New Region-Based Algorithms for Deriving Bounded Petri Nets

Josep Carmona, Jordi Cortadella, *Member*, *IEEE*, and Mike Kishinevsky, *Senior Member*, *IEEE*

Abstract—The theory of regions was introduced in the early nineties as a method to bridge state and event-based models. This paper tackles the problem of deriving a Petri net from a state-based model, using the theory of regions. Some of the restrictions required in the traditional approach are dropped in this paper, together with significant extensions that make the approach applicable in new scenarios. One of these scenarios is Process Mining, where accepting (discovering) additional behavior in the synthesized Petri net is sometimes valued. The algorithmic emphasis used in this paper contributes to the demystification of the theory of regions as been only a good theoretical exercise, opening the door for its application in the industrial domain.

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

1 Introduction

State-based representations, like FSMs [17], I/O Automata [21], Team Automata [27], or burst-mode automata [24], 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 [25] or CCS [22], 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 [18] provides a bridge between state- and event-based representations. This theory, which is now almost two decades old, was introduced to solve the synthesis problem. This problem consists of 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 (TS) into a Petri net. The theory was initially defined for *elementary transition systems* deriving 1-bounded Petri nets, whereas in [11] this restriction was dropped. Mukund [23] extended the theory of [18] for k-bounded Petri nets with weighted arcs, while in this paper we extend [11] for the same purpose.

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*. The available tool support for synthesis based on the theory of regions is relegated to the academic domain. For synthesis, we can mention the tools Petrify [11] and Synet [7]. For process mining, the Parikh language miner [31] (within the ProM tool) and the ViPTool [5], [4].

In this paper, we provide a uniform approach for deriving Petri nets from transitions systems, based on 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 to derive (mine) k-bounded Petri nets whose corresponding language includes the one from the initial transition systems. 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 extends the work presented in [9], [8].

1.1 Two Introductory Examples

Let us introduce the two main results of this paper by means of two simple examples.

Synthesis of k-bounded Petri nets. Fig. 1a shows a 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 $\{aaa, ab, ba, bb\}$. From this transition system, a 6-bounded Petri net can be derived by the theory developed in this paper, as shown in Fig. 1b. 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 five transitions and five places. The theory in this paper extends the one presented in [11], which was restricted to 1-bounded Petri nets, representing a significant step toward 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

- *J. Carmona and J. Cortadella are with the Universitat Politècnica de Catalunya, C/Jordi Girona, 1-3, 08034 Barcelona, Spain. E-mail: jcarmona@lsi.upc.edu, jordi.cortadella@upc.edu.*
- *M. Kishinevsky is with Intel Corporation, 2111 NE 25th Ave., Hillsboro, OR 97124. E-mail: michael.kishinevsky@intel.com.*

Manuscript received 26 Nov. 2008; revised 27 July 2009; accepted 22 Aug. 2009; published online 3 Sept. 2009.
Recommended for acceptance by S. Shukla.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-11-0590.
Digital Object Identifier no. 10.1109/TC.2009.131.

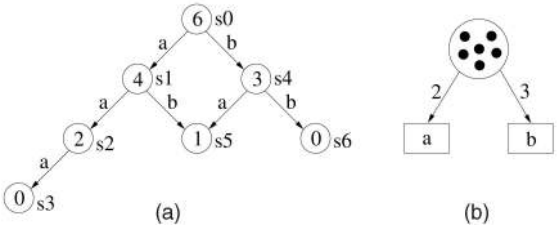


Fig. 1. (a) TS and (b) equivalent 6-bounded Petri net.

events. Each one of the regions corresponds to a place in the derived Petri net. For instance, Fig. 1a shows the (only one) region r corresponding to the place shown in Fig. 1b. 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 Fig. 1b. The behavior of this net corresponds exactly to the one described in Fig. 1a.

Mining of k-bounded Petri nets. For some applications, reproducing exactly the behavior is not a requirement, but to have instead a clear visualization of what are the main processes is preferred. Process Mining [29] 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 toward 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], [31] with the same goal.

Let us use the simple example of Fig. 2a, representing a 3-input OR gate. Although simple, this example illustrates a typical behavior that cannot 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, a 1-bounded Petri net representing the

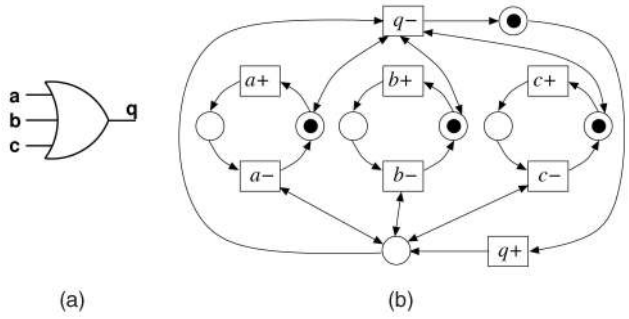


Fig. 2. (a) The 3-or gate and (b) mined Petri net.

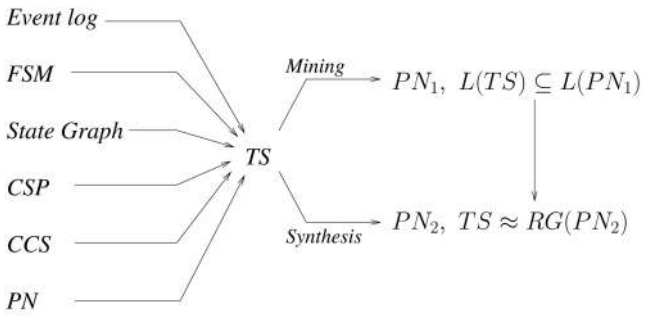


Fig. 3. Flow for Petri net derivation.

behavior has 11 places, 17 transitions, and 74 arcs. If a 3-bounded Petri net is synthesized instead, it does not get significantly better: eight places, eight 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 Fig. 2b. 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 is allowed. The intention is to restrict extra behaviors as much as possible and to obtain a tight overapproximation of the observed behaviors.

