Incremental Process Mining

Marc Solé[1] and Josep Carmona[2]

[1] Computer Architecture Department, UPC `msole@ac.upc.edu`
[2] Software Department, UPC `jcarmona@lsi.upc.edu`

Abstract. The problem of synthesis of Petri nets from transition systems or languages has many applications, ranging from CAD for VLSI to medical applications, among others. The most common algorithms to accomplish this task are based on the theory of regions. However, one of the problems of such algorithms is its space requirements: for real-life or industrial instances, some of the region-based algorithms cannot handle in memory the internal representation of the input or the exploration lattice required. In this paper, the incremental derivation of a basis of regions and the later partitioned basis exploration is presented, which allows the splitting of large inputs.

1 Introduction

The introduction of the theory of regions [1] in the early nineties enabled a new area of research that strives for transforming language or state-based representations into event-based ones. This transformation, known as *synthesis*, was initially devoted to derive a Petri net whose reachability graph was bisimilar or isomorphic to the input transition system. A variant of this problem, known as *mining*, has weaker requirements: the language of the derived Petri net must be a superset (maybe proper) of the language of the input transition system [2]. The theory of this paper provides algorithms for mining.

Many research has been carried out since the introduction of regions, specially of theoretical nature, which has brought a better understanding of the theory [3–6], and has provided meaningful extensions to make it more general [7–10].

As a consequence of the aforementioned theoretical work, tools based on the theory of regions started to be available by the end of the nineties [11, 12] and still new ones are developed nowadays [9, 10]. These tools are the outcome of bridging the gap between the theory and practice, and many of them are used extensively in the academic domain, whereas few are used in industry. The reasons for the limited success of these tools in industry might be:

1. The algorithms involved are complex, i.e. in general polynomial with the size of the input [3], that might be prohibitive for large inputs, and the use of efficient data structures like BDDs or heuristics only alleviates the problem.
2. No high-level techniques are provided to cope with the inherent complexity of the problem.

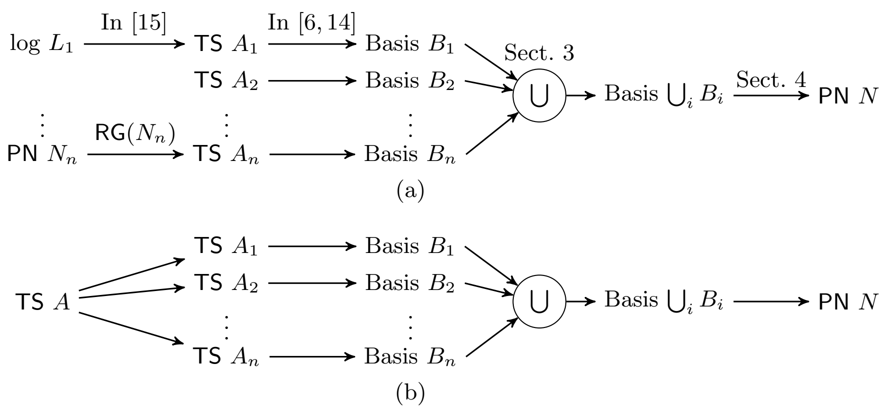


Fig. 1. Incremental Process Mining.

In this work we provide the theoretical basis for deriving high-level strategies that may allow to handle large specifications. More concretely, as space requirements are typically the bottleneck for some of the tools listed above, in this paper we present an incremental technique that allows splitting the input into several smaller parts. Moreover, we show how the theory of regions can be extended algorithmically to combine the regions of each part in order to derive the regions of the whole input.

The theory presented will be oriented for the problem of mining: given a set of objects describing behaviors (like a Petri net, a transition systems or an event log) $O_1, O_2, \ldots, O_n$, we want to obtain a Petri net N such that $\mathcal{L}(N) \supseteq \bigcup_{i=1}^{n} \mathcal{L}(O_i)$. The traditional way to solve this problem is to generate a transition system A_i for each O_i, join all these A_i to create a single transition system, and then apply a synthesis technique to derive a Petri net from a transition system [12, 13].

In this paper we explore a different approach. Instead of working with a monolithic transition system, we use the fact that a transition system can be represented by a basis of regions, such that any other region is a linear combination of the regions in the basis (a recent publication shows an efficient technique to accomplish this task [14]). Then, bases can be combined to obtain a region basis for the whole system, from which we can derive the Petri net N.

The general picture of the approach of this paper is outlined on Fig. 1(a). Some arcs are labeled with an indication of the section or the reference where the conversion/operation is explained. Basically the first step is to convert inputs that are not transition systems into a transition system, and then compute a region basis from which a PN can be generated. Another possible application is also shown in Fig. 1(b), where a large TS is split into smaller subsystems of manageable size, with the only restriction that all the subsystems must include the initial state and be connected.

