Region-based Algorithms for Process Mining and Synthesis of Petri nets

Josep Carmona, Jordi Cortadella, Mike Kishinevsky

Abstract—The theory of regions was introduced in the early nineties as a method to bridge state-based and event-based models. In this paper, the theory of regions is relaxed and extended to enable the synthesis of Petri nets whose language includes the one of the state-based model. The nets are proved to generate the smallest language with this property. This extension makes the theory applicable in the area of Process Mining. The paper also presents algorithmic solutions to the synthesis of bounded Petri nets. The reported experiments confirm the applicability of the solutions.

Index Terms—Petri nets, transition systems, theory of regions, process mining, bisimulation.

1 Introduction

State-based representations, like FSMs [16], I/O Automata [22], Team Automata [28] or burst-mode automata [26], among others are typical models of complex systems. Such formalizations represent the behavior by means of sequences of events carrying state information. Event-based specifications like Petri nets [27] or CCS [24], model event causality, conflict and concurrency, thus providing alternative information to state-based models often captured in a more concise form.

The theory of regions [17] provides a bridge between state-based and event-based representations. This theory, which is now almost two decades old, was introduced to solve the synthesis problem. This problem consists on transforming a state-based into an event-based specification while preserving the behavior. More specifically, the theory of regions was used for transforming a transition system into a Petri net. Although the theory was initially defined for *elementary transition systems* deriving 1-bounded Petri nets, Mukund [25] extended the theory for k-bounded Petri nets with weighted arcs.

Transforming a transition system into a Petri net is particularly useful when modeling concurrent systems: the state-based model (which represents the concurrency implicitly) can be too complex to understand whereas the equivalent Petri net (which represents the concurrency explicitly) is usually a more concise and clear representation. Moreover, the formal analysis of the model can be highly alleviated if done at the Petri net level. Examples of concurrent systems in the real life range from digital circuits to databases systems. Another useful application of the synthesis problem comes from the *Business Intelligence* domain: business information systems that record transactions (called *event logs*) might *mine* a model from the set of transactions observed in order to realize the processes underlying the system. This is known as *Process Mining* [2]. The available tool support for process mining or synthesis based on the theory of regions is relegated to the academic domain. In the synthesis part, we can mention the tools `petrify` [11] and Synet [7]. For process mining, the Parikh language miner [32] (within the ProM tool) and the ViPTool [5].

In this paper we provide a uniform approach for synthesis or mining of Petri nets from transitions systems, based in the theory of regions. We extend the synthesis theory of [11] (only valid for 1-bounded Petri nets) to k-bounded Petri nets. Second, we show how this extension can be used to mine k-bounded Petri nets from transition systems representing event logs of a business information system. The practicality of our approach is demonstrated by providing efficient algorithms, heuristics and data structures to implement the methods developed in this paper. This paper is an extended version of the papers [9] [8].

1.1 Contributions and comparison with related work

In synthesis, apart from the differences with respect to related work and the contributions reported for the 1-bounded case ([11], Sections 1.3-1.4), there is a high algorithmic emphasis on the extension presented in this paper. This contrasts with some of the approaches presented in the literature for the same goal [17], [25], [15], [4], [19], [20], making the approach presented in this paper more suitable for a practical implementation.

Our approach to Process Mining derives the tightest possible overapproximation of the language defined by the input traces. Other approaches, as was demonstrated in [8] often derive a much looser overapproximation. Other Petri net mining approaches also based on the theory of regions [5], [32] use different strategy from the one presented in this paper: integer linear models are repeatedly solved to find the Petri net places that forbid some of the unseen behavior until some halting criteria is reached. These approaches often derive very loose overapproximations, because sometimes it is not possible to find a place between two transitions that can forbid a particular behavior.

1.2 Two introductory examples

Let us introduce the ideas of this paper by means of two examples, one for each of the approaches presented.

Synthesis of k-bounded Petri nets

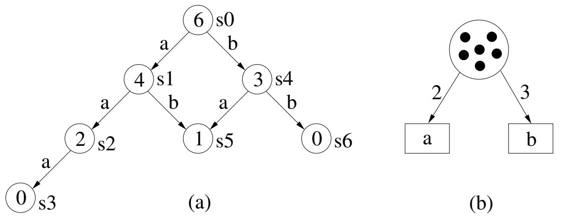


Fig. 1. (a) TS, (b) equivalent 6-bounded Petri net.

Figure 1(a) shows a transition system (TS) with two events a and b (we ask the reader to ignore for the moment the numbers in states). The language defined by the transition system is $(a^3|ab|ba|b^2)$. From this transition system, a 6-bounded Petri net can be derived by the theory developed in this paper, as shown in Figure 1(b). In contrast, if a 1-bounded Petri net is required, more than one transition with the same label is needed to represent the same behavior, leading to a Petri net with 5 transitions and 5 places. The theory in this paper extends the one presented in [11], which was restricted to 1-bounded Petri nets, representing a significant step towards the use of Petri nets in more complex scenarios.

Informally, the theory works as follows: the states of the transition system are collected into regions, which are multisets satisfying conditions with respect to the set of events. Each one of the regions correspond to a place in the derived Petri net. For instance, Figure 1(a) shows the (only one) region r corresponding to the place shown in Figure 1(b). The region is $r(s0) = 6$, $r(s1) = 4$, $\ldots$, $r(s6) = 0$. The crucial idea is that r is a region because each event has a homogeneous enter and exit relation with it: for instance, each time event a occurs, the cardinality of the state reached is decremented by two in r. Once the region is computed, the corresponding place can be created in the Petri net, and the initial marking and input and output arcs for the place can be derived: since $r(s0) = 6$, i.e. the initial state of the transition system has cardinality 6, the initial marking of the place is 6. Moreover, due to the aforementioned relation between event a and r, there is an arc with weight two between the corresponding place and the transition a in Figure 1(b).

Mining of k-bounded Petri nets

For some applications, reproducing exactly the behavior observed is not a requirement, but to have instead a clear visualization of what are the main processes involved is preferred. Process Mining [30] is a clear application of this, but many other applications will appear in the future, given the increasing complexity of software and hardware-based systems. The theory presented in this paper for Petri net Mining is a first step towards bridging the gap between the classical theory of region-based synthesis and real industrial applications. Alternative approaches have been presented in the last years [5], [32] with the same goal.

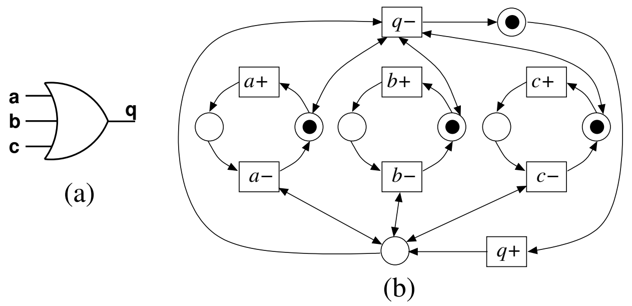


Fig. 2. (a) 3-or gate, (b) Mined Petri net.

Let us use the simple example of Figure 2(a), representing a 3-input OR gate. Although simple, this example illustrates a typical behavior that can not be easily represented in a Petri net: the or-causality between events. The behavior of this gate can be represented with a transition system of 14 states. However, to reproduce exactly this behavior in a Petri net requires complex interactions between transitions. For instance the 1-bounded Petri net representing the behavior has 11 places, 17 transitions and 74 arcs. If a 3-bounded Petri net is synthesized instead, it does not get significantly better: 8 places, 8 transitions and 65 arcs. In contrast to synthesis, Petri net mining can summarize nicely the most important part of the 3-or gate by means of a 1-bounded Petri net, shown in Figure 2(b). The Petri net contains every possible behavior of the gate, including the correct alternation of every rising and falling transitions of the signals in the gate (denoted by $a+$ and $a-$, respectively), and the input-output relation. It also shows additional behavior not observed in the gate: initially the gate might be in an unstable state by changing infinitely its state (i.e. trace $(q+q-)^*$ is possible in the initial marking of the Petri net). In other words, in mining of Petri nets the goal is in finding a Petri net that can reproduce all observed behaviors, but extra behavior are allowed. The intention is to restrict extra behaviors as much as possible and to obtain tight overapproximation of the observed behaviors. In this paper, the theory of k-bounded synthesis is adapted to Process mining, and it is shown that our method derives the tightest possible over-approximation in terms of generated languages.

The organization of the paper is as follows: Section 2 includes the theory needed for the understanding of the paper. Section 3 shows how to derive a k-bounded Petri net from a set of k-bounded regions. Sections 4 and 5 provide two alternatives for deriving a Petri net, depending on the properties to achieve (language-inclusion or bisimilarity). Section 6 defines an algorithm for the generation of k-bounded minimal regions. Section 7 introduces the notion of irredundant cover, to restrict the set of necessary regions. In Section 8 the technique of label splitting is presented for the general case, to force the bisimulation approach of Section 5 to obtain a k-bounded Petri net bisimulating the initial TS. Finally, Section 9 provides experiments of language-inclusion and bisimilarity based methods, some of them taken from real life applications.

2 Basic theory

2.1 Finite transition systems and Petri nets

Definition 2.1 (Transition system): A *transition system* (TS) is a tuple (S, E, A, s_{in}), where S is a set of *states*, E is an alphabet of *actions*, such that $S \cap E = \emptyset$, $A \subseteq S \times E \times S$ is a set of *(labelled) transitions*, and s_{in} is the *initial state*.