1.2 Overall Approach for Synthesis and Mining

Fig. 3 describes the complete flow for k-bounded PN derivation. The transition system representing the input of our methods can be obtained from some of the traditional formalisms to specify a concurrent system (FSM, CSP, CCS), from graphs describing hardware (State graphs) or also from event logs [28]. Our approach offers two possibilities for obtaining the corresponding Petri nets: mining and synthesis. In mining, the language formed by the set of feasible traces in the derived PN (PN_1) includes the language of the transition system. In synthesis, the obtained PN (PN_2) has a reachability graph bisimilar to the transition system. Mining can be considered as an intermediate step for synthesis [12], and this is why an arc exists between PN_1 and PN_2 in Fig. 3. In this paper, we emphasize the mining due to the applications that it has for discovering unseen behaviors, as was demonstrated in the Process Mining area [30], [29].

The organization of the paper is as follows: Section 2 includes the theory needed for 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 synthesis and mining, 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* (labeled) transitions, *and s_{in} is the* initial state.

Let $\mathrm{TS} = (S, E, A, s_{in})$ 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: $\forall s \in S : s_{in} \stackrel{*}{\rightarrow} s$.

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

Definition 2.2 (Simulation, Bisimulation [2]). *Let $\mathrm{TS}_1 = (S_1, E, A_1, s_{in_1})$ and $\mathrm{TS}_2 = (S_2, E, A_2, s_{in_2})$ 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 vice versa with relation π^{-1}, TS_1 and TS_2 are *bisimilar* [2].

Definition 2.3 (Petri net [25]). *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 $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ 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 $M \stackrel{t}{\rightarrow} M'$.

The set of all markings reachable from 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 $M_1 \stackrel{t}{\rightarrow} M_2$. We use $L(\mathrm{PN})$ as a shortcut for $L(\mathrm{RG}(\mathrm{PN}))$.

2.2 Multisets

In general, a region in a TS is a multiset where additional conditions hold [23]. We recap in this section the main definitions and operations on multisets. From now on, we will assume a $\mathrm{TS} = (S, E, A, s_{in})$ providing the necessary context for the corresponding definitions.

Given the 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 ($\mathrm{supp}(r)$) is defined as $\{s \in S | r(s) > 0\}$. The *power* of r ($r^\diamond$) is $\max_{s \in S} r(s)$. For instance, for $r = \{s_1^3, s_2^2, s_3\}$, $\mathrm{supp}(r) = \{s_1, s_2, s_3\}$ and $r^\diamond = 3$.

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

$$(r_1 \cup r_2)(s) = \max(r_1(s), r_2(s)),$$
$$(r_1 \cap r_2)(s) = \min(r_1(s), r_2(s)),$$
$$(r_1 - r_2)(s) = \max(0, r_1(s) - r_2(s)).$$

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 $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{K}$ when $r(s) = 0$, $r(s) = 1, \ldots, r(s) = k$, for every $s \in S$, respectively.

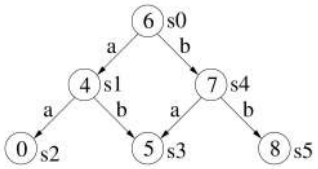
Multiset r is *k-bounded* if for all $s \in S : r(s) \leq k$. The multiset r_1 is a *subset* of r_2 ($r_1 \subseteq r_2$) if $\forall s \in S : r_1(s) \leq r_2(s)$. As usual, we will denote by $r_1 \subset r_2$ the fact that $r_1 \subseteq r_2$ and $r_1 \neq r_2$. The *k-topset* of r, denoted by $\top_k(r)$, 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 is 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 nonconstant gradient in r if there are two transitions (s_1, e, s_1') and (s_2, e, s_2') such that $\Delta_r(s_1, e, s_1') \neq \Delta_r(s_2, e, s_2')$. As example, consider the following multiset r. Event a has nonconstant gradient in r, because $\Delta_r(s_0, a, s_1) = -2 \neq -4 = \Delta_r(s_1, a, s_2)$. However, b has constant gradient 1.



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

For instance, the multiset shown in Fig. 1a is a region, whereas the one shown in the figure above is not a region due to event a.

The original notion of region from [18] 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 $(\mathbf{0}, \mathbf{1}, \ldots, \mathbf{K})$. The set of nontrivial general regions of TS will be denoted by R_{TS}.

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*

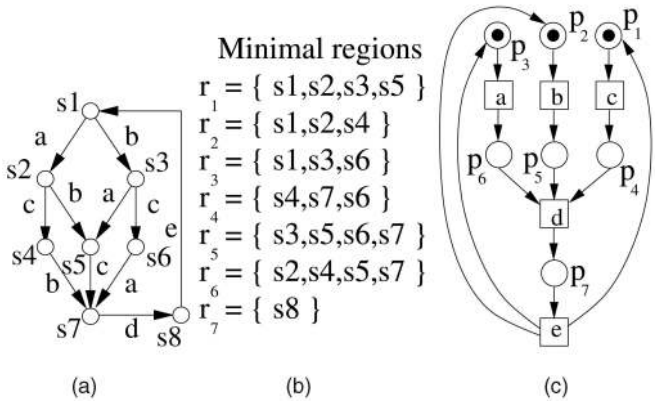


Fig. 4. (a) Initial TS, (b) 1-bounded minimal regions, and (c) Petri net derived by Algorithm 1.

$$\Delta_r(e) = \Delta_r(s, e, s').$$

For instance, in the example of Fig. 1a, $\Delta_r(a) = -2$.

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 PN from a set of k-bounded regions. The relation between the initial transition system and the PN will be established in the next two sections: language inclusion (Section 4), and bisimilarity (Section 5). First, the following definition describes these sets of states that must be covered by a region in order to have an arc in the derived Petri net.

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

$$\mathrm{ER}(e) = \{s | \exists s' : (s, e, s') \in E\}.$$

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

$$\mathrm{SR}(e) = \{s | \exists s' : (s', e, s) \in E\}.$$

For convenience, $\mathrm{ER}(e)$ and $\mathrm{SR}(e)$ will be also considered as multisets of states when necessary. As example, in the TS shown in Fig. 4a, $\mathrm{ER}(a) = \{s_1, s_3, s_6\}$ and $\mathrm{SR}(a) = \{s_2, s_5, s_7\}$.

