued already in the introduction, the process model in the ideal case serves as a reference model which can be interpreted by practitioners. Therefore the model should be as small as possible. As we will show, there is a trade-off between the size of the constructed model and the degree of the match of the behaviour generated by the model and the log. In this paper we formalize process models as Petri nets and consider the following *process mining problem*:

Given: An event log σ. **Searched:** A preferably small finite marked p/t-net (N, m_0) such that **(1)** $L(\sigma) \subseteq L(N, m_0)$ and **(2)** $L(N, m_0) \setminus L(\sigma)$ is small.

In the following we will consider a fixed process language $L(\sigma)$ given by an event log σ with set of activities T. An adequate method to solve the process mining problem w.r.t. $L(\sigma)$ is applying synthesis algorithms using regions of languages. Region-based synthesis algorithms all follow the same principle: Given a language $L(\sigma)$, the set of transitions of the searched net is given by the set of characters T used in $L(\sigma)$. Then each $w \in L(\sigma)$ is an enabled occurrence sequence w.r.t. the resulting marked p/t-net $(\emptyset, T, \emptyset, \emptyset, \emptyset)$ consisting only of these transitions (having an empty set of places), because there are no causal dependencies between the transitions. That means $L(\sigma) \subseteq L(\emptyset, T, \emptyset, \emptyset, \emptyset) = T^*$ and thus $(\emptyset, T, \emptyset, \emptyset, \emptyset)$ fulfills (1). But this net has many enabled occurrence sequences not specified in $L(\sigma)$, because $L(\sigma)$ is finite. That means (2) is not fulfilled. Thus, the behaviour of this net is restricted by adding places leading to a marked p/t-net (N, m_0), $N = (P, T, F, W)$. Every place $p \in P$ is defined by its initial marking $m_0(p)$ and the weights $W(p, t)$ and $W(t, p)$ of the arcs connecting them to each transition $t \in T$. In order to preserve (1), only places are added, which do not prohibit sequences of $L(\sigma)$. Such places are called *feasible (w.r.t. $L(\sigma)$)*.

Definition 3 (Feasible place). *Let (N, m_p), $N = (\{p\}, T, F_p, W_p)$ be a marked p/t-net with only one place p (F_p, W_p, m_p are defined according to the definition of p). The place p is called* feasible (w.r.t. $L(\sigma)$)*, if $L(\sigma) \subseteq L(N, m_p)$, otherwise* non-feasible.

Adding only feasible places yields a net fulfilling (1), while adding any non-feasible place yields a net not fulfilling (1). The more feasible places we add the smaller is the set $L(N, m_0) \setminus L(\sigma)$. Adding *all* feasible places minimizes $L(N, m_0) \setminus L(\sigma)$ (preserving (1)). That means the resulting net – called the *saturated feasible net* – is an optimal solution for the process mining problem concerning (1) and (2) (but it is not small).

Definition 4 (Saturated feasible net). *The marked p/t-net (N_{sat}, m_{sat}), $N_{sat} = (P, T, F, W)$, such that P is the set of all places feasible w.r.t. $L(\sigma)$ is called* saturated feasible (w.r.t. $L(\sigma)$) *(F, W, m_0 are defined according to the definitions of the feasible places).*

Theorem 1. *The saturated feasible p/t-net (N_{sat}, m_{sat}) w.r.t. $L(\sigma)$ satisfies $L(\sigma) \subseteq L(N_{sat}, m_{sat})$ and $\forall (N, m_0) : \ L(N, m_0) \subsetneq L(N_{sat}, m_{sat}) \Longrightarrow L(\sigma) \nsubseteq L(N, m_0)$.*

In particular there holds either $L(N_{sat}, m_{sat}) = L(\sigma)$ or there is no p/t-net (N, m_0) satisfying $L(N, m_0) = L(\sigma)$.

The problem is that (N_{sat}, m_{sat}) is not finite. Therefore, Theorem 1 has only theoretical value. Moreover, in the above considerations we did not include the formulated aim to construct a small p/t-net. Here the trade-off between the size of the constructed net and (2) comes into play: The more feasible places we add the better (2) is reached, but the bigger becomes the constructed net. The central question is which feasible places should be added.

There are two basic algorithmic approaches throughout the literature to synthesize a finite net (N, m_0) from a language. In both approaches (N, m_0) represents (N_{sat}, m_{sat}) in the sense that $L(N, m_0) = L(\sigma) \Leftrightarrow L(N_{sat}, m_{sat}) = L(\sigma)$. The crucial idea in these approaches is to define feasible places structurally on the level of the given language. Every feasible place is defined by a so called *region* of the language. A region 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 occurrence sequence of the given language is prohibited by this place.