Let $\mathsf{TS} = (S, E, A, s_{in})$ be a transition system. We consider connected TSs that satisfy the following axioms:

- S and E are finite sets.
- Every event has an occurrence: $\forall e \in E : \exists (s, e, s') \in A$;
- Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

A TS is called *deterministic* if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. The relation between TSs will be studied in this paper. The *language* of a TS, $L(\mathsf{TS})$, is the set of traces feasible from the initial state. When, $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$, we will denote TS_2 as an over-approximation of TS_1. The notion of simulation between two TSs is related to this concept:

Definition 2.2 (Simulation, Bisimulation [3]): Let $\mathsf{TS}_1 = (S_1, E, A_1, s_{in_1})$ and $\mathsf{TS}_2 = (S_2, E, A_2, s_{in_2})$ be two TSs with the same set of events. A *simulation* of TS_1 by TS_2 is a relation π between S_1 and S_2 such that

- for every $s_1 \in S_1$, there exists $s_2 \in S_2$ such that $s_1 \pi s_2$.
- for every $(s_1, e, s_1') \in A_1$ and for every $s_2 \in S_2$ such that $s_1 \pi s_2$, there exists $(s_2, e, s_2') \in A_2$ such that $s_1' \pi s_2'$.

When TS_1 is simulated by TS_2 with relations π, and viceversa with relation π^{-1}, TS_1 and TS_2 are *bisimilar* [3].

Definition 2.3 (Petri net [27]): A Petri net (PN) is a tuple (P, T, W, M_0) where P and T represent finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.

A transition $t \in T$ is *enabled* in a marking M if $\forall p \in P : M[p] \geq W(p, t)$. Firing an enabled transition t in a marking M leads to the marking M' defined by $M'[p] = M[p] - W(p, t) + W(t, p)$, for $p \in P$, and is denoted by $M \xrightarrow{t} M'$.

The set of all markings reachable from the initial marking M_0 is called its *reachability set*. The *reachability graph* of PN (RG(PN)) is a transition system in which the set of states is the reachability set, the events are the transitions of the net and a transition (M_1, t, M_2) exists if and only if $M_1 \xrightarrow{t} M_2$. We use $L(\mathsf{PN})$ as a shortcut for $L(\mathsf{RG}(\mathsf{PN}))$.

2.2 Multisets

In general, a region is a multiset where additional conditions hold [25]. We recap in this section the main definitions and operations on multisets.

Given a set S, a multiset r of S is a mapping $r : S \longrightarrow \mathbb{N}$. We will also use a set notation for multisets. For example, let $S = \{s_1, s_2, s_3, s_4\}$, then a multiset $r = \{s_1^3, s_2^2, s_3\}$ corresponds to the following mapping $r(s_1) = 3, r(s_2) = 2, r(s_3) = 1, r(s_4) = 0$. The *support* of r (*supp*(r)) is defined as $\{s \in S \mid r(s) > 0\}$. The *power* of r ($r^\diamond$) is $\max_{s \in S} r(s)$. For instance, for $r = \{s_1^3, s_2^2, s_3\}$, *supp*$(r) = \{s_1, s_2, s_3\}$ and $r^\diamond = 3$.

The union, intersection and difference of two multisets r_1 and r_2 are defined as follows:

$$\begin{aligned}(r_1 \cup r_2)(s) &= \max(r_1(s), r_2(s))\\ (r_1 \cap r_2)(s) &= \min(r_1(s), r_2(s))\\ (r_1 - r_2)(s) &= \max(0, r_1(s) - r_2(s))\end{aligned}$$

A multiset r is said to be trivial if $r(s) = r(s')$ for all $s, s' \in S$. The trivial multisets will be denoted by **0**, **1**, …, **K** when $r(s) = 0$, $r(s) = 1$, …, $r(s) = k$, for every $s \in S$, respectively.

Multiset r is *k-bounded* if for all $s \in S : r(s) \leq k$. The multiset r_1 is a *subset* of r_2 ($r_1 \subseteq r_2$) if $\forall s \in S : r_1(s) \leq r_2(s)$. As usual, we will denote by $r_1 \subset r_2$ the fact that $r_1 \subseteq r_2$ and $r_1 \neq r_2$. The *k-topset* of r, denoted by $\top_k(r)$, is defined as follows:

$$\top_k(r)(s) = \begin{cases} r(s) & \text{if } r(s) \geq k \\ 0 & \text{otherwise} \end{cases}$$

and k represents the *degree* of such k-topset. Multiset r_1 is a *topset* of r_2 if there exists some k for which $r_1 = \top_k(r_2)$. For example, the multiset $\{s_1^3, s_3\}$ is a subset of $\{s_1^3, s_2^2, s_3\}$, but it is not a topset. The multisets $\{s_1^3, s_2^2\}$ and $\{s_1^3\}$ are the 2- and 3-topsets of $\{s_1^3, s_2^2, s_3\}$, respectively.

2.3 General regions

Given a multiset r, the *gradient* of a transition (s, e, s') is defined as $\Delta_r(s, e, s') = r(s') - r(s)$. An event e is said to have a non-constant gradient in r if there are two transitions (s_1, e, s_1') and (s_2, e, s_2') such that $r(s_1') - r(s_1) \neq r(s_2') - r(s_2)$.

Definition 2.4 (Region): A multiset r is a *region* if all events have a constant gradient in r.

The original notion of region from [17] was restricted to subsets of S, i.e. events could only have gradients in $\{-1, 0, +1\}$. We say that a region is trivial if it is a trivial multiset (**0**, **1**, …, **K**). The set of non-trivial general regions of TS will be denoted by R_{TS}.

Algorithm 1: BoundedPNDerivation

```
Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$,
       $R$ is a set of regions from $\mathsf{TS}$
Output: Petri net $\mathsf{PN} = (R, E, W, M_0)$
1  begin
2      foreach region $r \in R$ do
3          $M_0[r] = r(s_{in})$
4      end
5      foreach event $e \in E$ do
6          foreach $r \in {}^\circ e$ do
7              $g = \mathsf{MAX\text{-}K\text{-}TOP}(r, e)$
8              $W = W \cup \{(r \xrightarrow{g} e)\}$
9              if $(\Delta_r(e) > -g)$ then
10                 $W = W \cup \{(e \xrightarrow{g+\Delta_r(e)} r)\}$
11             end
12         end
13         foreach $r \in e^\circ$ do $W = W \cup \{(e \xrightarrow{\Delta_r(e)} r)\}$
14     end
15 end
```

Definition 2.5 (Gradient of an event): Given a region r and an event e with $(s, e, s') \in E$, the gradient of e in r is defined as

$$\Delta_r(e) = r(s') - r(s)$$

Definition 2.6 (Minimal region): A region r is minimal if there is no other region $r' \neq \mathbf{0}$ such that $r' \subset r$.

3 Deriving K-bounded Petri nets

This section presents an algorithm that derives a k-bounded Petri net from a set of k-bounded regions. The relation between the initial transition system and the derived Petri net will be established in the next two sections: language-inclusion (Section 4), and bisimilarity (Section 5).

Definition 3.1 (Excitation and switching regions): The excitation region[1] of an event e, $\mathsf{ER}(e)$, is the set of states in which e is enabled, i.e.

$$\mathsf{ER}(e) = \{s \mid \exists s' : (s, e, s') \in E\}$$

The switching region of an event e, $\mathsf{SR}(e)$, is the set of states reachable from $\mathsf{ER}(e)$ after having fired e, i.e.

$$\mathsf{SR}(e) = \{s \mid \exists s' : (s', e, s) \in E\}$$

For convenience, $\mathsf{ER}(e)$ and $\mathsf{SR}(e)$ will be also considered as multisets of states when necessary.

Definition 3.2 (Pre- and post-regions): A region r is a pre-region of e if $\mathsf{ER}(e) \subseteq r$. A region r is a post-region of e if $\mathsf{SR}(e) \subseteq r$. The sets of pre- and post-regions of an event e are denoted by ${}^\circ e$ and e° respectively.

1. Excitation and switching regions are not regions in the terms of Definition 2.4. They correspond to the set of states in which an event is enabled or just fired, correspondingly. The terms are used due to historical reasons.

Definition 3.3 (Topset and its degree): Given and event e, and a pre-region r, the multiset $\mathsf{TOP}(r, e)$ is the multiset q such that $q = \top_g(r)$, $\mathsf{ER}(e) \subseteq \top_g(r)$ and $\mathsf{ER}(e) \not\subseteq \top_{g+1}(r)$. The degree g of $\mathsf{TOP}(r, e)$ is denoted as $\mathsf{MAX\text{-}K\text{-}TOP}(r, e)$.

Informally, $\mathsf{TOP}(e, r)$ is the smallest topset of r still covering $\mathsf{ER}(e)$. Note that this is always a single multiset. The value $\mathsf{MAX\text{-}K\text{-}TOP}(r, e)$ establishes the maximal number of tokens that can be safely removed from the place corresponding to r when e fires guaranteeing that no negative markings are reached. For instance, in the example of Figure 1(a), if we consider the region r shown, $\mathsf{TOP}(a, r) = \top_2(r) = \{s_0^6, s_1^4, s_2^2, s_4^3\}$, and therefore $\mathsf{MAX\text{-}K\text{-}TOP}(r, a) = 2$.

Now we define an important objects called *enabling topsets*, which are crucial for deriving the weighted flow relation in the algorithm.

Definition 3.4 (Enabling topsets): The set of smallest enabling topsets of an event e is denoted by ${}^\star e$ and defined as follows:

$${}^\star e = \{q \mid \exists r \in {}^\circ e \wedge q = \mathsf{TOP}(e, r)\}$$

In Algorithm 1, every region is a place and each event is a transition in the derived PN. The flow relation from places to transitions is defined with respect to the enabling topsets: for a pre-region r of an event e, $g = \mathsf{MAX\text{-}K\text{-}TOP}(r, e)$ tokens can be removed from place r (line 7-8). However the gradient of e in r can be greater than $-g$, i.e. e removes less tokens than g. In this situation, part of the removed tokens ($g + \Delta_r(e)$) are put back in r (lines 9-10). The flow relation from transition e to places is defined for regions covering $\mathsf{SR}(e)$ and the corresponding gradient (line 13).