1.1 Related work

In [16] an incremental approach was suggested based on the observation that any region of the union of two transition systems can be expressed as the union of regions of those systems. As in the approach presented in this paper, the approach in [16] trades space for time, since it must first compute all the regions and then obtain the minimal ones, a slower process than finding directly the minimal regions if the whole transition system fits into memory. The main drawback of this method is that the complete set of regions of each component transition system must be stored (either in memory or disk) in order to compute the set of regions of the union. In this work we propose a faster methodology based on the fact that the complete set of regions can be succintly represented by a basis of regions.

1.2 Organization

We start by giving the necessary background in Sect. 2. The process of combining a set of bases to produce a unique basis for the whole system is explained in Sect. 3. A description of the generation of a PN from a region basis is given in Sect. 4 and, finally, Sect. 5 concludes this paper.

2 Background

2.1 Finite Transition Systems and Petri Nets

Definition 1 (Transition system). *A* transition system *(TS) is a tuple* $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$*, where* S *is a set of* states*,* Σ *is an alphabet of* actions*,* $T \subseteq S \times \Sigma \times S$ *is a set of* (labelled) transitions*, and* $\mathsf{s}_0 \in S$ *is the* initial state*.*

We use $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$ as a shortcut for $(\mathsf{s}, \mathsf{e}, \mathsf{s}') \in T$, and we denote its transitive closure as $\xrightarrow{*}$. A state s' is said to be *reachable from state* s if $\mathsf{s} \xrightarrow{*} \mathsf{s}'$. We extend the notation to transition sequences, i.e., $\mathsf{s}_1 \xrightarrow{\sigma} \mathsf{s}_{n+1}$ if $\sigma = \mathsf{e}_1 \ldots \mathsf{e}_n$ and $(\mathsf{s}_i, \mathsf{e}_i, \mathsf{s}_{i+1}) \in T$. We denote $\#(\sigma, \mathsf{e})$ the number of times that event e occurs in σ. Let $A = \langle S, \Sigma, T, \mathsf{s}_0 \rangle$ be a TS. We consider connected TSs that satisfy the following axioms: i) S and Σ are finite sets, ii) every event has an occurrence and iii) every state is reachable from the initial state. The *language* of a TS A, $\mathcal{L}(A)$, is the set of transition sequences feasible from the initial state.

For two TSs A_1 and A_2, when $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, we will say that A_2 is an *over-approximation* of A_1.

Definition 2 (Union of TSs). *Given two TSs* $A_1 = \langle S_1, \Sigma_1, T_1, \mathsf{s}_0 \rangle$ *and* $A_1 = \langle S_2, \Sigma_2, T_2, \mathsf{s}_0 \rangle$*, the union of* A_1 *and* A_2 *is the TS* $A_1 \cup A_2 = \langle S_1 \cup S_2, \Sigma_1 \cup \Sigma_2, T_1 \cup T_2, \mathsf{s}_0 \rangle$*.*

Clearly, the TS $A_1 \cup A_2$ is an over-approximation of the TSs A_1 and A_2, i.e. $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1 \cup A_2)$.

Definition 3 (Petri net [17]). *A Petri net (PN) is a tuple (P, T, W, M_0) where the sets P and T represent disjoint 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 N, denoted $\mathsf{RG}(N)$, 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 $\mathcal{L}(N)$ as a shortcut for $\mathcal{L}(\mathsf{RG}(N))$.

2.2 Generalized Regions

The theory of regions [1, 5] provides a way to derive a Petri net from a transition system. Intuitively, a region corresponds to a place in the derived Petri net. In the initial definition, a region was defined as a subset of states of the transition system satisfying a homogeneous relation with respect to the set of events. Later extensions [7, 18, 10] generalize this definition to multisets, which is the notion used in this paper.

Definition 4 (Multiset, k-bounded Multiset, Subset). *Given a set S, a multiset r of S is a mapping $r : S \to \mathbb{Z}$. The number $r(s)$ is called the* multiplicity *of s in r. Multiset r is k-bounded if all its multiplicities are less or equal than k. Multiset r_1 is a* subset *of r_2 ($r_1 \subseteq r_2$) if $\forall s \in S : r_1(s) \leq r_2(s)$.*

We define the following operations on multisets:

Definition 5 (Multiset operations).

Maximum power	$\mathrm{pow}(r) = \max_{s \in S} r(s)$
Minimum power	$\mathrm{minp}(r) = \min_{s \in S} r(s)$
Scalar product	$(k \cdot r)(s) = k \cdot r(s)$, *for* $k \in \mathbb{Z}$
Scalar sum	$(r + k)(s) = r(s) + k$, *for* $k \in \mathbb{Z}$
Union	$(r_1 \cup r_2)(s) = \max(r_1(s), r_2(s))$
Sum	$(r_1 + r_2)(s) = r_1(s) + r_2(s)$
Subtraction	$(r_1 - r_2)(s) = r_1(s) - r_2(s)$

The operations described above have algebraic properties, e.g., $r + r = 2 \cdot r$ and $r_1 - k \cdot r_2 = r_1 + (-k) \cdot r_2$.