Definition 5 (Region). *Denoting $T = \{t_1, \ldots, t_n\}$, a* region *of $L(\sigma)$ is a tuple $\mathbf{r} = (r_0, \ldots, r_{2n}) \in \mathbb{N}^{2n+1}$ satisfying for every $wt \in L(\sigma)$ ($w \in L(\sigma), t \in T$):*

$$(*) \qquad r_0 + \sum_{i=1}^{n} (|w|_{t_i} \cdot r_i - |wt|_{t_i} \cdot r_{n+i}) \geq 0.$$

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

From Lemma 1, we deduce:

Theorem 2. *Each place corresponding to a region of $L(\sigma)$ is feasible w.r.t. $L(\sigma)$. and each place feasible w.r.t. $L(\sigma)$ corresponds to a region of $L(\sigma)$.*

Thus, the set of feasible places w.r.t. $L(\sigma)$ corresponds to the set of regions of $L(\sigma)$. The set of regions can be characterized as the set of non-negative integral solutions of a homogenous linear inequation system

$$\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}.$$

The matrix $\mathbf{A}_{L(\sigma)}$ consists of rows $\mathbf{a}_{wt} = (a_{wt,0}, \ldots, a_{wt,2n})$ for all $wt \in L(\sigma)$, satisfying $\mathbf{a}_{wt} \cdot \mathbf{r} \geq \mathbf{0} \Leftrightarrow (*)$. This is achieved by setting for each $wt \in L(\sigma)$:

$$a_{wt,i} = \begin{cases} 1 & \text{for } i = 0, \\ |w|_{t_i} & \text{for } i = 1, \ldots, n \\ -|wt|_{t_{i-n}} & \text{for } i = n+1, \ldots, 2n. \end{cases}$$

The next table shows this inequation system for the process language of Example 1.

a	$r_0 - r_6$	≥ 0
ab	$r_0 + r_1 - r_6 - r_7$	≥ 0
abb	$r_0 + r_1 + r_2 - r_6 - 2r_7$	≥ 0
abbe	$r_0 + r_1 + 2r_2 - r_6 - 2r_7 - r_{10}$	≥ 0
ac	$r_0 + r_1 - r_6 - r_8$	≥ 0
acd	$r_0 + r_1 + r_3 - r_6 - r_8 - r_9$	≥ 0
acde	$r_0 + r_1 + r_3 + r_4 - r_6 - r_8 - r_9 - r_{10}$	≥ 0
ad	$r_0 + r_1 - r_6 - r_9$	≥ 0
adc	$r_0 + r_1 + r_4 - r_6 - r_9 - r_8$	≥ 0
adce	$r_0 + r_1 + r_4 + r_3 - r_6 - r_9 - r_8 - r_{10}$	≥ 0

The inequation system may have less inequations than the number of words in the considered language. In this example the inequations for $acde$ and $adce$ coincide.

Altogether the idea of adding feasible places is very well suited for process mining, because this guarantees (1). But so far it is not clear which feasible places should be added such that the resulting net does not become too big and (2) is still satisfactorily fulfilled. The two mentioned region based approaches to synthesize a finite net from a language propose two different procedures to add a finite set of feasible places. These procedures are the candidates to yield a good solution of the process mining problem. Both approaches are based on linear programming techniques and convex geometry to calculate a certain finite set of regions based on the above characterization of the set of regions by a linear inequation system. In the following section we adjust both approaches to the considered process mining problem and discuss their applicability and their results in this context.

4 Solving the Process Mining Problem

We first introduce three basic principles to identify *redundant* places. Redundant places can be omitted from a marked p/t-net (N, m_0) without changing $L(N, m_0)$. That means, when constructing a net by adding feasible places, we do not add redundant feasible places (since this does not influence (2)).

Definition 6 (Redundant place). *Given a marked p/t-net* (N, m_0), $N = (P, T, F, W)$, *a place* $p \in P$ *is called* redundant *if* $L(N, m_0) = L(P \setminus \{p\}, T, F \cap ((P \setminus \{p\} \times T) \cup (T \times P \setminus \{p\})), W|_{(P\setminus\{p\}\times T)\cup(T\times P\setminus\{p\})}, m_0|_{P\setminus\{p\}})$.

A place p fulfilling $W(p,t) \leq W(t,p)$ for each $t \in T$ and $m_0(p) \geq max\{W(p,t) \mid t \in T\}$ induces no behavioural restriction and is therefore called *useless*. A place p is called *a non-negative linear combination of places* $p_1, \ldots, p_k$ if there are non-negative real numbers $\lambda_1, \ldots, \lambda_k$ ($k \in \mathbb{N}$) such that $m_0(p) = \sum_{i=1}^{k} \lambda_i \cdot m_0(p_i)$, $W(p,t) = \sum_{i=1}^{k} \lambda_i \cdot W(p_i, t)$ and $W(t,p) = \sum_{i=1}^{k} \lambda_i \cdot W(t, p_i)$ for all transitions t. In such a case we shortly write $p = \sum_{i=1}^{k} \lambda_i \cdot p_i$. A place p is called *less restrictive* than a place p' if $\lambda \cdot m_0(p) \geq m_0(p')$ and $\lambda \cdot W(t,p) \geq W(t,p')$ as well as $\lambda \cdot W(p,t) \leq W(p',t)$ for all transitions t and some $\lambda > 0$. In such a case we shortly write $p \preceq p'$.