4 Language-inclusion

4.1 Motivation

In this section we answer the following question: what is the relationship between the PN derived by applying Algorithm 1 to the set of minimal regions of a TS ? As the title of this section suggests, $L(\mathsf{PN}) \subseteq L(\mathsf{TS})$. Hence we assume that the set of minimal regions has been computed. In Section 6 we present an efficient algorithm to compute minimal regions.

Let us show the main message of this section with the help of an example. Figure 3(a) shows a TS. Assume that 1-bounded minimal regions are used in Algorithm 1. These regions are reported in Figure 3(b), and the PN derived from Algorithm 1 on these regions is shown in Figure 3(c). It is easy to see that $L(\mathsf{PN}) \supset L(\mathsf{TS})$: the sequence *cbade* belongs to $L(\mathsf{PN})$ but it does not belong to $L(\mathsf{TS})$.

The remainder of this section tries to formalize construction of Petri Nets based on language overapproximation and presents an important result stating a minimality property on the derived PN.

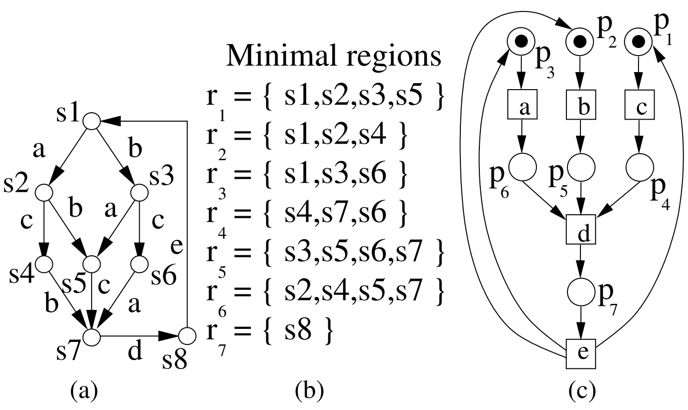


Fig. 3. (a) Initial TS, (b) 1-bounded minimal regions, (c) PN derived by Algorithm 1.

4.2 Mining properties

Formally, given a TS=(S, E, A, s_{in}) , Algorithm 1 derives a k-bounded Petri net PN=(P, T, W, M_0) with the following characteristics:

1) $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$,
2) $T = E$, i.e. there is only one transition per event (no label splitting see Section 8), and
3) *Minimal language containment* (MLC) property:

$$\text{For any k-bounded } \mathsf{PN}' = (P', T', W', M'_0)$$
$$s.t.\ T' = E:\ L(\mathsf{TS}) \subseteq L(\mathsf{PN}') \Rightarrow L(\mathsf{PN}) \subseteq L(\mathsf{PN}')$$

Therefore, the algorithm generates the tightest over-approximation (measured by language containment) among all possible Petri Nets that can be obtained without splitting labels. Since Algorithm 1 clearly guarantees item 2), we will focus on proving two other properties in the next two sections.

4.3 Inclusion

Let us focus on proving the inclusion of language with respect to the initial TS. The following lemma is required for proving the main result of this section:

Lemma 4.1: Let TS=(S, E, A, s_{in}) be a transition system, and PN=(P, T, W, M_0) be the synthesized net (using Algorithm 1) with the set of minimal regions of TS. If trace σ is enabled both in TS and PN leading to state s and marking M respectively, then each minimal region r_i satisfies $r_i(s) = M[p_i]$.

Proof: Induction on $|\sigma|$. For $|\sigma| = 0$, line 3 of the algorithm makes $r_i(s_{in}) = M_0[p_i]$ for every minimal region r_i. Assume the lemma holds for any σ such that $|\sigma| < n$. Let us show it holds for $\sigma = \sigma' e$, with $|\sigma| = n$. By this assumption, the state s' and marking M' reached after firing σ' satisfies $r_i(s') = M'[p_i]$ for every minimal region r_i. We focus on the influence of e in regions in ${}^\circ e$ and e°, because regions not in these two sets are not affected by the firing of e. Now let us consider the firing of e in s' leading to state s: for every region $r_i \in {}^\circ e$, $r_i(s) = r_i(s') + \Delta_{r_i}(e)$. In M' the firing of e results in the following marking M for every minimal region r_i:

$$\begin{aligned} M[p_i] &= M'[p_i] - W(p_i, e) \\ &= M'[p_i] - \mathsf{MAX\text{-}K\text{-}TOP}(r_i, e) \\ &= r_i(s') - \mathsf{MAX\text{-}K\text{-}TOP}(r_i, e) \\ &\geq r_i(s) \end{aligned}$$

and the last inequality holds because lines 9-13 of the algorithm always guarantee $-\mathsf{MAX\text{-}K\text{-}TOP}(r_i, e)\} \geq \Delta_{r_i}(e)$. But the inequality is in fact an equality: if $-\mathsf{MAX\text{-}K\text{-}TOP}(r_i, e)\} > \Delta_{r_i}(e)$, then there exist a transition (s, e, s') satisfying $r_i(s) = g$, $r_i(s') = g + \Delta_{r_i}(e) < 0$, which contradicts the region condition on state s'. Finally, places corresponding to regions $r_i \in e^\circ$ are incremented exactly with $\Delta_{r_i}(e)$ tokens (line 15 of the algorithm), hence the equality also holds for these places. □

Theorem 4.1: Let TS=(S, E, A, s_{in}) be a transition system, and PN=(P, T, W, M_0) be the synthesized net (using Algorithm 1) with the set of minimal regions of TS. Then $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$.

Proof: Induction on the length of the traces in $L(\mathsf{TS})$: in the case $|\sigma| = 1$, we have that if $s_{in} \in \mathsf{ER}(e)$ then $M_0 \xrightarrow{e}$, because otherwise there exists a place $p_i \in {}^\bullet e$ such that $M_0[p_i] < W(p_i, e) = \mathsf{MAX\text{-}K\text{-}TOP}(r_i, e)$, which implies $\mathsf{TOP}(r_i, e)$ does not contain s_{in}, and therefore $s_{in} \notin \mathsf{ER}(e)$, contradiction.

Assume the theorem holds for traces of length less than n, and consider the trace $\sigma = \sigma' e$ of length n. Using the induction hypothesis, let s_m, M be the state reached in TS and the marking reached in PN after firing the trace σ', respectively, and consider the minimal regions $r_1 \ldots r_k$ corresponding to the places $p_1 \ldots p_k \in {}^\bullet e$. Lemma 4.1 guarantees $r_i(s_m) = M[p_i]$, for $i \in 1 \ldots k$ and therefore, if we assume that there exists a $p_j \in {}^\bullet e$ such that $M[p_j] < W(p_j, e)$, then:

$$r_j(s_m) < W(p_j, e) = \mathsf{MAX\text{-}K\text{-}TOP}(r_i, e)$$

which implies that $s_m \notin \mathsf{ER}(e)$, a contradiction. □

4.4 Minimal Language Containment

Given a set $S' \subseteq S$ and a multiset r, $r\mid_{S'}$ represents the *projection* of the multiset r into the set S', i.e. $r\mid_{S'} = r \cap S'$. As the following results show, regions are preserved under language containment or simulation.

Lemma 4.2: Let TS=(S, E, A, s_{in}) and $\mathsf{TS}' = (S', E', A', s_{in})$ be two transition systems such that $S \subseteq S'$, $E \subseteq E'$, $T \subseteq T'$. If $r \in R_{\mathsf{TS}'}$ then $r\mid_S \in R_{\mathsf{TS}}$.

Proof: If the gradient of an event in a region ($\Delta_e(r)$) is constant for transitions in A', then it is also constant for the transitions in A when r is restricted to S, given that $A \subseteq A'$ and $S \subseteq S'$, i.e. by removing arcs, no new violations of the region condition can be created. □

We now prove a similar lemma on the correspondence of regions between simulated TSs.

Lemma 4.3: Let TS=(S, E, A, s_{in}) and $\mathsf{TS}' = (S', E, A', s'_{in})$ be such that there exists a simulation relation of TS by TS' with relation π. If $r \in R_{\mathsf{TS}'}$, then $\pi^{-1}(r) \in R_{\mathsf{TS}}$, and for every event e, $\Delta_e(r)$ is preserved in $\pi^{-1}(r)$.

Proof: The proof for this lemma is similar to the one used in Lemma 4.2, but on simulated states: for every transition $(s, e, s') \in A$ there exists a transition $(\pi(s), e, \pi(s')) \in A'$. Therefore, gradients are preserved in TS for the set $\pi^{-1}(r)$. □

In general, language containment between two TSs does not imply simulation [18]. However, if the TS corresponding to the superset language is deterministic then language containment guarantees the existence of a simulation:

Lemma 4.4: Let $\mathsf{TS}_1 = (S_1, E_1, A_1, s_{in1})$ and $\mathsf{TS}_2 = (S_2, E_2, A_2, s_{in2})$ be two TSs such that TS_2 is deterministic, and $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$. Then TS_2 is a simulation of TS_1.