Definition 3.2 (Pre- and postregions). *A region r is a preregion of e if $\mathrm{ER}(e) \subseteq r$. A region r is a postregion of e if $\mathrm{SR}(e) \subseteq r$. The sets of pre- and postregions of an event e are denoted by ${}^\circ e$ and e°, respectively.*

Intuitively, pre- and postregions will be represented by predecessor and successor places of the event in the derived PN. Following the example of Fig. 4, the region r_3 is a preregion of a, whereas r_6 is a postregion of a. Next, we define the basic objects that will be used to establish the weight on the arcs in the derived PN:

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 an event e and a preregion r, the multiset $\mathrm{TOP}(r, e)$ is the multiset q such that $q = \top_g(r)$, $\mathrm{ER}(e) \subseteq \top_g(r)$, and $\mathrm{ER}(e) \not\subseteq \top_{g+1}(r)$. The degree g of $\mathrm{TOP}(r, e)$ is denoted as $\text{MAX-K-TOP}(r, e)$.*

Informally, $\mathrm{TOP}(e, r)$ is the smallest topset of r still covering $\mathrm{ER}(e)$. Note that this is always a single multiset. The value $\text{MAX-K-TOP}(r, e)$ 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 Fig. 1a, if we consider the region r shown, $\mathrm{TOP}(a, r) = \top_2(r) = \{s_0^6, s_1^4, s_2^2, s_4^3\}$, and therefore $\text{MAX-K-TOP}(r, a) = 2$.

Now we define 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 ${}^*e$ and defined as follows:*

$${}^*e = \{q \mid \exists r \in {}^\circ e \wedge q = \mathrm{TOP}(e, r)\}.$$

In the PN derived by Algorithm 1, every region becomes a place and each event a transition. The flow relation from places to transitions is defined with respect to the enabling topsets: for a preregion r of an event e, $g = \text{MAX-K-TOP}(r, e)$ tokens can be removed from place r (lines 7 and 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 and 10). The flow relation from transition e to places is defined for regions covering $\mathrm{SR}(e)$ and the corresponding gradient (line 13). For instance, in the example of Fig. 1, the region r shown is preregion of both a and b, with $\text{MAX-K-TOP}(r, a) = 2$ and $\text{MAX-K-TOP}(r, b) = 3$, and so the corresponding weighted arcs derived by Algorithm 1 are connecting the region to transitions a and b, respectively. The initial marking of the region is 6 due to line 3 of the algorithm. Another example of application of Algorithm 1 is shown in Figs. 4a, 4b, and 4c.

Algorithm 1: BoundedPNDerivation

Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$,
R is a set of regions from TS
Output: Petri net $\mathsf{PN} = (R, E, W, M_0)$

```
1  begin
2    foreach region r ∈ R do
3      M0[r] = r(s_in)
4    end
5    foreach event e ∈ E do
6      foreach r ∈ °e do
7        g = MAX-K-TOP(r, e)
8        W = W ∪ {(r, e) with weight g}
9        if (Δ_r(e) > −g) then
10         W = W ∪ {(e, r) with weight g + Δ_r(e)}
11       end
12     end
13     foreach r ∈ e° do W = W ∪ {(e, r) with weight Δ_r(e)}
14   end
15 end
```

4 Mining

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) and the original TS? We will show that $L(\mathrm{PN}) \subseteq L(\mathrm{TS})$. This result is consistent with the classical theory of region-based synthesis according to which the net synthesized using minimal regions is an approximation from above (i.e., includes the original language, see for instance [12]). In this section, we assume that the set of minimal regions has been computed: in Section 6, we will present an efficient algorithm to compute minimal regions.

Let us consider the example of Fig. 4 to illustrate the main message of this section. Fig. 4a shows a TS. Assume that 1-bounded minimal regions from this TS are used in Algorithm 1. These regions are reported in Fig. 4b, and the PN derived from Algorithm 1 on these regions is shown in Fig. 4c. It is easy to see that $L(\mathrm{PN}) \supset L(\mathrm{TS})$: the sequence $cbade$ belongs to $L(\mathrm{PN})$ but it does not belong to $L(\mathrm{TS})$.

The remainder of this section formalizes the construction of Petri nets based on language overapproximation and presents an important result stating a minimality property on the derived PN. Formal proofs and necessary lemmas are provided in Appendix.

4.2 Mining Properties

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

1. $L(\mathrm{TS}) \subseteq L(\mathrm{PN})$,
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:

$$\text{For any k-bounded } \mathrm{PN}' = (P', T', W', M_0') \\ s.t.\ T' = E:\ L(\mathrm{TS}) \subseteq L(\mathrm{PN}') \Rightarrow L(\mathrm{PN}) \subseteq L(\mathrm{PN}').$$

Therefore, the algorithm generates the tightest overapproximation (measured by language containment) among all possible Petri nets that can be obtained without splitting labels. Since Algorithm 1 clearly guarantees item 2, it only remains to demonstrate the two other properties. First, language inclusion holds due to the following result:

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

This theorem guarantees that no trace is lost when using only the set of minimal regions in TS. Moreover, as the following theorem states, the MLC property holds in the PN:

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

Hence, this property guarantees a tight overapproximation of the input language. Compared to the language-based regions from [31], this is a distinguishing factor that might be important if the degree of overapproximation must be controlled.

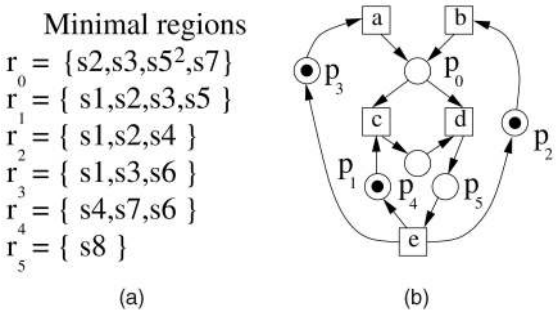


Fig. 5. (a) The 2-bounded minimal regions in the TS of Fig. 4a, where only region r_0 is not 1-bounded, and (b) bisimilar Petri net derived by Algorithm 1.

5 Synthesis

5.1 Motivation

For some applications, deriving a Petri net equivalent to the initial transition system is required. However, as the example of Section 4.1 demonstrates, this cannot be always accomplished. 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 8.1. Finally, in Section 7, excitation closure is used to reduce the number of necessary regions while preserving the properties in the derived net.

5.2 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*

$$\bigcap_{q \in {}^*e} \mathrm{supp}(q) = \mathrm{ER}(e).$$

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

For instance, the TS shown in Fig. 4a is not 1-ECTS, since event c is not excitation closed for the set of regions shown in Fig. 4b:

$$\bigcap_{q \in {}^*c} \mathrm{supp}(q) = r_1 \supset \mathrm{ER}(c),$$

while the rest of events (a, b, d, and e) are excitation closed. However, the TS is 2-ECTS, as Fig. 5 demonstrates. We will say that a TS is ECTS if it is k-ECTS for some k.