Lemma 2. *Let* (P, T, F, W, m_0) *be a marked p/t-net and let* $p, p', p_1, \ldots, p_k \in P$ *be pairwise different places. Then there holds:*

(i) p *useless* $\Longrightarrow p$ *is redundant.*
(ii) $p \preceq p' \Longrightarrow p$ *is redundant.*
(iii) $p = \sum_{i=1}^{k} \lambda_i \cdot p_i \Longrightarrow p$ *is redundant.*

Proof. (i) and (ii) are clear by definition, (iii) is proven in [12].

These results can be used to effectively construct a finite net solving the process mining problem. In the following subsections the two mentioned existing basic approaches are introduced, optimized w.r.t. the process mining problem and compared.

4.1 Method 1: Finite Basis of Feasible Places

The first strategy to add a certain finite set of feasible places, used in [12], computes a so called *finite basis* of the set of all feasible places. Such a basis is a finite set of feasible places $P_b = \{p_1, \ldots, p_k\}$, such that each other feasible place p is a non-negative linear combination of $p_1, \ldots, p_k$. The idea is to add only basis places. Adding all basis places leads to a finite representation of the saturated feasible net. By Lemma 2 (iii) the resulting marked p/t-net (N_b, m_b), $N = (P_b, T, F_b, W_b)$, fulfills $L(N_{sat}, m_{sat}) = L(N_b, m_b)$. Consequently, this approach leads to an optimal solution of the process mining problem concerning (2). The following considerations show that such a finite basis always exists and how it can be computed.

As mentioned, the set of feasible places can be defined exactly as the set of non-negative integer solutions of $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}$. The set of non-negative real solutions of such a system is a pointed *polyhedral cone* [19]. According to a theorem of Minkowski [15, 19] polyhedral cones are finitely generated, that means there are finitely many solutions $\mathbf{y}_1, \ldots, \mathbf{y}_k$, called *basis solutions*, such that each element $\mathbf{r}$ of the polyhedral cone is a non-negative linear sum $\mathbf{r} = \sum_{i=1}^{k} \lambda_i \cdot \mathbf{y}_i$ for some $\lambda_1, \ldots, \lambda_k \geqslant 0$. Pointed polyhedral cones have a unique (except for scaling) minimal (w.r.t. set inclusion) set of basis solutions given by the rays of the cone [19]. If all entries of $\mathbf{A}_{L(\sigma)}$ are integers, then also the entries of the basis solutions can be chosen as integers. If $\mathbf{r} = \sum_{i=1}^{k} \lambda_i \cdot \mathbf{y}_i$ for basis solutions $\mathbf{y}_1, \ldots, \mathbf{y}_k$ of $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$, then $p_{\mathbf{r}} = \sum_{i=1}^{k} \lambda_i \cdot p_{\mathbf{y}_i}$. Thus, to compute a finite representation of (N_{sat}, m_{sat}), we compute a finite set of integer basis solutions of $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$. The set of places P_b corresponding to such basis solutions forms a basis of the set of all feasible places. The minimal set of basis solutions $\mathbf{y}_1, \ldots, \mathbf{y}_k$ can be effectively computed from $\mathbf{A}_{L(\sigma)}$ (see for example [16]). The time complexity of the computation essentially depends on the number k of basis solutions which is bounded by $k \leqslant \binom{|L(\sigma)|+2|T|+1}{2|T|+1}$. That means, in the worst case the time complexity is exponential in the number of words of $L(\sigma)$, whereas in most practical examples of polyhedral cones the number of basis solutions is reasonable.

The finite set P_b usually still includes redundant places. The redundant places described in Lemma 2 by (i) and (ii) are deleted from (N_b, m_b) in order to get a preferably small net solving the process mining problem. Algorithm 1 computes (N_b, m_b).

An example for a net calculated by this algorithm is shown for the event log of Example 1. We used the Convex Maple [14] package from [11] to calculate the rays of the inequation system $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}$. The number of basis places corresponding to rays is 55 in this example. Steps 11 to 13 of Algorithm 1 delete 15 of these places.

```
1:  L(σ) ← getProcessLanguage(σ)
2:  A ← EmptyMatrix
3:  (P, T, F, W, m_0) ← (∅, getActivities(σ), ∅, ∅, ∅)
4:  for all w ∈ L(σ) do
5:      A.addRow(a_w)
6:  end for
7:  Solutions ← getIntegerRays(A · r ≥ 0, r ≥ 0)
8:  for all r ∈ Solutions do
9:      P.addCorrespondingPlace(r)
10: end for
11: for all (p, p') ∈ P × P, p ≠ p' do
12:     if p.isUseless() ∨ p ⪯ p' then P.delete(p) end if
13: end for
14: return (P, T, F, W, m_0)
```

Algorithm 1: Computes (N_b, m_b) from an event log σ.