Proof: The relation $\pi \subseteq (S_1 \times S_2)$ defined as follows:

$$s_1 \pi s_2 \Leftrightarrow \exists\ \sigma : s_{in1} \xrightarrow{\sigma} s_1 \wedge s_{in2} \xrightarrow{\sigma} s_2$$

represents a simulation of TS_1 by TS_2: the first item of Definition 2.2 holds since $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$. If the contrary is assumed, i.e. $\exists s_1 \in S_1 : \nexists s_2 \in S_2 : s_1 \pi s_2$ then the trace leading to s_1 in TS_1 is not feasible in TS_2, which contradicts the assumption. The second item holds because the first item and the determinism of TS_2: for every $s_1 \in S_1$, TS_2 deterministic implies that there is only one state possible $s_2 \in S_2$ such that $s_1 \pi s_2$. But now if e is enabled in s_1 and not enabled in s_2, it will imply that the trace σe, with $s_{in1} \xrightarrow{\sigma} s_1$, is not feasible in TS_2, leading to a contradiction of $L(\mathsf{TS}_1) \subseteq L(\mathsf{TS}_2)$. □

And now we can prove the MLC property (see the definition in Section 4.2) on the derived Petri net from a TS:

Theorem 4.2: Let TS=(S, E, A, s_{in}) be a transition system, and PN=(P, T, W, M_0) be the synthesized net (using Algorithm 1) with the set of minimal regions of TS. Then PN satisfies the MLC property.

Proof: By contradiction. Let $\mathsf{PN}' = (P', T', W', M_0')$ exist with the reachability graph corresponding to the $\mathsf{TS}' = (S', E', A', s_{in}')$ such that $E' = T$, $L(\mathsf{TS}) \subseteq L(\mathsf{TS}')$ and $L(\mathsf{PN}) \not\subseteq L(\mathsf{TS}')$. The following facts can be observed:

- TS' and $RG(\mathsf{PN})$ are deterministic because $E = E' = T$ and therefore they correspond to the reachability graph of Petri nets with a different label for each transition.
- Since TS' is deterministic and $L(\mathsf{TS}) \subseteq L(\mathsf{TS}')$, then there is a simulation π of TS by TS' (Lemma 4.4).
- $\forall r' \in R_{\mathsf{TS}'}$, $r = \pi^{-1}(r) \in R_{\mathsf{TS}}$, and events with constant gradient are the same in r' and r (Lemma 4.3).
- Each non-minimal region can be described as the union of disjoint minimal regions [11], and therefore we can focus only on minimal regions.
- Each minimal region $r \in R_{\mathsf{TS}}$ is a region in $R_{RG(\mathsf{PN})}$, since PN has been obtained with Algorithm 1 from the set of minimal regions in TS. Moreover, since $RG(\mathsf{PN})$ is deterministic and $L(\mathsf{TS}) \subseteq L(\mathsf{PN})$ (Theorem 4.1), then there is a simulation of TS by $RG(\mathsf{PN})$ (Lemma 4.4). Now using Lemma 4.3, together with the fact that r is a region both in R_{TS} and $R_{RG(\mathsf{PN})}$, events with constant gradient in TS have also constant gradient in $RG(\mathsf{PN})$.

Hence, the previous items show that for a region in TS' there is a corresponding region in $RG(\mathsf{PN})$ with the same gradient on the events. In Petri net terms, this fact means that the flow relation of PN' is included in the flow relation of PN. Additionally, the simulations connecting both transition systems preserve the initial states (see Lemma 4.4). This contradicts the assumption that $L(\mathsf{PN}) \not\subseteq L(\mathsf{TS}')$. □

Section 9 applies the results presented in this section in the area of *Process Mining*.

5 Bisimilarity

For some applications, deriving a Petri net that represents the behavior of the initial transition system exactly is required. However, as the example of Section 4.1 demonstrates, this can not be accomplished in general. In this section we tackle this problem by defining a property (*excitation closure*) that a TS must satisfy in order to derive a PN with bisimilar behavior. Moreover, a label splitting technique is presented in Section 8, to repair excitation closure violations in a TS. This allows to present a complete method for deriving bisimilar k-bounded PNs, described in Section 8.3. Finally, in Section 7 excitation closure is used to reduce the number of necessary regions while preserving the properties in the derived net.

5.1 Excitation closure

The concept of excitation closure was presented in [11] for 1-bounded regions. Here we generalize it for arbitrary k-bounded regions:

Definition 5.1 (k-ECTS): A TS is *k-excitation closed* (k-ECTS) if using minimal k-bounded regions, it satisfies the following two properties:

1) Excitation closure. For each event e

$$\bigcap_{q \in {}^{\star}e} \mathit{supp}(q) = \mathsf{ER}(e)$$

2) Event effectiveness. For each event e, ${}^{\circ}e \neq \emptyset$

For instance, the TS shown in Figure 3(a) is not 1-ECTS, since event c is not excitation closed for the set of regions shown in Figure 3(b):

$$\bigcap_{q \in {}^{\star}c} \mathit{supp}(q) = r_1 \supset \mathsf{ER}(c)$$

while the rest of events (a, b, d and e) are excitation closed. However, the TS is 2-ECTS, as the Figure 4 demonstrates. We will say that a TS is ECTS if it is k-ECTS for some k.

Minimal regions

$r_0 = \{s2,s3,s5^2,s7\}$

$r_1 = \{ s1,s2,s3,s5 \}$

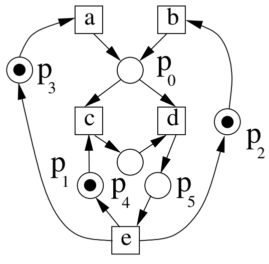
$r_2 = \{ s1,s2,s4 \}$

$r_3 = \{ s1,s3,s6 \}$

$r_4 = \{ s4,s7,s6 \}$

$r_5 = \{ s8 \}$

(a)



(b)

Fig. 4. (a) 2-bounded minimal regions in the TS of Figure 3(a), where only region r_0 is not 1-bounded, (b) Bisimilar PN derived by Algorithm 1.

The intuition behind the notion of excitation closure is that, provided that Algorithm 1 uses the enabling topsets to decide when a transition is enabled, the set of states corresponding to the enabling of the transition must be exactly the set of states where the transition is actually enabled in the TS (ER). In other words, if one state in the intersection of enabling topsets does not belong to the ER of the event, then the RG(PN) is an over-approximation of the TS and hence is not able to simulate the TS.

Theorem 5.1: Let TS=(S, E, A, s_{in}) be a k-excitation closed transition system. Algorithm 1 on the set of minimal regions derives a PN=(P, T, W, M_0) with reachability graph bisimilar to TS.

Proof: Theorem 4.1 provides $L(\text{TS}) \subseteq L(\text{RG(PN)})$, and together with the fact that RG(PN) is deterministic (since $T = E$), Lemma 4.4 can be applied to prove the existence of a simulation of TS by RG(PN). The simulation is

$$s_1 \pi s_2 \Leftrightarrow \exists\ \sigma : s_{in1} \xrightarrow{\sigma} s_1 \wedge s_{in2} \xrightarrow{\sigma} s_2$$

We will prove that π^{-1} is a simulation of RG(PN) by TS (see Definition 2.2). For that purpose, assume that there exists a reachable state s_2 in RG(PN) such that no state in TS is related to. We can prove by induction on the length of traces leading to the state that this never happens. In the base case ($n = 0$), this trivially holds because the initial states are related by the relation π^{-1}. In the induction step, assume that there exists a trace $\sigma = \sigma' e$, leading to a state s_2 in RG(PN) but no state in S is related to s_2 by π^{-1}. Now consider the state s_1, with corresponding marking M in PN, reached in RG(PN) after firing σ'. By the induction hypothesis, $\pi^{-1}(s_1) \in S$ is related to s_1, and according to our assumption, $\pi^{-1}(s_1) \notin \text{ER}(e)$ (otherwise π will relate the state reached after firing e at $\pi^{-1}(s_1)$ with s_2). But then there exists a region $r_i \in {}^{\star}e$ for which $\pi^{-1}(s_1) \notin q$, with $q = \text{TOP}(r_i, e)$. Due to the excitation closure, the corresponding place p_i derived by Algorithm 1 satisfies $M[p_i] < W(p_i, e)$ due to Lemma 4.1, contradicting the assumption that e is enabled at s_1. The other condition for proving the simulation can be also deduced from the reasoning above. □

6 Computation of general regions

The first step in the approaches presented in the two previous sections is to compute the set of regions from the TS. The fact that the number of k-bounded regions in a given TS might be exponential with respect to the number of states makes this step a challenging one. In this section, we will try to convince the reader that practical algorithms can be used to fight the complexity underlying the generation of regions.

Independently on the desired properties in the derived Petri net, not all the regions are necessary. Regions that are

- Non-minimal [15], or
- Not a pre-region or post-region (see Property 6.4), or
- Trivial, i.e. **0**, **1**, ..., **K**

are not required. Moreover, as Section 7 reports, depending on the properties in the derived Petri net some minimal regions can also be excluded:

- In language-inclusion based methods, only a *subset irredundant minimal cover* of regions is needed.
- In bisimilarity based methods, only an *irredundant minimal cover* of regions is needed.

6.1 Basic algorithm

In this section we will present an algorithm to generate a set of regions sufficient for deriving a correct Petri net, independently of the approach followed (language-inclusion or bisimilarity). Excitation/switching regions will be the sets of states that serve as seeds in the algorithm. The following properties will be useful to ensure the generation of minimal pre/post-regions [9]:

Property 6.1: Let TS $= (S, \Sigma, E, s_{in})$ be a transition system without deadlock states. Then, for any region r there is an event e for which $\Delta_r(e) \leq 0$.

Property 6.2: Let TS $= (S, \Sigma, E, s_{in})$ be a transition system in which there is a transition $(s, e', s_{in}) \in E$. Then, for any region r there is an event e for which $\Delta_r(e) \geq 0$.

Property 6.3: Let TS $= (S, \Sigma, E, s_{in})$ be a transition system without deadlock states. Then, any region $r \neq \mathbf{0}$ is the pre-region of some event e.