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 overapproximation of the TS and hence it cannot be simulated by TS. The following theorem links excitation closure and bisimilarity (see proof in Appendix):

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

For example, the reachability graph of the PN from Fig. 5b is bisimilar to the TS of Fig. 4a.

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 large 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

- Nonminimal [16], or
- Not a preregion or postregion (see Property 6.1), or
- Trivial, i.e., $\mathbf{0}, \mathbf{1}, \ldots, \mathbf{K}$

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 mining, only a *subset irredundant minimal cover* of regions is needed.
- In synthesis, 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 property will be useful to ensure the generation of minimal pre/postregions [9]:

Property 6.1. *Let* $\mathrm{TS} = (S, \Sigma, E, s_{in})$ *be a transition system. Then, any region* $r \neq \mathbf{0}$ *is the preregion or the postregion of some event* e.

Due to Property 6.1, 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 nonconstant 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*

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

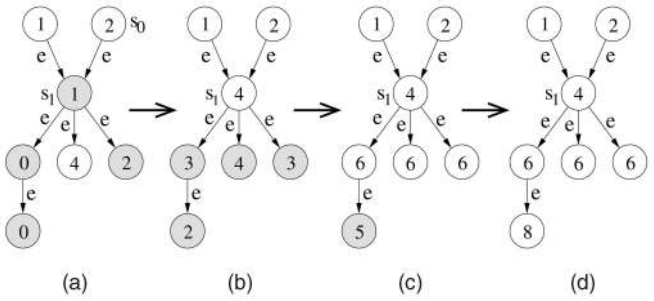


Fig. 6. Successive calculations of $\sqcap^2(r,e)$. States violating the constraint $\Delta_r(e) \geq 2$ are depicted with yellow background.

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

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

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 Fig. 6a 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 gray 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:*

$$\sqcap_g(r,e)(s) = r(s) + \delta_g(r,e,s),$$
$$\sqcap^g(r,e)(s) = r(s) + \delta^g(r,e,s).$$

Intuitively, $\sqcap_g(r,e)$ is a move toward 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 Figs. 6a, 6b, 6c, and 6d. The following theorem formalizes the notion:

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

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. $\mathcal{R}^g(r, e) = \mathcal{R}^g(\sqcap^g(r, e), e)$.

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 nonconstant gradient is chosen at each node (line 7). All possible gradients for that event are explored by means 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.

Algorithm 2: GenerateMinimalRegions

Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$, bound k
Output: R is the set of k-bounded minimal regions from TS

```
1  begin
2    R = ∅
3    P = {ER(e) | e ∈ E} ∪ {SR(e) | e ∈ E}
4    while P ≠ ∅ do
5      r = remove_one_element (P)
6      if r ∉ R then
7        e = event_with_non_constant_gradient (r)
8        (g_min, g_max) = (min Δ_r(e)), max Δ_r(e))
9        g = ⌊(g_min + g_max)/2⌋
10       r_1 = ⊓_g(r, e)
11       if ((r_1° ≤ k) ∧ (1 ⊄ r_1)) then P = P ∪ {r_1}
12       r_2 = ⊓^{g+1}(r, e)
13       if ((r_2° ≤ k) ∧ (1 ⊄ r_2)) then P = P ∪ {r_2}
14     end
15   end
16   R = R \ {r | r is not a region}
17   R = R \ {r | ∃r' ∈ R : r' ⊂ r}
18 end
```

Fig. 7 shows how Algorithm 2 will explore the $\mathrm{ER}(a)$ (center). After successive calculations of $r_1 = \sqcap_{-1}(\mathrm{ER}(a), a)$, it reaches the region shown on the left. Analogously, the region on the right is obtained after successive calculations of $r_2 = \sqcap^0(\mathrm{ER}(a), a)$, and notice that provided that r_2 covers both $\mathrm{ER}(a)$ and $\mathrm{SR}(a)$, it corresponds to a self-loop place for transition a.

Theorem 6.2. *Algorithm 2 calculates all k-bounded minimal regions.*

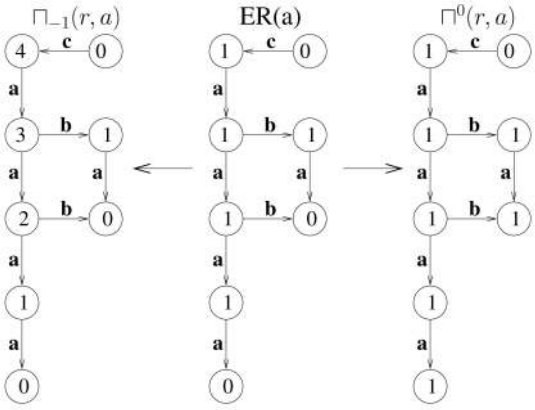


Fig. 7. Exploration of regions in Algorithm 2 for $\mathrm{ER}(a)$: $\min\Delta_r(a) = -1, \max\Delta_r(a) = 0$.

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 [14, 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 [13], by solving a finite system of linear constraints over the integers.
2. Construction of the covering tree [20] 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 [15].

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].

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

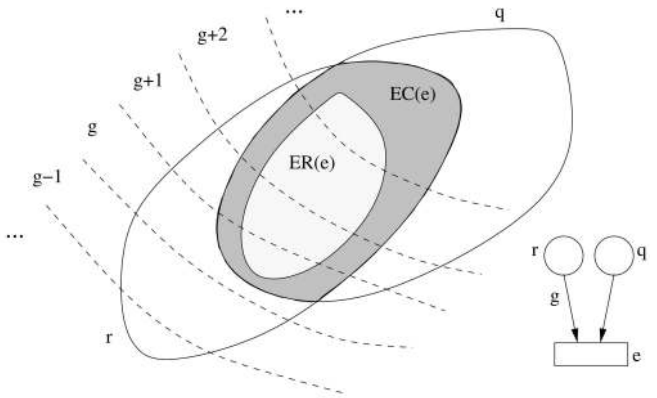


Fig. 8. Situation where the arc $r \xrightarrow{g} e$ can be converted into $p \xrightarrow{g-1} e$.