Many of the 40 places of the resulting net (N_b, m_b) are still redundant. It is possible to calculate a minimal subset of places, such that the resulting net has the same behaviour as (N_b, m_b). This would lead to the net shown in Figure 2 with only five places. But this is extremely inefficient. Thus, more efficient heuristic approaches to delete redundant places are of interest. The practical applicability of Algorithm 1 could be drastically improved with such heuristics. In the considered example, most of the redundant places are so called loop places. A loop place is in the pre- and the postset of one transition. If we delete all loop places from (N_b, m_b), there remain the five places shown in Figure 2 plus the eight redundant places shown in Figure 3. In this case this procedure did not change the behaviour of the net (i.e. all loop places were redundant).

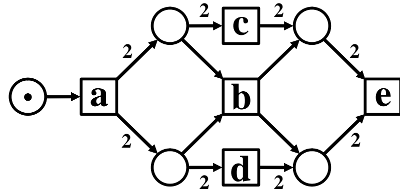

Fig. 2. Key places of the net constructed from the event log of Example 1 with method 1.

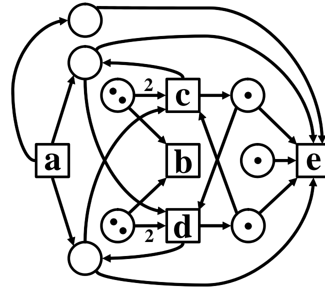

Fig. 3. Further places of the net constructed from the event log of Example 1 with method 1.

In this example the process language is exactly reproduced by the constructed net, i.e. $L(N_b, m_b) = L(\sigma)$. Usually this is not the case. For example omitting the word $acde$ (but not its prefixes) from the process language, the inequation system is not changed, since $adce$ defines the same inequation. Therefore the net constructed from

this changed language with Algorithm 1 coincides with the above example. This net has the additional occurrence sequence $adce$ not belonging to the changed process language. Since the net calculated by Method 1 is the best approximation to the given process language, the changed process language (given by a respective log) has to be completed in this way to be describable through a p/t-net.

The main advantage of method 1 is the optimality w.r.t. (2). The resulting process model may be seen as a natural completion of the given probably incomplete log file. Problematic is that the algorithm in some cases may be inefficient in time and space consumption. Moreover, the resulting net may be relatively big.

4.2 Method 2: Separating Feasible Places

The second strategy, used e.g. in [1, 2], is to add such feasible places to the constructed net, which *separate* specified behaviour from non-specified behaviour. That means for each $w \in L(\sigma)$ and each $t \in T$ such that $wt \notin L(\sigma)$, one searches for a feasible place p_{wt}, which prohibits wt (as shown in the second part of Lemma 1). Such wt is called *wrong continuation* (also called faulty word in [1]) and such places are called *separating feasible places*. If there is such a separating feasible place, it is added to the net. The number of wrong continuations is bounded by $|L(\sigma)| \cdot |T|$. Thus the set P_s containing one separating feasible place for each wrong continuation, for which such place exists, is finite. The resulting net (N_s, m_s), $N_s = (P_s, T, F_s, W_s)$ yields a good solution for the process mining problem: It holds $L(N_{sat}, m_{sat}) = L(\sigma) \Leftrightarrow$ there exists a separating feasible place for each wrong continuation $\Leftrightarrow L(N_s, m_s) = L(\sigma)$. That means, if the process language of the log can exactly be generated by a p/t-net, the constructed net (N_s, m_s) is such a net. Consequently, in this case (2) is optimized. But in general (N_s, m_s) does not necessarily optimize (2), since $L(N_s, m_s) \supsetneq L(N_{sat}, m_{sat}) = L(N_b, m_b)$ is possible (because even if there is no feasible place prohibiting wt, there might be one prohibiting wtt' – but such places are not added). However, in most practical cases $L(N_s, m_s) = L(N_b, m_b)$ is fulfilled (see Subsection 4.3). In situations, where this is not the case, $L(N_s, m_s) \setminus L(N_b, m_b)$ is usually small and thus $L(N_s, m_s) \setminus L(\sigma)$ is small. The following heuristic can be used to further reduce $L(N_s, m_s) \setminus L(\sigma)$ in such situations: If there is no feasible place prohibiting a wrong continuation wt, try to construct a feasible place prohibiting wtt', and if there is no such place, try to construct a feasible place prohibiting $wtt't''$, and so on, until you reach a satisfactory result.