Property 6.4: Let TS $= (S, \Sigma, E, s_{in})$ be a transition system. Then, any region $r \neq \mathbf{0}$ is the pre-region or the post-region of some event e.

Due to Property 6.4, only supersets of the ER and SR of every event must be considered. When a given (multi)set contains some region violation, it is expanded to avoid it. Formally, given a multiset r and an event e with non-constant gradient, the following definitions characterize the set of regions that include r.

Definition 6.1: Let $r \neq \mathbf{0}$ be a multiset. We define

$$\begin{aligned}\mathcal{R}_g(r, e) &= \{r' \supseteq r | r' \text{ is a region and } \Delta_{r'}(e) \leq g\} \\ \mathcal{R}^g(r, e) &= \{r' \supseteq r | r' \text{ is a region and } \Delta_{r'}(e) \geq g\}\end{aligned}$$

$\mathcal{R}_g(r, e)$ is the set of all regions larger than r in which the gradient of e is smaller than or equal to g. Similar for

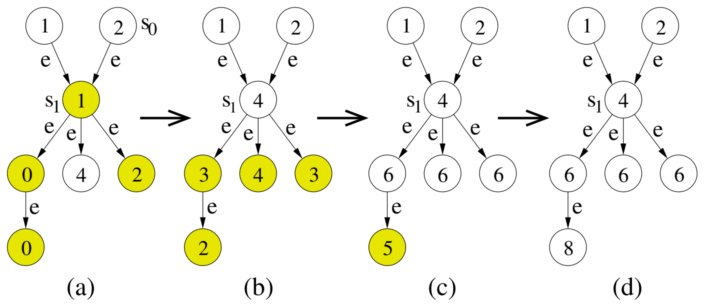


Fig. 5. Successive calculations of $\sqcap^2(r,e)$. States violating the constraint $\Delta_r(e) \geq 2$ are depicted with grey background.

$\mathcal{R}^g(r,e)$ and the gradient of e greater than or equal to g. Notice that in this definition a gradient g may be used to partition the set of regions including r into two classes. This binary partition is the basis for the calculation of minimal k-bounded regions that will be presented at the end of this section.

The expansion of a set consists of increasing the arity of some states to satisfy the region condition. As it is proved below, there is a bound in the increase for each state:

Definition 6.2: Given a multiset r, a state s and an event e, the following δ functions are defined[2]:

$$\begin{aligned}\delta_g(r,e,s) &= \max(0, \max_{(s,e,s')\in E}(r(s')-r(s)-g))\\ \delta^g(r,e,s) &= \max(0, \max_{(s',e,s)\in E}(r(s')-r(s)+g))\end{aligned}$$

Informally, δ_g denotes a lower bound for the increase of $r(s)$, taking into account the arcs leaving from s, to force $\Delta_{r'}(e) \leq g$ in some region r' larger than r. Similarly, δ^g denotes a lower bound taking into account the arcs arriving at s, to force $\Delta_{r'}(e) \geq g$.

Let us use Figure 5(a) to illustrate this concept. In the figure each state is labeled with $r(s)$. Now imagine that we want to force the gradient on event e to be greater or equal to 2. For state s_1, we have $\delta^2(r,e,s_1) = 3$, which is determined from the arc $s_0 \xrightarrow{e} s_1$, indicating that $r'(s_1) \geq 4$ in case we seek a region $r' \supset r$ with $\Delta_{r'}(e) \geq 2$. The states that, as s_1, violate this constraint are shown with grey background.

Definition 6.3: Given a multiset r and an event e, the multisets $\sqcap_g(r,e)$ and $\sqcap^g(r,e)$ are defined as follows:

$$\begin{aligned}\sqcap_g(r,e)(s) &= r(s)+\delta_g(r,e,s)\\ \sqcap^g(r,e)(s) &= r(s)+\delta^g(r,e,s)\end{aligned}$$

Intuitively, $\sqcap_g(r,e)$ is a move towards growing r and obtaining all regions r' with $\Delta_r(e) \leq g$. Similarly, $\sqcap^g(r,e)$ for those regions with $\Delta_r(e) \geq g$. It is easy to see that $\sqcap_g(r,e)$ and $\sqcap^g(r,e)$ always derive multisets larger than r. The successive steps of calculating $\sqcap^2(r,e)$ are illustrated in Figures 5(a)-(d).

2. For convenience, we consider $\max_{x\in D} P(x) = 0$ when the domain D is empty.

Algorithm 2: GenerateMinimalRegions

```
Input: Transition system TS = $(S, E, A, s_{in})$,
       bound $k$
Output: $R$ is the set of $k$-bounded minimal regions
        from TS
1  begin
2      $R = \emptyset$
3      $P = \{\mathsf{ER}(e) \mid e \in E\} \cup \{\mathsf{SR}(e) \mid e \in E\}$
4      while $P \neq \emptyset$ do
5          $r$ = remove_one_element ($P$)
6          if $r \notin R$ then
7              $e$ = event_with_non_constant_gradient ($r$)
8              $(g_{min}, g_{max}) = (\min \Delta_r(e)), \max \Delta_r(e))$
9              $g = \lfloor (g_{min} + g_{max})/2 \rfloor$
10             $r_1 = \sqcap_g(r,e)$
11             if $((r_1^{\diamond} \leq k) \wedge (\mathbf{1} \not\subset r_1))$ then $P = P \cup \{r_1\}$
12             $r_2 = \sqcap^{g+1}(r,e)$
13             if $((r_2^{\diamond} \leq k) \wedge (\mathbf{1} \not\subset r_2))$ then $P = P \cup \{r_2\}$
14         end
15     end
16     $R = R \setminus \{r \mid r$ is not a region$\}$
17     $R = R \setminus \{r \mid \exists r' \in R : r' \subset r\}$
18 end
```

Theorem 6.1 (Expansion on events): (a) Let $r \neq \mathbf{0}$ be a multiset and e an event such that there exists some (s,e,s') with $r(s') - r(s) > g$. The following hold:

1) $r \subset \sqcap_g(r,e)$
2) $\mathcal{R}_g(r,e) = \mathcal{R}_g(\sqcap_g(r,e),e)$

(b) Let $r \neq \mathbf{0}$ be a multiset and e an event such that there exists some (s,e,s') with $r(s') - r(s) < g$. The following hold:

1) $r \subset \sqcap^g(r,e)$
2) $\mathcal{R}^g(r,e) = \mathcal{R}^g(\sqcap^g(r,e),e)$

Proof: (We prove item (a), item (b) is similar.)
(a.1) Given the event e, for all states s, s' with (s,e,s') either (i) $r(s') - r(s) \leq g$ or (ii) $r(s') - r(s) > g$. In case (i), the following equality holds: $r(s) = \sqcap_g(r,e)(s)$ because $\delta_g(r,e,s) = 0$ by Definition 6.2. In case (ii), $\delta_g(r,e,s) > 0$, and therefore $r(s) < \sqcap_g(r,e)(s)$. Given that the rest of states without outgoing arcs labeled e fulfill also $r(s) = \sqcap_g(r,e)(s)$, and because there exists at least one transition satisfying (ii), the claim (a.1) of the theorem holds.
(a.2) To obtain $\mathcal{R}_g(r,e)$ it is necessary to guarantee $\Delta_{r'}(e) \leq g$ for each region $r' \supseteq r$. Given (s,e,s') with $r(s') - r(s) > g$, two possibilities can induce a gradient lower than g, decreasing $r(s')$ or increasing $r(s)$, but only the latter leads to a multiset with r as a subset. □

The calculation of all minimal k-bounded regions is presented in Algorithm 2. It is based on a dynamic programming approach that, starting from a multiset, generates an exploration tree in which an event with non-constant gradient is chosen at each node (line 7). All possible gradients for that event are explored by means

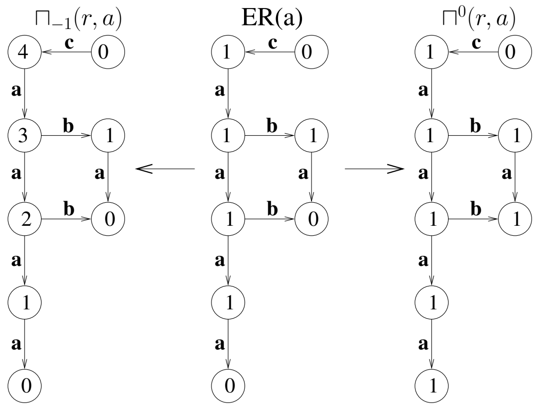


Fig. 6. Exploration of regions in Algorithm 2 for $\mathsf{ER}(a)$: $\min \Delta_r(a) = -1$, $\max \Delta_r(a) = 0$.

of a binary search (lines 8-13). Dynamic programming with memoization [10] avoids the exploration of multiple instances of the same node (line 6). The final step of the algorithm (lines 16-17) removes all those multisets that are neither regions nor minimal regions that have been generated during the exploration.

Figure 6 shows how Algorithm 2 will explore the $\mathsf{ER}(a)$ (center). After successive calculations of $r_1 = \sqcap_{-1}(\mathsf{ER}(a), a)$, it reaches the region shown on the left. Analogously, the region on the right is obtained after successive calculations of $r_2 = \sqcap^0(\mathsf{ER}(a), a)$, and notice that provided that cover both $\mathsf{ER}(a)$ and $\mathsf{SR}(a)$, it corresponds to a self-loop place for transition a.

Theorem 6.2: Algorithm GenerateMinimalRegions calculates all k-bounded minimal regions.