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*

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

The enabling closure of an event e represents the set of states that enable e if the set of preregions 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 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. Fig. 8 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 $\mathrm{TS} = (S, E, A, s_{in})$ and R a set of regions, let e be an event and a region $r \in {}^\circ e \cap R$ such that $\mathrm{ER}(e) \subseteq \top_g(r)$. If the following equality holds*

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

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, $L(\mathrm{PN}) = L(\mathrm{PN}')$.

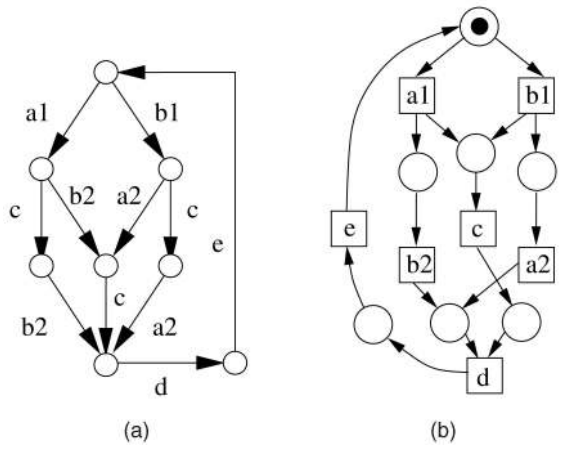


Fig. 9. (a) Transition system after application of label splitting on events a and b, (b) Petri net derived by Algorithm 1 on the set of minimal regions from TS (a).

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 necessarily imply that p_2 is redundant with respect to $R - \{r_1\}$, or vice versa.

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 those 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 cannot be exceeded. If a TS is not k-ECTS and the bound cannot be incremented further, then only PNs overapproximating 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 Fig. 4a. Assume that the maximal bound accepted is 1, and a PN with bisimilar behavior must be generated. As reported in Figs. 4b and 4c, the derived PN on the set of minimal 1-bounded regions accepts more traces than the initial TS. Now let us *split* events a and b in the TS as shown in Fig. 9a. 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 Fig. 9b. The fact that transitions keep the label of the original events

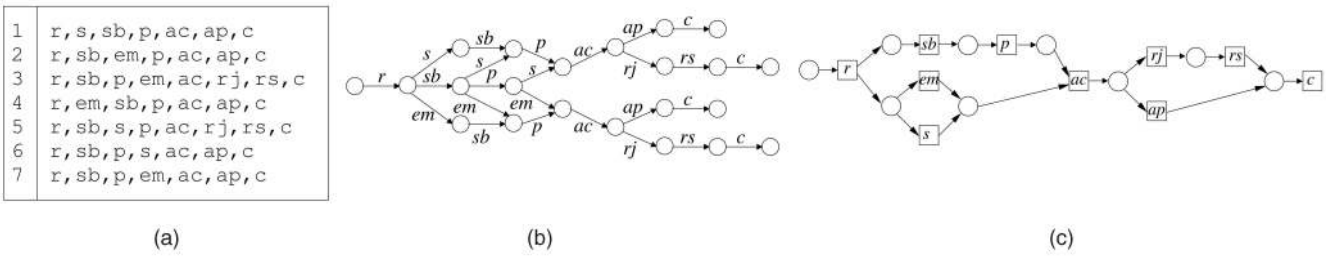


Fig. 10. Process mining: (a) event log, (b) corresponding transition system, and (c) Mined Petri net.

guarantees that the reachability graph of the PN in Fig. 9b is bisimilar to TS of Fig. 4a. The formal definition of the technique is presented:

Definition 8.1 (Label splitting). *Let* $\mathrm{TS} = (S, E, A, s_{in})$ *be a transition system. The splitting of event* $e \in E$ *derives a transition system* $\mathrm{TS}' = (S, E', A', s_{in})$, *with* $E' = E - \{e\} \cup \{e_1, \ldots, e_n\}$, *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.*

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.

In [9], some effective heuristics based on the excitation closure condition (see Definition 5.1) are presented for label splitting. Here, we briefly sketch the main idea of each heuristic:

- *Splitting disconnected* ERs: when the ER of an event can be partitioned into disconnected components, and some of these components satisfy the excitation closure.
- EC-*guided splitting*: for a nonexcitation closed event e, the multisets reached when expanding $\mathrm{ER}(e)$ (see Section 6) are ordered according to the number of events with constant gradient, and the more promising multiset (the one with more events with constant gradient) induce the event to split.

8.1 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.