In order to compute a separating feasible place which prohibits a wrong continuation wt, one defines so called *separating regions* defining such places:

Definition 7 (Separating region). *Let* $\mathbf{r}$ *be a region of* $L(\sigma)$ *and let* wt *be a wrong continuation. The region* $\mathbf{r}$ *is a* separating region (w.r.t. wt) *if*

$$(**) \qquad r_0 + \sum_{i=1}^{n} (|w|_{t_i} \cdot r_i - |wt|_{t_i} \cdot r_{n+i}) < 0.$$

Lemma 1 shows that each separating feasible place prohibiting a wrong continuation wt corresponds to a separating region w.r.t. wt and vice versa. A separating region

$\mathbf{r}$ w.r.t. a wrong continuation wt can be calculated (if it exists) as a non-negative integer solution of a homogenous linear inequation system with integer coefficients of the form

$$\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}$$
$$\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}.$$

The vector $\mathbf{b}_{wt} = (b_0, \ldots, b_{2n})$ is defined in such a way that $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0} \Leftrightarrow (**)$. This is achieved by setting

$$b_{wt,i} = \begin{cases} 1 & \text{for } i = 0, \\ |w|_{t_i} & \text{for } i = 1, \ldots, n \\ -|wt|_{t_{i-n}} & \text{for } i = n+1, \ldots, 2n. \end{cases}$$

The matrix $\mathbf{A}_{L(\sigma)}$ is defined as before. For example the inequation $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$ for the wrong continuation abc of the process language of Example 1 reads as follows:

$$r_0 + r_1 + r_2 - r_6 - r_7 - r_8 < 0.$$

If there exists no non-negative integer solution of this system, there exists no separating region w.r.t. wt and thus no separating feasible place prohibiting wt. If there exists a non-negative integer solution of the system, any such solution defines a separating feasible place prohibiting wt. There are several linear programming solver to decide the solvability of such a system and to calculate a solution if it is solvable. The choice of a concrete solver is a parameter of the process mining algorithm, that can be used to improve the results or the runtime. Since the considered system is homogenous, we can apply solvers searching for rational solutions, because each rational solution of the system can be transformed to an integer solution by multiplying with the common denominator. In order to decide if there is a non-negative rational solution and to find such solution in the positive case, the ellipsoid method by Khachiyan [19] can be used. The runtime of this algorithm is polynomial in the size of the inequation system. Since there are at most $|L(\sigma)| \cdot |T|$ wrong continuations, the time complexity for computing (N_s, m_s) is polynomial in the size of the input event log σ. Although the method of Khachiyan yields an algorithm to solve the process mining problem in polynomial time, usually a better choice is the classical Simplex algorithm or variants of the Simplex algorithm [25]. While the Simplex algorithm is exponential in the worst case, probabilistic and experimental results [19] show that the Simplex algorithm has a significant faster average runtime than the algorithm of Khachiyan. The standard procedure to calculate a starting edge with the Simplex algorithm is a natural approach to decide, if there is a non-negative integer solution of the linear inequation system and to find such solution in the positive case. But it makes also sense to use the whole Simplex method including a linear objective function that is optimized (minimized or maximized). The choice of a reasonable objective function for the Simplex solver is a parameter of the algorithm to improve the results. An appropriate example for this is a function minimizing the resulting separating region, i.e. generating minimal arc weights and a minimal initial marking. Moreover, there are several variants of the Simplex algorithm that can improve the runtime of the mining algorithm [25]. For example the inequation systems for the different wrong continuations only differ in the last inequation $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$. This enables the efficient application of incremental Simplex methods.

Independently from the choice of the solver, certain separating feasible places may separate more than one wrong continuation. For not yet considered wrong continuations, that are prohibited by feasible places already added to the constructed net, we do not have to calculate a separating feasible place. Therefore we choose a certain ordering of the wrong continuations. We first add a separating feasible place for the first wrong continuation (if such place exists). Then we only add a separating feasible place for the second wrong continuation, if it is not prohibited by an already added feasible places, and so on. This way we achieve, that in the resulting net (N_s, m_s), various wrong continuations are prohibited by the same separating feasible place. The chosen ordering of the wrong continuations can be used as a parameter to positively adjust the algorithm. In particular, given a fixed solver, there always exists an ordering of the wrong continuations, such that the constructed net has no redundant places.

In general (N_s, m_s) may still include redundant places. There exist no redundant places w.r.t. (i) of Lemma 2, but there may exist redundant places w.r.t. (ii). These places can finally be deleted from (N_s, m_s) in order to get a preferably small net. Algorithm 2 calculates (N_s, m_s).

1: $L(\sigma) \leftarrow getProcessLanguage(\sigma)$
2: $WC \leftarrow getWrongContinuations(L(\sigma))$
3: $A \leftarrow EmptyMatrix$
4: $(P, T, F, W, m_0) \leftarrow (\emptyset, getActivities(\sigma), \emptyset, \emptyset, \emptyset)$
5: **for all** $w \in L(\sigma)$ **do**
6: $\quad A.addRow(a_w)$
7: **end for**
8: **for all** $w \in WC$ **do**
9: $\quad$ **if** $isOccurrenceSequence(w, (P, T, F, W, m_0))$ **then**
10: $\quad\quad r \leftarrow Solver.getIntegerSolution(\mathbf{A} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{r} \geq \mathbf{0}, \mathbf{b}_w \cdot \mathbf{r} < \mathbf{0})$
11: $\quad\quad$ **if** $r \neq null$ **then**
12: $\quad\quad\quad p \leftarrow correspondingPlace(r)$
13: $\quad\quad\quad$ **for all** $p' \in P$ **do**
14: $\quad\quad\quad\quad$ **if** $p' \preceq p$ **then** $P.delete(p')$ **end if**
15: $\quad\quad\quad$ **end for**
16: $\quad\quad\quad P.add(p)$
17: $\quad\quad$ **end if**
18: $\quad$ **end if**
19: **end for**
20: **return** (P, T, F, W, m_0)

Algorithm 2: Computes (N_s, m_s) from an event log σ.