Proof: The proof is based on the following facts:

1) All minimal regions are a pre- or a post-region of some event (Property 6.4). Any pre- (post-) region of an event is larger than its ER (SR). Line 3 of the algorithm puts all seeds for exploration in P. These seeds are the ERs and SRs of all events.
2) Each r that is not a region is enlarged by $\sqcap_g(r,e)$ and $\sqcap^{g+1}(r,e)$ for some event e with non-constant gradient. Given that $g = \lfloor (g_{min} + g_{max})/2 \rfloor$, there is always some transition $s_1 \overset{e}{\rightarrow} s_2$ such that $r(s_2) - r(s_1) = g_{max} > g$ and some transition $s_3 \overset{e}{\rightarrow} s_4$ such that $r(s_4) - r(s_3) = g_{min} < g + 1$. Therefore, the conditions for theorem 6.1 hold. By exploring $\sqcap_g(r,e)$ and $\sqcap^{g+1}(r,e)$, no minimal regions are missed.
3) The algorithm halts since the set of k-bounded multisets with $\subseteq$ is a lattice and the multisets derived at each level of the tree are larger than their predecessors. Thus, the exploration will halt at those nodes in which the power of the multiset is larger than k (lines 11-13). The condition ($\mathbf{1} \not\subset r'$) in lines 11-13 improves the efficiency of the search, since regions containing $\mathbf{1}$ are not minimal.

□

6.2 Computing an upper bound for k

The algorithm presented in Section 6.1 assumes an input parameter k determining the maximal bound required for the derivation of a PN. If k is unknown, an upper bound can be computed from the transition system. In [13] (Section IV), a method to compute the bounded Petri net closure of a regular language is presented. This method consists of:

1) Unbounded Petri net synthesis of the language [12], by solving a finite system of linear constraints over the integers,
2) Construction of the covering tree [21] related to the unbounded Petri net derived in step 1): the vertices of this tree are tuples $v = (M, q)$, where M is a marking of the Petri net constructed in the previous step, and q is a state of the transition system,
3) Iterate 1) and 2) until each leaf vertex $v = (M, q)$ of the corresponding covering tree is identical to some ancestor vertex $v' = (M', q')$, i.e., $M' = M$ and $q' = q$. When iteration takes place ($M' < M$), new constraints are added to the linear problem solved in step 1). These new constraints account for the boundedness of the new net derived, for the case of the repetitive sequence between M' and M.

If step 1) is done with language T^*, the final net obtained provides an upper bound for k [14].

Although this strategy to provide an upper bound has high complexity, it might be used when there is no knowledge on the maximal cardinality needed for the regions. For many practical purposes it is also possible to progressively increase the size of k during synthesis iterations. Note that this problem was not considered in previous work [11], [9].

6.3 Implementation details

All the operations required in Algorithm 2 to manipulate the multisets can be efficiently implemented by using a symbolic representation. A multiset can be modeled as a vector of Boolean functions, where the function at position i describes the characteristic function of the set of states having cardinality i. Hence, multiset operations (union, intersection and complement) can be performed as logic operations (disjunction, conjunction and complement) on Boolean functions. In practice, an array of Binary Decision Diagrams (BDDs) [6] can be used to represent implicitly a multiset.

7 IRREDUNDANT COVERS

In the methods presented in previous sections, all the minimal regions were used to satisfy the properties (bisimilarity and language-inclusion, respectively). In this section we show that, as in the case of 1-bounded minimal regions [11], the number of minimal general regions needed might be smaller than the total number of minimal regions. Regions that are not needed

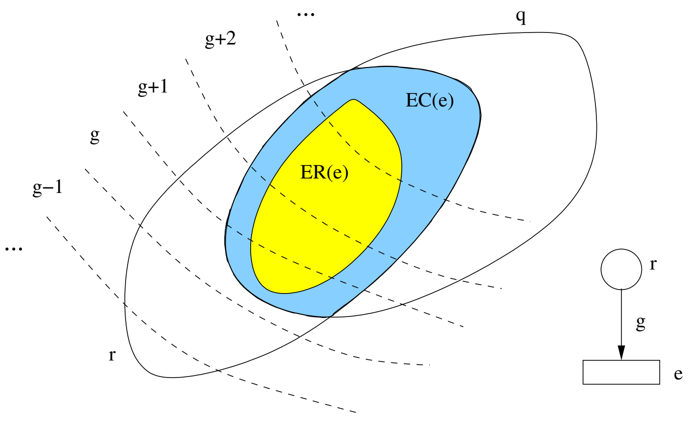


Fig. 7. Situation where the arc $r \xrightarrow{g} e$ can be converted into $p \xrightarrow{g-1} e$.

for deriving a correct PN (under language-inclusion or bisimilarity) are called *redundant*.

We start by defining an important set of states related to an event:

Definition 7.1 (Enabling Closure): Given an event e, and a set of regions R the enabling closure of e with respect to R is defined as:

$$EC(e) = \bigcap_{q \in\ ({}^{\star}e \cap R)} supp(q)$$

The enabling closure of an event e represents the set of states that enable e if the set of pre-regions is only considered for a given set of regions R. The intuition behind the theory of the remainder of this section is the following: given a set of regions R, if a region r does not reduces the enabling closure of any event, then its removal does not modify the language of the Petri net derived by Algorithm 1. In other words, the language of the Petri net derived by Algorithm 1 from R and from $R - \{r\}$ is the same. To prove that a given region is not needed by Algorithm 1, a stepwise simplification of the arcs arising in the place is applied. Figure 7 illustrates the idea: region $r \in\ {}^{\circ}e$ is shown together with the corresponding partition based on the cardinality of its states (only the partition for states corresponding to the arities $g-1, \ldots, g+2$ is shown). Accordingly, the enabling topset derived from region r (see Definition 3.4) will be $\top_g(r)$. However, if $\top_{g-1}(r)$ is considered instead as enabling topset, $EC(e)$ is still preserved due to the presence of region q. As a consequence, the arc derived by Algorithm 1 will have weight $g-1$ instead of g. In general, this methodology can lead to arcs with zero weight, which can be removed. When all arcs arising in a place have been removed, then the place can be safely removed from the Petri net without modifying its language. The rest of the section formalizes this idea.

Theorem 7.1: Given a TS $= (S, E, A, s_{in})$ and R a set of regions, let e be an event and a region $r \in\ {}^{\circ}e \cap R$ such that $\mathsf{ER}(e) \subseteq \top_g(r)$. If the following equality holds

$$EC(e) = \left(\bigcap_{q \in ({}^{\star}e - \{r\})} supp(q) \right) \cap \top_{g-1}(r)$$

then the arcs $p \xrightarrow{g} e$ and $e \xrightarrow{g+\Delta_r(e)} p$ in the PN derived by Algorithm 1 can be substituted by the arcs $p \xrightarrow{g-1} e$ and $e \xrightarrow{g+\Delta_r(e)-1} p$ respectively as long as $g + \Delta_r(e) - 1 \geq 0$, leading to PN$'$. Moreover, $L(\mathsf{PN}) = L(\mathsf{PN}')$.

Proof: $L(\mathsf{PN}) \subseteq L(\mathsf{PN}')$ holds because PN$'$ is PN with one constraint less on the arc connecting p and event e. $L(\mathsf{PN}) \supseteq L(\mathsf{PN}')$ can be proven by induction on the length of traces. Case $|\sigma| = 0$ holds. In the induction step, let trace σe be enabled in PN$'$. By the induction hypothesis, σ is enabled in PN, leading to state s. And now one can observe that if e is not enabled in s, then there is a region $r_i \in\ {}^{\circ}e$ for which the state $s \notin argmax_q\{q = \top_g(r_i)\}$. But provided that the enabling closure is preserved, event e can not be enabled in PN$'$ after σ, because state s is still left out from the enabling closure in PN$'$ by the topset of r_i, contradiction. □

Corollary 7.1: Given a place p, if successive applications of Theorem 7.1 are applied making every arc $p \xrightarrow{0} e$, then p is redundant.

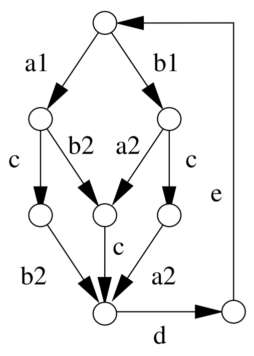
In the results presented above, a set of regions R is used for which the set ${}^{\circ}e$ is defined for every event e, i.e. ${}^{\circ}e \subseteq R$. Therefore the notion of redundancy presented in Corollary 7.1 is done with respect to a given set of regions. In general the notion of redundancy is not monotonic, i.e. if places p_1 and p_2 are redundant with respect to the set of regions R, it does not necessary imply that p_2 is redundant with respect to $R - \{r_1\}$, or viceversa.

Hence redundancies can arise in the approaches described in the previous sections. We can define irredundant minimal covers for each one of these approaches:

- In the language inclusion approach (Section 4): the cover will contain only these regions whose removal modify the enabling closure of some event.
- In the bisimilarity approach (Section 5): regions that are not necessary for the excitation closure of any event will not appear in the cover.