Algorithm 3: BisimulationBasedAlgorithm

Input: Transition system $\mathsf{TS} = (S, E, A, s_{in})$, k_{max} maximal bound allowed
Output: Petri net $\mathsf{PN} = (P, T, W, M_0)$ 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
```

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 overapproximation must be done for these events, then one can use label splitting to force only excitation closure on these events.

9 Experiments

In the next two sections, we show particular uses of the algorithms presented in this paper (implemented in the tool `Genet`) for different applications. In the first section, we illustrate how to apply the algorithms in the area of Process Mining.[3] In the second section, we abstract away the particular application and show the benefits of representing complex behaviors with k-bounded Petri nets.

9.1 Synthesis and Mining from Event Logs

First, we provide a simple example to illustrate the applicability of the flow presented in Section 1.2 in the emerging area of Process Mining [30]. The example is taken from [29] and considers the process of handling customer orders. The starting point in Process mining is typically a set of traces representing the log of a system. In our example, the event log contains seven traces with the following activities: $r = register$, $s = ship$, $sb = send\_bill$, $p = payment$, $ac = accounting$, $ap = approved$, $c = close$, $em = express\_mail$, $rj = rejected$, and $rs = resolve$. The set of traces recorded is shown in Fig. 10a, while Fig. 10b shows

3. We decided to apply both the synthesis and mining algorithms in Section 9.1 on purpose: traditionally synthesis has also been applied in Process Mining [6], [28].

TABLE 1
Petri Net Mining Applied to Event Logs from `processmining.org`

				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	∞

one of the possible TSs representing the event log, constructed with the methods described in [28]. Since this TS is 1-ECTS, both synthesis and mining derive the same 1-bounded PN depicted in Fig. 10c.

The synthesis/mining of some small examples is summarized in Table 1. Following the two-step mining approach described above (Log $\rightarrow$ TS $\rightarrow$ PN), we have obtained the transition systems from each log with the *FSM Miner* plug-in available in the tool ProM [32]. 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 the synthesis tool `petrify` [11] is reported. For each version of the mining algorithm (safe and 2-bounded) implemented in the tool `Genet`, the number of places of the mined PN and number of states of the corresponding minimal bisimilar reachability graph are reported. The mining of the examples in Table 1 took few 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 and places is reduced, due to the absence of label splitting (see row for pn_ex_10). In mining, the Parikh miner tends to derive very aggressive abstractions, as it is demonstrated in the pn_ex_10 and herbst-Fig6p21 logs. Sometimes, the Petri nets obtained with this miner contain unconnected transitions, because the miner could not find places connecting them to the net.

Table 2 compares the results reported in the recent paper [31] for the Parikh miner.[4] The goal of this experiment is twofold: first, to show the capability to deal with large logs, and second, to provide some measure on the quality of the derived results. For the latter, we used the well-known *fitness* and *appropriateness* measures [26]. Briefly, fitness is a metric that evaluates the extent to which the log traces can be associated with traces in the model, and appropriateness quantifies the degree of accuracy in which the model describes the observed behavior, combined with the degree of clarity in which it is represented. Both metrics can be normalized to be a real number between 0 (low) to 1 (high).

Using the approach described at the beginning of this section (in particular, Log $\rightarrow$ TS $\overset{mining}{\rightarrow}$ PN), we have been able to derive PNs with high fitness and appropriateness factors in significantly less CPU time than the one reported in [31].

9.2 Synthesis Experiments

In this section, a set of parameterizable benchmarks is synthesized using the methods described in this paper. The goal of this experiment is to illustrate the capability in reproducing complex behaviors in a very succinct manner, by using the theory of k-bounded synthesis described in this paper. The following examples have been artificially created:

1. A model for n processes competing for m shared resources (SR), where $n > m$. Fig. 11a describes the Petri net.[5]
2. A model for m producers and n consumers (PC), where $m > n$. Fig. 11b describes the Petri net.
3. A 2-bounded pipeline of n processes (BP). Fig. 11c describes the Petri net.

Table 3 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 tool `Genet`. For each benchmark, the size of the transition system (states and arcs), number of places and transitions, and CPU time are shown for the two approaches. The transition system has been generated from the Petri nets.

The main message from Table 3 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 3, because the transition systems considered are derived from the corresponding Petri nets shown in Fig. 11.

4. Only CPU times are provided in [31].

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

TABLE 2
Large Benchmarks from [31]

log	#cases	$\|S\|$	Parikh	Genet	Fitness	App.
a12f0n00_1	200	18	0.4s	0.2s	1.000	1.000
a12f0n00_2	600	18	0.9s	0.2s	1.000	1.000
a12f0n00_3	1000	18	1.1s	0.3s	1.000	1.000
a12f0n00_4	1400	18	1.2s	0.1s	1.000	1.000
a12f0n00_5	1800	18	1.2s	0.2s	1.000	1.000
a22f0n00_1	100	322	1m40s	12s	0.896	0.983
a22f0n00_2	300	993	5m07s	11s	0.893	0.978
a22f0n00_3	500	1170	7m50s	6s	0.893	0.988
a22f0n00_4	700	585	10m24s	6s	0.893	0.985
a22f0n00_5	900	619	12m29s	5s	0.890	0.989
a32f0n00_1	100	940	32m14s	37m56s	0.909	0.971
a32f0n00_2	300	1647	66m24s	36m04s	0.920	0.965
a32f0n00_3	500	2034	106m34s	29m26s	0.920	0.947
a32f0n00_4	700	2266	163m40s	39m29s	0.920	0.955
a32f0n00_5	900	2461	174m01s	27m19s	0.919	0.939

10 Related Work

Regarding 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 [18], [23], [16], [3], [19], making the approach presented in this paper more suitable for a practical implementation. Moreover, regarding the classical theory of region-based synthesis, the methods described in this paper drop the restriction on elementary transition systems.

Our approach for 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. The Parikh miner approach, which is also based on the theory of regions [31], uses a 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. As has been shown in the previous section, 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.

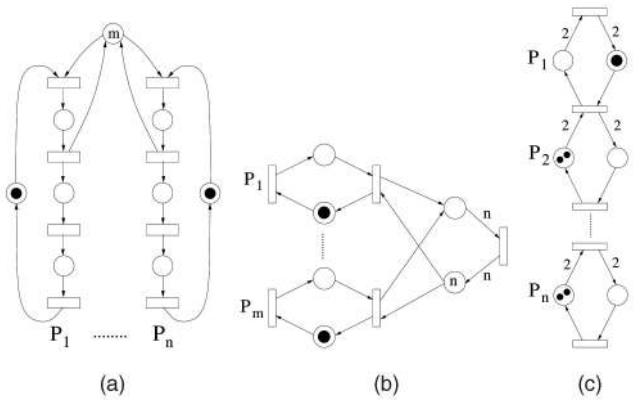


Fig. 11. Parameterized benchmarks: (a) n processes competing for m shared resources, (b) m producers and n consumers, and (c) a 2-bounded pipeline of n processes.

TABLE 3
Synthesis of Parameterized Benchmarks

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