An example for a net calculated by this algorithm is shown for the log of Example 1. We considered the length-plus-lexicographic order of the 45 wrong continuations: b, c, d, e, aa, ae, aba, abc, abd, abe, aca, acb, acc, ace, ada, adb, add, ade, $abba$, $abbb$, $abbc$, $abbd$, $acda$, $acdb$, $acdc$, $acdd$, $adca$, $adcb$, $adcc$, $adcd$, $abbea$, $abbeb$, $abbec$, $abbed$, $abbee$, $acdea$, $acdeb$, $acdec$, $acded$, $acdee$, $adcea$, $adceb$, $adcec$, $adced$, $adcee$. To calculate a separating feasible place for a given wrong continuation, we used the Simplex

method of the Maple Simplex package [14]. We chose an objective function (for the Simplex algorithm) that minimizes all arc weights outgoing from the constructed place as well as the initial marking. Figure 4 shows the places resulting from the first five wrong continuations b, c, d, e and aa. In Figures we annotate the constructed separating feasible places with the wrong continuation, for which the place was calculated. The next wrong continuation ae leads the ae-place in Figure 5. Then aba is already prohibited by the aa-place and thus steps 10 to 17 of Algorithm 2 are skipped for this wrong continuation. The next three wrong continuations abc, abd and abe lead to the respective separating feasible places in Figure 5. All remaining 35 wrong continuations are prohibited by one of the already calculated feasible places. The b-, c-, and d-place from Figure 4 are deleted in Figure 5, because each of these places is less restrictive than either the abc- or the abd-place. Thus, they are deleted as redundant places according to step 13 to 15 of Algorithm 2. Consequently the net in Figure 5 with six places results from Algorithm 2 (only the e-place is still redundant). Altogether the Simplex algorithm was used to calculate nine separating feasible places, of which we deleted three as redundant places.

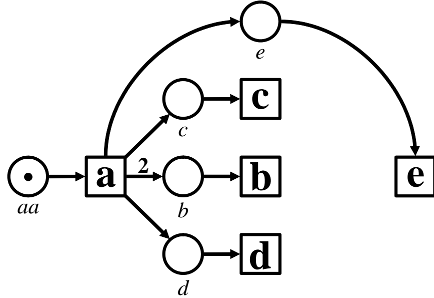

Fig. 4. First five places calculated from the event log of Example 1 with method 2.

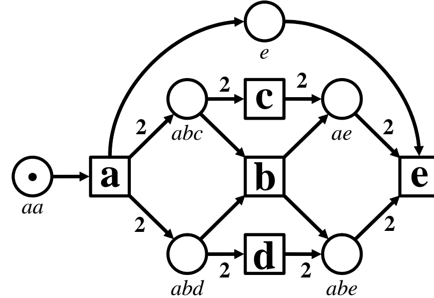

Fig. 5. Final net constructed from the event log of Example 1 with method 2.

The resulting net exactly reproduces the behaviour of the log. Omitting the word $acde$ (but not its prefixes) from the process language, the algorithm calculates also the net from Figure 5 (the inequation systems are not changed). This net does not exactly match the behaviour of this changed process language. But it is still an optimal solution regarding (2) (although this is not guaranteed, since $L(N_b, m_b) \neq L(\sigma)$).

The main advantage of method 2 is that the number of added places is bounded by $|L(\sigma)| \cdot |T|$ and that in most practical cases it is a lot smaller. Usually the resulting net is small and concise. The calculation of the net is efficient. There exists a polynomial time algorithm. Problematic is, that a good solution regarding (2) is not guaranteed, i.e. there may be intricate examples leading to a bad solution of the process mining problem. The next subsection shows an example, where the constructed net is not optimal regarding (2), but this example was really hard to find. Therefore, in most cases the net should be an optimal solution. In the special case $L(N_{sat}, m_{sat}) = L(\sigma)$, the optimality of

method 2 is even guaranteed. Moreover if the constructed net is not optimal, the example of the next subsection indicates that it is usually still a good solution of the process mining problem. Altogether the process model resulting from method 2 is a reasonable completion of the given probably incomplete log file. Lastly it remains to mention that in contrast to method 1, method 2 also computes if the process language of the log is exactly reproduced by the constructed net, since there holds $L(N_s, m_s) = L(\sigma)$ if and only if there is a separating feasible place for every wrong continuation.