8 Label splitting

In some applications the maximal bound accepted in the derived PN can not be exceeded. If a TS is not k-ECTS and the bound can not be incremented further, then only PNs over-approximating the initial TS can be derived with the method presented so far (see Section 4). This section presents a simple yet practical technique to escape from this problem. Let us illustrate the technique with the running example shown in Figure 3(a). Assume that the maximal bound accepted is 1, and a PN with bisimilar behavior must be generated. As reported in Figures 3(b)-(c), the derived PN on the set of minimal



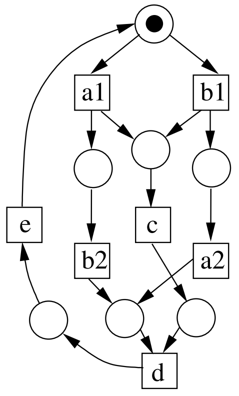
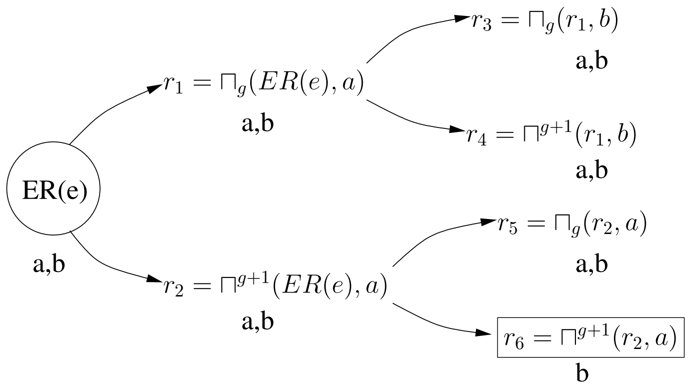


Fig. 8. (a) TS after application of label splitting on events a and b, (b) PN derived by Algorithm 1 on the set of minimal regions from TS (a).



ER(e)
a,b
$r_1 = \sqcap_g(ER(e), a)$
a,b
$r_2 = \sqcap^{g+1}(ER(e), a)$
a,b
$r_3 = \sqcap_g(r_1, b)$
a,b
$r_4 = \sqcap^{g+1}(r_1, b)$
a,b
$r_5 = \sqcap_g(r_2, a)$
a,b
$r_6 = \sqcap^{g+1}(r_2, a)$
b

Fig. 10. Multisets reached while exploring the ER(e). For instance, multisets r_1 and r_2 are obtained by an iteration of Algorithm 2 (lines 10,12) on event with non-constant gradient a.

1-bounded regions accepts more traces than the initial TS. Now let us *split* events a and b in the TS as shown in Figure 8(a). The new events appearing, $a1$, $a2$, $b1$ and $b2$ are treated as different events and will generate a transition each in Algorithm 1. The new TS is 1-ECTS with corresponding PN shown in Figure 8(b). The fact that transitions keep the label of the original events guarantees that the reachability graph of the PN in Figure 8(b) is bisimilar to TS of Figure 3(a).

The formal definition of the technique is presented:

Definition 8.1 (Label splitting): Let TS $= (S, E, A, s_{in})$ be a transition system. The splitting of event $e \in E$ derives a transition system TS$' = (S, E', A', s_{in})$, with $E' = E - \{e\} \cup \{e_1, \dots, e_n\}$, and such that every transition $(s_1, e, s_2) \in A$ corresponds to exactly one transition (s_1, e_i, s_2), and the rest of transitions for events different from e are preserved in A'.

This section presents heuristics to guide the label splitting technique for the achievement of the excitation closure. The following theorem guarantees completeness of the method:

Theorem 8.1: Let TS be a non k-ECTS. There exists a finite sequence of label splitting transformations that derive a k-ECTS TS$'$ bisimilar to TS.

Proof: It is always possible to apply label splitting until every arc of the TS is labelled as a unique event. In this extreme case, the TS is 1-ECTS, and therefore it is also k-ECTS, for $k > 1$. □

Theorem 8.1 provides only an upper bound on the number of splittings to be done to achieve excitation closure. In the remainder of this section we provide heuristics to choose, among all the possible splittings, those that may be more likely to repair the excitation closure violations.

8.1 Splitting disconnected ERs

The following property can be used to present the first heuristic for label splitting:

Property 8.1: Regions are preserved by the label splitting technique.

Figure 9 illustrates an example in which $EC(e) \neq$ ER(e). ER(e) at the top line of the Figure has several disconnected components ER$_i(e)$. In the figure, the white part in a ER$_i(e)$ represents those states in $EC(e) -$ ER$_i(e)$. ER$_2(e)$ is covered precisely by teh corresponding subset of $EC(e)$. By splitting event e into two events e_1 and e_2 in such a way that e_2 corresponds to ERs with no extra states in $EC(e)$, we can ensure at least the excitation closure for e_2.

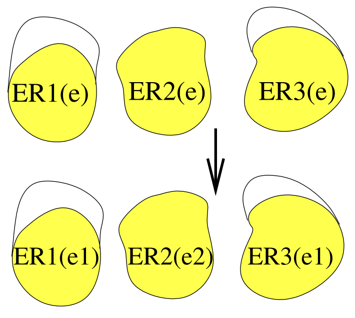


Fig. 9. Disconnected ERs.

8.2 EC-guided splitting

Algorithm 2 for generating minimal regions explores expansions $\sqcap_k$ and $\sqcap^k$ of some multisets covering the ER of an event (see Figure 10). When excitation closure does not hold, all these multisets are stored. Finally, given an event without excitation closure, the expansion r containing the maximum number of events with constant gradient (i.e. the expansion where less effort is needed to transform it into a region by splitting) is selected as source for splitting. The idea is to add new regions covering a non-excitation closed event e in order to make them excitation closed (by reducing the set $\bigcap_{q \in {}^\star e} supp(q)$). For instance, in Figure 10 assume that initially in ER(e) both event a and b have non-constant gradient (events with non-constant gradient are shown at the bottom of every set in the figure), and after the two expansions on event a leading to multiset r_6, event a has constant gradient. In the rest of multisets shown, neither

Algorithm 3: BisimulationBasedAlgorithm

```
Input: Transition system TS = $(S, E, A, s_{in})$,
       $k_{max}$ maximal bound allowed
Output: Petri net PN = $(P, T, W, M_0)$ bisimilar to TS
1  begin
2      repeat
3          $k = 1$
4          repeat
5              $R$ =GenerateMinimalRegions(TS,$k$)
6              $k = k + 1$
7          until $k > k_{max}$ or ExcitationClosed(TS,$R$)
8          if not (ExcitationClosed(TS,$R$)) then
9              TS = SplitLabels(TS)
10         end
11     until ExcitationClosed(TS,$R$)
12     PN = BoundedPNDerivation(TS,$R$)
13 end
```

a nor b achieved constant gradient. In this situation, r_6 will be selected.

Given the selected expansion r where some events have non-constant gradient, let $| \triangle_r (e)|$ represent the number of different gradients for event e in r. The event a with minimal $| \triangle_r (a)|$ is selected for splitting. Let $g_1, g_2, \ldots g_n$ be the different gradients for a in r. As explained in Definition 8.1, event a is split into n different events, one for each gradient g_i. Intuitively, the splitting of the event with minimal $| \triangle_r (e)|$ represents the minimal necessary legalization of some illegal event in r in order to convert it into a region.

8.3 Bisimulation-based algorithm

A complete algorithm for deriving a PN with bisimilar reachability graph to the initial TS is presented in Algorithm 3. The algorithm combines the generation of minimal regions (Algorithm 2, presented in the previous section) together with the label splitting technique presented in the previous section, which is applied only when excitation closure does not hold for the actual TS and the maximal allowed bound (k_{max}) is reached.

The label splitting technique can be used also in the language-inclusion approach: if some events are considered to be *critical* in the sense that no over-approximation must be done for these events, then one can use label splitting to force excitation closure on these events. The corresponding algorithm will be similar to Algorithm 3, but where only a subset of the events (a new input of the algorithm) are checked for excitation closure.

9 Experiments

9.1 Mining experiments

The mining of some examples is summarized in Table 1. Following the two-step mining approach from [29], we have obtained the transition systems from each log with the *FSM Miner* plugin available in ProM. For each log, columns report the number of states of the initial log $|S|$, number of states of the minimal bisimilar transition system $|[S]|$ (that gives an idea of the amount of redundancy present in the initial log) and number of events $|E|$. Next, the number of places $|P|$ and transitions $|T|$ of the PN obtained by synthesis is reported. For each version of the mining algorithm (safe and 2-bounded), the number of places of the mined PN and number of states of the corresponding minimal bisimilar reachability graph are reported. The CPU time for the mining of all examples but the last one has taken less than two seconds. The mining of pn_ex_10, 2-bounded version, took five seconds. Finally the same information is provided for two well-known mining algorithms in ProM: the *Parikh Language-based Region* and the *Heuristics* [34] miners. The number of unconnected transitions ($|T_U|$) derived by the Parikh miner and the number of invisible transitions introduced by the Heuristic miner is also reported ($|T_I|$).

The numbers in Table 1 suggest some remarks. If the synthesis is compared with the mining in the case of safe PNs, it should be noticed that even for those small examples the number of transitions is reduced, due to the absence of label splitting (see row for pn_ex_10). The number of places is also reduced in general. It should also be noticed that 2-bounded mining represents the log more accurately, and thus more places are needed with respect to the mining of safe nets. Sometimes the mined PN accepts more traces but the corresponding minimal bisimilar transition system has less states, e.g. pn_ex_10: after over-approximating the initial TS, several states become bisimilar and can be minimized.

The Parikh miner tends to derive very aggressive abstractions, as it is demonstrated in the pn_ex_10 and herbstFig6p21 logs. Sometimes the Petri nets obtained with this miner contain isolated transitions, because the miner could not find places connecting them to the net. The Heuristics miner is based on the frequency of patterns in the log. The miner derives a heuristic net that can be afterwards converted to Petri net with ProM. Some of the Petri nets obtained with this conversion turned out to be unbounded (denoted with symbol ∞ in the table), and contain a significant amount of invisible transitions. This miner is however very robust to noise in the log. In conclusion, different miners can achieve different mining goals, widening the application of Process mining into several directions.