Another interesting approach is presented in [5], [4], where regions of languages are applied directly to the log, assuming that each trace in the log is a particular partial order. The approach yields always a net with the behavior being the tightest overapproximation of the initial behavior. Additionally, it can handle proper partial orders that can arise from real applications.

To the best of our knowledge, the work in [6] was one of the first theoretical approaches to apply the theory of regions to mining. In this approach, however, only the synthesis (and not mining) of elementary transitions systems is considered, thus restricting the application to particular types of logs.

11 Conclusions

In this paper, the theory of regions has 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.

The theory presented in this paper has been implemented in a tool [1], and the experimental results reported are promising. More research is required to improve the speed of the synthesis and mining algorithms.

Appendix

Proof of the Main Results

Lemma 11.1. *Let* $\mathrm{TS} = (S, E, A, s_{in})$ *be a transition system, and* $\mathrm{PN} = (P, T, W, M_0)$ *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 that 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:

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

and the last inequality holds because lines 9-13 of the algorithm always guarantee $-\text{MAX-K-TOP}(r_i, e)\} \geq \Delta_{r_i}(e)$. But the inequality is in fact an equality: if $-\text{MAX-K-TOP}(r_i, e)\} > \Delta_{r_i}(e)$, then there exists a transition (s, e, s'') satisfying $r_i(s) = g, r_i(s'') = g + \Delta_{r_i}(e) < 0$, which will imply $r_i(s'') < 0$, a contradiction. 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. □

Proof of Theorem 4.1

Proof. Induction on the length of the traces in $L(\text{TS})$: in the case $|\sigma| = 1$, we have that if $s_{in} \in \text{ER}(e)$ then $M_0 \xrightarrow{e}$, because otherwise there exists a place $p_i \in {}^\bullet e$ such that $M_0[p_i] < W(p_i, e) = \text{MAX-K-TOP}(r_i, e)$, which implies $\text{TOP}(r_i, e)$ does not contain s_{in}, and therefore $s_{in} \notin \text{ER}(e)$, 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 11.1 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

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

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

Lemma 11.2. *Let* $\text{TS} = (S, E, A, s_{in})$ *and* $\text{TS}' = (S', E, A', s'_{in})$ *be such that there exists a simulation relation of* TS *by* TS' *with relation* π. *If* $r \in R_{\text{TS}'}$, *then* $\pi^{-1}(r) \in R_{\text{TS}}$, *and for every event* $e, \Delta_e(r)$ *is preserved in* $\pi^{-1}(r)$.

Proof. 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 $\pi^{-1}(r)$. □

Lemma 11.3. *Let* $\text{TS}_1 = (S_1, E_1, A_1, s_{in1})$ *and* $\text{TS}_2 = (S_2, E_2, A_2, s_{in2})$ *be two* TSs *such that* TS_2 *is deterministic, and* $L(\text{TS}_1) \subseteq L(\text{TS}_2)$. *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 $L(\text{TS}_1) \subseteq L(\text{TS}_2)$. 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 $s_1 \in S_1, \text{TS}_2$ 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 $L(\text{TS}_1) \subseteq L(\text{TS}_2)$. □

Proof of Theorem 4.2

Proof. By contradiction. Let $\text{PN}' = (P', T', W', M'_0)$ exist with the reachability graph $\text{TS}' = (S', E', A', s'_{in})$ such that $E' = T$, $L(\text{TS}) \subseteq L(\text{TS}')$, and $L(\text{PN}) \not\subseteq L(\text{TS}')$. The following facts can be observed:

- TS' and $RG(\text{PN})$ are deterministic because $E = E' = T$ and therefore they correspond to the reachability graph of Petri nets with a different label for each transition.[6]
- Since TS' is deterministic and $L(\text{TS}) \subseteq L(\text{TS}')$, there is a simulation π of TS by TS' (Lemma 11.3).
- $\forall r' \in R_{\text{TS}'}$, $r = \pi^{-1}(r') \in R_{\text{TS}}$, and events with constant gradient are the same in r' and r (Lemma 11.2).
- Each nonminimal region can be described as the union of disjoint minimal regions [11], and therefore we can focus only on minimal regions.
- Each minimal region $r \in R_{\text{TS}}$ is a region in $R_{RG(\text{PN})}$, since PN has been obtained with Algorithm 1 from the set of minimal regions in TS. Moreover, since $RG(\text{PN})$ is deterministic and $L(\text{TS}) \subseteq L(\text{PN})$ (Theorem 4.1), then there is a simulation of TS by $RG(\text{PN})$ (Lemma 11.3). Now using Lemma 11.2, together with the fact that r is a region both in R_{TS} and $R_{RG(\text{PN})}$, events with constant gradient in TS have also constant gradient in $RG(\text{PN})$.

Hence, the previous items show that for a region in TS' there is a corresponding region in $RG(\text{PN})$ 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 11.3). This contradicts the assumption that $L(\text{PN}) \not\subseteq L(\text{TS}')$. □

Proof of Theorem 5.1

Proof. Theorem 4.1 provides $L(\text{TS}) \subseteq L(\text{RG}(\text{PN}))$, and together with the fact that $\text{RG}(\text{PN})$ is deterministic (since $T = E$), Lemma 11.3 can be applied to prove the existence of a simulation of TS by $\text{RG}(\text{PN})$. 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 $\text{RG}(\text{PN})$ by TS (see Definition 2.2). For that purpose, assume that there exists a reachable state s_2 in $\text{RG}(\text{PN})$ 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

6. Here we abuse the notation used in the paper to incorporate transition labels when needed.

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 $\mathrm{RG}(\mathrm{PN})$ 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 $\mathrm{RG}(\mathrm{PN})$ after firing σ'. By the induction hypothesis, $\pi^{-1}(s_1) \in S$ is related to s_1, and according to our assumption, $\pi^{-1}(s_1) \notin \mathrm{ER}(e)$ (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 $q = \mathrm{TOP}(r_i, e)$. 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 11.1, 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. □

Proof of Theorem 6.2

Proof. The proof is based on the following facts:

1. All minimal regions are pre- or postregions of some event (Property 6.1). 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 nonconstant gradient. Given that $g = \lfloor (g_{min} + g_{max})/2 \rfloor$, there is always some transition $s_1 \xrightarrow{e} s_2$ such that $r(s_2) - r(s_1) = g_{max} > g$ and some transition $s_3 \xrightarrow{e} s_4$ 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. □

Proof of Theorem 7.1

Proof. $L(\mathrm{PN}) \subseteq L(\mathrm{PN}')$ holds because PN' is PN with one constraint less on the arc connecting p and event e. $L(\mathrm{PN}) \supseteq L(\mathrm{PN}')$ 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 $s \notin \mathrm{TOP}(r_i, e)$. But then provided that the enabling closure is preserved, event e cannot 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. □

ACKNOWLEDGMENTS

The authors would like to thank Eric Verbeek for helpful discussions and guidance in using ProM, and anonymous referees for their help in improving the paper. This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

REFERENCES