4.3 Comparison of Method 1 and Method 2

While $L(N_b, m_b)$ is the smallest net language including $L(\sigma)$ and thus (N_b, m_b) is optimal w.r.t. (2), this must not be the case for (N_s, m_s). On the other hand, the examples and considerations in Subsection 4.1 and 4.2 have shown, that method 2 (calculating (N_s, m_s)) is more efficient than method 1 (calculating (N_b, m_b)) and that method 2 leads to significantly smaller nets. The nets resulting from method 1 usually need some heuristical adaptions to get a manually tractable size.

In the following we consider an example event log leading to a situation, in which method 1 can lead to a better solution than method 2 (dependent on the chosen parameters of method 2) regarding (2).

event log (activity,case):
(a,1) (a,1) (b,2) (b,2) (b,1)
process language:
a aa *aab* b *bb*

Example 2.

Method 1 computes the net (N_b, m_b) in Figure 6. The behaviour of the net (N_b, m_b) completes $L(\sigma)$ by the occurrence sequence ab, i.e. $L(N_b, m_b) \setminus L(\sigma) = \{ab\}$. The net resulting from method 2 is dependent on two parameters: the order of the wrong continuations and the solver calculating the separating feasible places (i.e. which solutions of the inequation systems are computed). We fixed the length-plus-lexicographic order for the wrong continuations. Our standard procedure using the Maple Simplex algorithm calculated the net on the bottom in Figure 7. This net is optimal regarding (2). But choosing another solver could also lead to the net depicted on the top in Figure 7. This net contains another separating feasible place for the wrong continuation $aabb$ (the corresponding region also solves the respective inequation system). It has one additional occurrence sequence abb in contrast to the nets in Figure 6 and Figure 7. Thus it is not optimal regarding (2). Since the region corresponding to this place is not an edge of the polyhedron defined by the inequation system corresponding to the wrong continuation $aabb$ and the Simplex algorithm always computes edges, the Simplex solver computed the edge-solution defining the $aabb$-place on the bottom of Figure 7. Thus, we could not find a solver leading to a non-optimal net regarding (2). But of course we cannot rule out the possibility, that there are log files such that a solver generates such a non-optimal solution. It remains to mention that we searched a long time for the presented example event log, in which it is at least possible that method 2 computes a non-optimal solution.

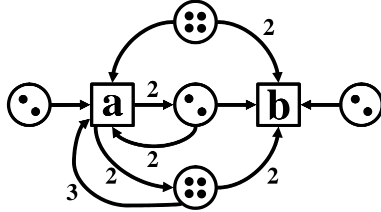

Fig. 6. Net calculated with method 1.

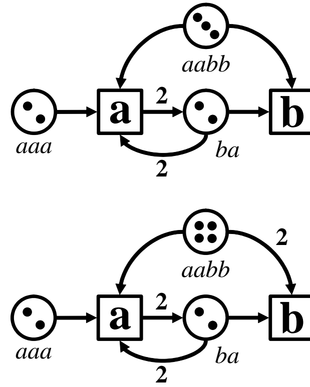

Fig. 7. Two alternative nets calculated with method 2

We showed in this subsection that it is actually possible that method 1 leads to a better solution regarding (2) than method 2. As argued, it is reasonable to assume that this happens only in really rare cases. Method 2 still provides a good solution in these cases. It can be optimized w.r.t. two parameters – the chosen solver and the ordering of the wrong continuations. The distinct advantages of method 2 concerning the runtime and the size of the calculated net altogether argue for method 2. But method 1 can still lead to valuable results, in particular if combined with some heuristics to decrease the number of places of the constructed net. Mainly, algorithms deleting redundant places are of interest.

5 Conclusion

The presented methods were restricted in two ways. Firstly we only considered p/t-nets as process models. Of course there are several other net classes of interest, such as for example workflow nets and elementary nets. Secondly there is one other definition of regions of languages (to define feasible places) in the literature, that could be applied. We also adapted and tested region based synthesis methods w.r.t. such other net classes and region definitions. In this section we will shortly argue that these methods follow the same lines of computing a finite representation of (N_{sat}, m_{sat}) through basis or separating solutions of linear inequation systems. We compare these methods with the presented ones.

First, we discuss alternative Petri net classes. In the example of Subsection 4.1, we proposed to omit loops to simplify the constructed net. Leaving loops from p/t-nets in general, leads to the simpler class of pure nets. The connection between a transition and a place can then be described by one integer number z: If z is positive, the transition produces z tokens in the place, and if z is negative, the transition consumes $-z$ tokens from the place. The process mining approach can be developed for this net class analogously as for p/t-nets. The inequation systems get simpler in this case, in particular the

number of variables is halved. Therefore the process mining approach based on regions of languages gets more efficient for pure nets, but the modelling power is restricted in contrast to p/t-nets.

Typical workflow Petri nets often have unweighted arcs. To construct such nets from a log with the presented methods, one simply has to add additional inequations ensuring arc weights smaller or equal than one to the considered inequation systems. The problem is that the resulting systems are inhomogeneous. Method 1 is not applicable in this case (adaptions of this method are in some cases still possible). Method 2 is still useable, but the linear programming techniques to find separating feasible places have to be adapted. The approaches become less efficient in the inhomogeneous case [19].