9.2 Synthesis experiments

In this section a set of parameterizable benchmarks are synthesized using the methods described in this paper. The following examples have been artificially created:

1) A model for n processes competing for m shared resources, where $n > m$. Figure 11(a) describes the Petri net[3],

3. A simplified version of this model was also synthesized by SYNET [7] in [4].

				synthesis		mining									
				`petrify` safe		`Genet` safe		`Genet` 2-bounded		ProM Parikh			ProM Heuristics		
benchmark	$\|S\|$	$\|[S]\|$	$\|E\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|[S]\|$	$\|P\|$	$\|[S]\|$	$\|P\|$	$\|T_U\|$	$\|[S]\|$	$\|P\|$	$\|T_I\|$	$\|[S]\|$
groupedFollowsa7	18	10	7	7	7	6	11	7	11	7	0	10	7	1	8
groupedFollowsal1	15	15	7	8	9	10	16	12	15	7	0	7	14	10	22
groupedFollowsal2	25	25	11	15	11	15	25	15	25	11	0	13	15	3	25
herbstFig6p21	16	16	7	11	13	7	22	11	16	1	6	2	18	15	∞
herbstFig6p34	32	32	12	16	13	16	34	18	32	8	2	12	19	12	∞
herbstFig6p41	20	18	14	16	14	16	18	16	18	17	0	18	14	0	18
staffware_15	31	24	19	20	20	18	22	19	31	18	0	21	19	0	19
pn_ex_10	233	210	11	64	218	13	281	16	145	8	2	14	41	25	∞

TABLE 1
PN mining applied to event logs from [2].

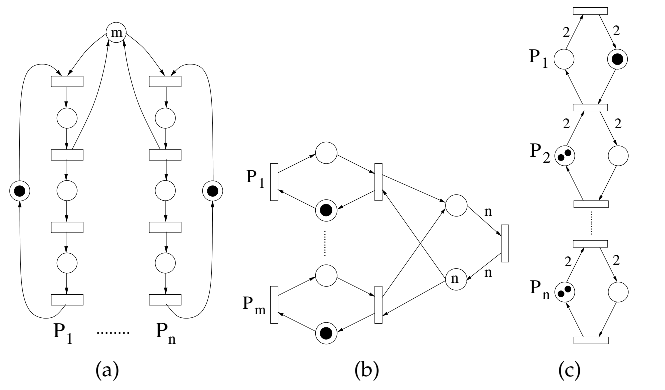


Fig. 11. Parameterized benchmarks: (a) n processes competing for m shared resources, (b) m producers and n consumers, (c) a 2-bounded pipeline of n processes.

2) A model for m producers and n consumers, where $m > n$. Figure 11(b) describes the Petri net.
3) A 2-bounded pipeline of n processes. Figure 11(c) describes the Petri net.

Table 2 contains a comparison between a synthesis algorithm of safe Petri nets [11], implemented in the tool `petrify`, and the synthesis of general Petri nets as described in this paper, implemented in the prototype tool `Genet`. For each benchmark, the size of the transition system (states and arcs), number of places and transitions and cpu time is shown for the two approaches. The transition system has been initially generated from the Petri nets. Clearly, the methods developed in this paper generalize those of the tool `petrify`, and particularly the generation of minimal regions for arbitrary bounds has significantly more complexity than its safe counterpart. However, many of the implementation heuristics and optimizations included in `petrify` must also be extended and adapted to `Genet`. Provided that this optimization stage is under development in `Genet`, we concentrate on the synthesis of small examples. Hence, cpu times are only preliminary and may be improved after the optimization of the tool.

The main message from Table 2 is the expressive power of the approach developed in this paper to derive an event-based representation with minimal size. If the initial transition system is excitation closed, using a bound large enough one can guarantee no splitting and therefore the number of events in the synthesized Petri net is equal to the number of different events in the transition system. Note that the excitation closure holds for all the benchmarks considered in Table 2, because the transition systems considered are derived from the corresponding Petri nets shown in Figure 11.

Label splitting is a key method to ensure excitation closure. However, its application can degrade the solution (both in terms of cpu time required for synthesis and the quality of the solution obtained), specially if many splittings must be performed to achieve excitation closure, as it can be seen in the benchmarks SharedResource(5,2) and BoundedPipeline(7). In these examples, the synthesis performed by `petrify` derives a safe Petri net with one order of magnitude more transitions than the bounded synthesis method of this paper.

10 Conclusions

In this paper the theory of regions have been extended into several dimensions to allow for a uniform approach for the synthesis and mining of general Petri nets from state-based specifications. The practicality of the approach is demonstrated by presenting efficient algorithms, heuristics and data structures.

We foresee a demanding need for synthesis and discovery in the near future, in applications ranging from Healthcare [23] to the Web [31], and methods like the ones presented in this paper may be crucial.

The theory presented in this paper has been implemented in a tool [1], and the experimental results reported are promising. However, provided the complexity of some of the problems faced, more research is expected into that direction.

References

[1] Genet. http://www.lsi.upc.edu/∼jcarmona/genet.html.
[2] Process mining. www.processmining.org.
[3] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.
[4] E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1998.

			Petrify			Genet		
benchmark	$\vert S\vert$	$\vert E\vert$	$\vert P\vert$	$\vert T\vert$	CPU	$\vert P\vert$	$\vert T\vert$	CPU
SharedResource(3,2)	63	186	15	16	0s	13	12	0s
SharedResource(4,2)	243	936	20	24	5s	17	16	0s
SharedResource(5,2)	918	4320	48	197	3h	24	20	0s
SharedResource(4,3)	255	1016	21	26	2s	17	16	0s
SharedResource(6,4)	4077	24372	36	68	2h40m	25	24	18s
SharedResource(7,5)	16362	114408	–	–	time	29	28	25m
ProducerConsumer(3,2)	24	68	9	10	0s	8	7	0s
ProducerConsumer(4,2)	48	176	11	13	0s	10	9	0s
ProducerConsumer(3,3)	32	92	10	13	0s	8	7	0s
ProducerConsumer(4,3)	64	240	12	17	1s	10	9	0s
ProducerConsumer(6,3)	256	1408	16	25	25s	14	13	0s
ProducerConsumer(8,3)	1024	7424	20	33	5m	18	17	2s
ProducerConsumer(8,5)	1536	11520	22	49	17m	18	17	1h10m
BoundedPipeline(4)	81	135	14	9	0s	8	5	0s
BoundedPipeline(5)	243	459	17	11	1s	10	6	1s
BoundedPipeline(6)	729	1539	27	19	6s	12	7	6s
BoundedPipeline(7)	2187	5103	83	68	110m	14	8	48s
BoundedPipeline(8)	6561	16767	34	23	8m	16	9	12m
BoundedPipeline(9)	19683	54765	37	23	46m	18	10	1h50m

TABLE 2
Synthesis of parameterized benchmarks

[5] R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser. Process mining based on regions of languages. In *Proc. 5th Int. Conf. on Business Process Management*, pages 375–383, Sept. 2007.

[6] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computer-Aided Design*, 35(8):677–691, 1986.

[7] B. Caillaud. Synet : A synthesizer of distributable bounded Petri-nets from finite automata. http://www.irisa.fr/s4/tools/synet/, 2002.

[8] J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering Petri nets from event logs. In M. Dumas, M. Reichert, and M. C. Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2008.

[9] J. Carmona, J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A symbolic algorithm for the synthesis of bounded Petri nets. In van Hee and Valk [33], pages 92–111.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[11] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, Aug. 1998.

[12] P. Darondeau. Unbounded Petri net synthesis. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–438. Springer, 2003.

[13] P. Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with Petri nets. *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 2107–2112, Dec. 2005.

[14] P. Darondeau. Private communication, 2008.

[15] J. Desel and W. Reisig. The synthesis problem of Petri nets. *Acta Inf.*, 33(4):297–315, 1996.

[16] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[17] A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part I, II. *Acta Informatica*, 27:315–368, 1990.

[18] J. Engelfriet. Determinacy - (observation equivalence = trace equivalence). *Theor. Comput. Sci.*, 36:21–25, 1985.

[19] P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. A trace semantics for petri nets. *Inf. Comput.*, 117(1):98–114, 1995.

[20] P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. An event structure semantics for general petri nets. *Theor. Comput. Sci.*, 153(1&2):129–170, 1996.

[21] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[22] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, British Columbia, Canada, Aug. 1987.

[23] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. van der Aalst, and P. J. M. Bakker. Process mining in healthcare - a case study. In L. Azevedo and A. R. Londral, editors, *HEALTHINF (1)*, pages 118–125. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2008.

[24] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[25] M. Mukund. Petri nets and step transition systems. *Int. Journal of Foundations of Computer Science*, 3(4):443–478, 1992.

[26] S. M. Nowick and D. L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 318–321. IEEE Computer Society Press, Nov. 1991.

[27] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962. (technical report Schriften des IIM Nr. 3).

[28] M. H. ter Beek, C. A. Ellis, J. Kleijn, and G. Rozenberg. Synchronizations in team automata for groupware systems. *Computer Supported Cooperative Work*, 12(1):21–69, 2003.

[29] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process mining: A two-step approach to balance between underfitting and overfitting. Technical Report BPM-08-01, BPM Center, 2008.

[30] W. M. P. van der Aalst and C. W. Günther. Finding structure in unstructured processes: The case for process mining. In T. Basten, G. Juhás, and S. K. Shukla, editors, *ACSD*, pages 3–12. IEEE Computer Society, 2007.

[31] W. M. P. van der Aalst and H. M. W. E. Verbeek. Process mining in web services: The websphere case. *IEEE Data Eng. Bull.*, 31(3):45–48, 2008.

[32] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In van Hee and Valk [33], pages 368–387.

[33] K. M. van Hee and R. Valk, editors. *Applications and Theory of Petri Nets, 29th International Conference, PETRI NETS 2008, Xi'an, China, June 23-27, 2008. Proceedings*, volume 5062 of *Lecture Notes in Computer Science*. Springer, 2008.

[34] A. Weijters, W. van der Aalst, and A. A. de Medeiros. Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology, 2006.