[1] Genet, http://www.lsi.upc.edu/~jcarmona/genet.html, 2009.
[2] A. Arnold, *Finite Transition Systems.* Prentice Hall, 1994.
[3] E. Badouel and P. Darondeau, "Theory of Regions," *Petri Nets,* W. Reisig and G. Rozenberg, eds., pp. 529-586, Springer, 1998.
[4] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Synthesis of Petri Nets from Finite Partial Languages," *Fundamenta Informaticae,* vol. 88, no. 4, pp. 437-468, 2008.
[5] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process Mining Based on Regions of Languages," *Proc. Fifth Int'l Conf. Business Process Management,* pp. 375-383, Sept. 2007.
[6] N. Busi and G.M. Pinna, "Characterizing Workflow Nets Using Regions," *Proc. Int'l Symp. Symbolic and Numeric Algorithms for Scientific Computing,* pp. 399-406, 2006.
[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," *Proc. Int'l Conf. Business Process Management (BPM),* M. Dumas, M. Reichert, and M.C. Shan, eds., pp. 358-373, 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," *Proc. 29th Int'l Conf. Applications and Theory of Petri Nets (PETRI NETS '08),* pp. 92-111, 2008.
[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 Trans. Computers,* vol. 47, no. 8, pp. 859-882, Aug. 1998.
[12] P. Darondeau, "Region Based Synthesis of P/T-Nets and its Potential Applications," *Proc. 21st Int'l Conf. Application and Theory of Petri Nets,* pp. 16-23, 2000.
[13] P. Darondeau, "Unbounded Petri Net Synthesis," *Lectures on Concurrency and Petri Nets,* J. Desel, W. Reisig, and G. Rozenberg, eds., pp. 413-438, Springer, 2003.
[14] P. Darondeau, "Distributed Implementations of Ramadge-Wonham Supervisory Control with Petri Nets," *Proc. 44th IEEE Conf. Decision and Control, 2005, and 2005 European Control Conference (CDC-ECC '05),* pp. 2107-2112, Dec. 2005.
[15] P. Darondeau Private Communication, 2008.
[16] J. Desel and W. Reisig, "The Synthesis Problem of Petri Nets," *Acta Informatica,* vol. 33, no. 4, pp. 297-315, 1996.
[17] D.L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits," ACM distinguished dissertations, MIT Press, 1989.
[18] A. Ehrenfeucht and G. Rozenberg, "Partial (Set) 2-Structures. Part I, II," *Acta Informatica,* vol. 27, pp. 315-368, 1990.
[19] P.W. Hoogers, H.C.M. Kleijn, and P.S. Thiagarajan, "A Trace Semantics for Petri Nets," *Information and Computation,* vol. 117, no. 1, pp. 98-114, 1995.
[20] R.M. Karp and R.E. Miller, "Parallel Program Schemata," *J. Computer and System Sciences,* vol. 3, pp. 147-195, 1969.
[21] N.A. Lynch and M.R. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," *Proc. Sixth Ann. ACM Symp. Principles of Distributed Computing,* pp. 137-151, Aug. 1987.
[22] R. Milner, *Communication and Concurrency.* Prentice-Hall, 1989.
[23] M. Mukund, "Petri Nets and Step Transition Systems," *Int'l J. Foundations of Computer Science,* vol. 3, no. 4, pp. 443-478, 1992.
[24] S.M. Nowick and D.L. Dill, "Automatic Synthesis of Locally-Clocked Asynchronous State Machines," *Proc. Int'l Conf. Computer-Aided Design (ICCAD),* pp. 318-321, Nov. 1991.
[25] C.A. Petri, "Kommunikation mit Automaten," PhD thesis, Bonn, Institut für Instrumentelle Mathematik (technical report Schriften des IIM Nr. 3), 1962.
[26] A. Rozinat and W.M.P. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems,* vol. 33, no. 1, pp. 64-95, 2008.
[27] M.H.T. Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, "Synchronizations in Team Automata for Groupware Systems," *Computer Supported Cooperative Work,* vol. 12, no. 1, pp. 21-69, 2003.

[28] 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.
[29] W.M.P. van der Aalst and C.W. Günther, "Finding Structure in Unstructured Processes: The Case for Process Mining," *Proc. Seventh Int'l Conf. Application of Concurrency to System Design (ACSD),* T. Basten, G. Juhás, and S.K. Shukla, eds., pp. 3-12, 2007.
[30] W.M.P. van der Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 9, pp. 1128-1142, Sept. 2004.
[31] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik, "Process Discovery Using Integer Linear Programming," *Proc. 29th Int'l Conf. Applications and Theory of Petri Nets (PETRI NETS '08),* pp. 368-387, 2008.
[32] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst, "The Prom framework: A New Era in Process Mining Tool Support," *Proc. Int'l Conf. Applications and Theory of Petri Nets (ICATPN),* G. Ciardo and P. Darondeau, eds., pp. 444-454, 2005.
[33] *Proc. 29th Int'l Conf. Applications and Theory of Petri Nets (PETRI NETS '08),* K.M. van Hee and R. Valk, eds., 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 Univ. of Technology, 2006.

Josep Carmona received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1999 and 2004, respectively. He is a lecturer in the Department of Software of the Universitat Politècnica de Catalunya, Barcelona, Spain. In 2003, he was a visiting scholar at the University of Leiden, The Netherlands. His research interests include formal methods and computer-aided design of VLSI systems with special emphasis on asynchronous circuits, concurrent systems, logic synthesis, and nanocomputing. He received the best paper award at the Ninth International Conference on Application of Concurrency to System Design (2009).

Jordi Cortadella (M'88) received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, in 1985 and 1987, respectively. He is a professor in the Department of Software of the same university and a chief scientist at Elastix Corporation. In 1988, he was a visiting scholar at the University of California, Berkeley. His research interests include formal methods and computer-aided design of VLSI systems with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis. He has coauthored numerous research papers and has been invited to present tutorials at various conferences. He has served on the technical committees of several international conferences in the field of design automation and concurrent systems. He received the best paper awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems (2004), the Design Automation Conference (2004), and the International Conference on Application of Concurrency to System Design (2009). In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya. He is a member of the IEEE.

Mike Kishinevsky received the MS and PhD degrees in CS from the Electrotechnical University of St. Petersburg. He leads a research group in front-end design at Strategic CAD Labs of Intel. Prior to joining Intel in 1998, he has been a research fellow at the Russian Academy of Science, a senior researcher at a start-up in asynchronous design (TRASSA), a visiting associate professor at the Technical University of Denmark, and a professor at the University of Aizu, Japan. He coauthored three books in asynchronous design and has published more than 70 journal and conference papers. He has served on the technical program committee at several conferences and workshops. He received the Semiconductor Research Corporation outstanding mentor award (2004) and the best paper awards at the Design Automation Conference (2004) and the International Conference on Application of Concurrency to System Design (2009). He is a senior member of the IEEE.