A popular net class with unweighted arcs are elementary nets. In elementary nets the number of tokens in a place is bounded by one. This leads to additional inhomogeneous inequations ensuring this property. The process mining methods can in this case be applied as described in the last paragraph. Note that the total number of possible places is finite in the case of elementary nets. Thus also the number of feasible places is finite. This leads to some simplifications concerning a compact representation of the saturated feasible net (similar to method 1), which can be calculated itself.

In [13] regions of partial languages are introduced (in partial languages concurrency between activities can be specified), and in [12] their calculation for the case of a finite partial language is shown. Since the process language of an event log considered in this paper is a special case of a finite partial language, this approach can directly be applied in our setting. Since the set of regions in this case can also be characterized as the set of non-negative integer solutions of a homogeneous inequation system, the two computation methods of Section 4 can analogously be used with this alternative regions definition. But the number of variables as well as the number of inequations is larger than with the regions definition of this paper, in particular the dimension of the resulting cone is bigger. We tested this approach, but the complexity of the algorithm as well as the size of the resulting nets are worse. Nevertheless the approach can be interesting for process mining, if there is some additional information, that can be used to identify independent (concurrent) events in the event log. Extracting such independency information in logs is already applied in [18].

The big advantage of the presented process mining approaches based on regions of languages is that they lead to reliable results. Other process mining algorithms are often more or less heuristic and their applicability is shown only with experimental results. We showed theoretical results that justify that the presented methods lead to a good or even optimal solution regarding (2), while (1) is guaranteed. A problem of the algorithms may be the required time and space consumption as well as the size of the resulting nets. The presented algorithms can be seen as a basis, that can be improved in several directions. Method 2 for computing separating feasible places is flexible w.r.t. the used solver and the chosen ordering of the wrong continuations. Varying the solver could improve time and space consumption, heuristics for fixing an appropriate ordering of the wrong continuations could lead to smaller nets. Moreover, both methods could be improved by additional approaches to find redundant places yielding smaller nets. For example, in this paper we used a simple special objective function in the simplex algorithm to rule out some redundant places. To develop such approaches, experimental

results and thus an implementation of the algorithms is necessary. An implementation of the presented process mining algorithms is the next step to realize a practically usable tool.

References

1. E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In P. D. Mosses; M. Nielsen; M. I. Schwartzbach, editor, *TAPSOFT*, volume 915 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 1995.
2. E. Badouel and P. Darondeau. Theory of regions. In W. Reisig; G. Rozenberg, editor, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1996.
3. J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.
4. 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.
5. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Hardware and petri nets: Application to asynchronous circuit design. In M. Nielsen; D. Simpson, editor, *ICATPN*, volume 1825 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.
6. P. Darondeau. Deriving unbounded petri nets from formal languages. In D. Sangiorgi; R. de Simone, editor, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 1998.
7. A. K. A. de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Workflow mining: Current status and future directions. In R. Meersman, Z. Tari, and D. C. Schmidt, editors, *CoopIS/DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer, 2003.
8. J. Desel and W. Reisig. The synthesis problem of petri nets. *Acta Inf.*, 33(4):297–315, 1996.
9. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. part i: Basic notions and the representation problem. *Acta Inf.*, 27(4):315–342, 1989.
10. A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. part ii: State spaces of concurrent systems. *Acta Inf.*, 27(4):343–368, 1989.
11. M. Franz. Convex - a maple package for convex geometry., 2006. http://www-fourier.ujf-grenoble.fr/ franz/convex/.
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 S. Donatelli and P. S. Thiagarajan, editors, *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2006.
14. Maplesoft. Maple-homepage. http://www.maplesoft.com.
15. H. Minkowski. *Geometrie der Zahlen*. Teubner, 1896.
16. T. Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. PhD thesis, Jerusalem, 1936.
17. R. Parekh and V. Honavar. Automata induction, grammar inference, and language acquisition. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 2000.
18. Process mining group eindhoven technical university: Prom-homepage. http://is.tm.tue.nl/ cgunther/dev/prom/.
19. A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

20. W. van der Aalst, V. Rubin, B. van Dongen, E. Kindler, and C. Guenther. Process mining: A two-step approach using transition systems and regions. Technical Report BPM Center Report BPM-06-30, Department of Technology Management, Eindhoven University of Technology, 2006.
21. W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT Press, Cambridge, Massachsetts, 2002.
22. W. van der Aalst, A. Weijters, H. W. Verbeek, and et al. Process mining: research tools application. http://www.processmining.org.
23. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
24. B. van Dongen, N. Busi, G. Pinna, and W. van der Aalst. An iterative algorithm for applying the theory of regions in process mining. Technical Report Beta rapport 195, Department of Technology Management, Eindhoven University of Technology, 2007.
25. R. J. Vanderbei. *Linear Programming: Foundations and Extensions.* Kluwer Academic Publishers, 1996.