Definition 6 (Gradient). *Let $\langle S, \Sigma, T, s_0 \rangle$ be a TS. Given a multiset r and a transition $s \xrightarrow{e} s' \in T$, its* gradient *is defined as $\delta_r(s \xrightarrow{e} s') = r(s') - r(s)$. If all the transitions of an event $e \in \Sigma$ have the same gradient, we say that the event e has* constant gradient, *whose value is denoted as $\delta_r(e)$.*

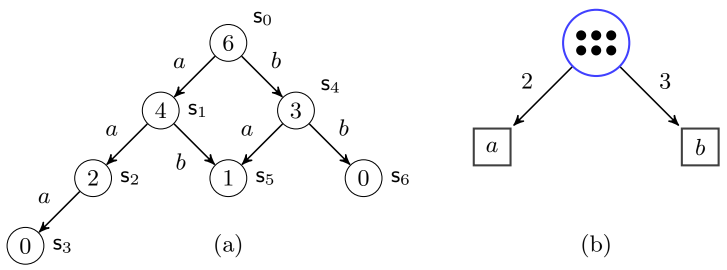


Fig. 2. (a) Region in a TS: $r(\mathsf{s}_0) = 6, r(\mathsf{s}_1) = 4, \ldots, r(\mathsf{s}_6) = 0$, (b) corresponding place in the Petri net.

Definition 7 (Region). *A* region *r is a multiset defined in a TS, in which all the events have constant gradient.*

Example 1. Fig. 2(a) shows a TS. The numbers within the states correspond to the multiplicity of the multiset r shown. Multiset r is a region because both events a and b have constant gradient, i.e. $\delta_r(\mathsf{a}) = -2$ and $\delta_r(\mathsf{b}) = -3$. There is a direct correspondence between regions and places of a PN. The gradient of the region describes the flow relation of the corresponding place, and the multiplicity of the initial state indicates the number of initial tokens [10]. Fig. 2(b) shows the place corresponding to the region shown in Fig. 2(a).

We say that region r is *normalized* if $\text{minp}(r) = 0$. Similarly, it is *non-negative* if $\text{minp}(r) \geq 0$. Any region r can become normalized by subtracting $\text{minp}(r)$ from the multiplicity of all the states.

Definition 8 (Normalization). *We denote by $\downarrow r$ the normalization of a region r, so that $\downarrow r = r - \text{minp}(r)$.*

It is useful to define a normalized version of the sum operation between regions, since it is closed in the class of normalized regions.

Definition 9 (Normalized sum). *Let r_1 and r_2 be normalized regions, we denote by $r_1 \oplus r_2$ their* normalized sum*, so that $r_1 \oplus r_2 = \downarrow(r_1 + r_2)$.*

Definition 10 (Gradient vector). *Let r be a region of a TS whose set of events is $\Sigma = \{\mathsf{e}_1, \mathsf{e}_2, \ldots, \mathsf{e}_n\}$. The* gradient vector of r*, denoted as $\Delta(r)$, is the vector of the event gradients, i.e. $\Delta(r) = (\delta_r(\mathsf{e}_1), \delta_r(\mathsf{e}_2), \ldots, \delta_r(\mathsf{e}_n))$.*

Proposition 1. *Gradient vectors have the following properties:*

$$\Delta(r_1 + r_2) = \Delta(r_1) + \Delta(r_2) \qquad \Delta(k \cdot r) = k \cdot \Delta(r)$$
$$\Delta(r + k) = \Delta(r) \qquad \Delta(r_1 - r_2) = \Delta(r_1) - \Delta(r_2)$$
$$\Delta(r_1 \oplus r_2) = \Delta(r_1) + \Delta(r_2)$$

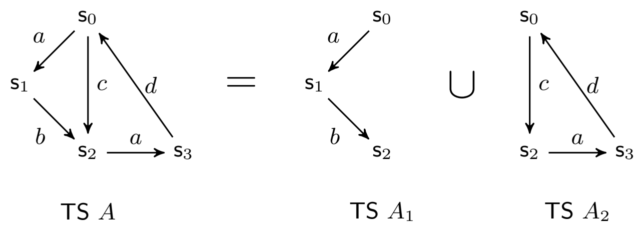


Fig. 3. TS A is split into two subsystems A_1 and A_2.

Regions can be partitioned into classes using their gradient vectors.

Definition 11 (Canonical region). *Two regions r_1 and r_2 are said to be equivalent if their gradient is the same,* i.e. $r_1 \equiv r_2 \Leftrightarrow \Delta(r_1) = \Delta(r_2)$. *Given a region r, the equivalence class of r, is defined as $[r] = \{r_i | \ r_i \equiv r\}$. A canonical region is the normalized region of an equivalence class,* i.e. $\downarrow r$.

Example of canonical regions are provided in Fig. 4, where two TSs are shown in which some regions have been shadowed. For instance, the canonical region $r_0 = \{s_1, s_2\}$ has gradient vector $\Delta(r_0) = (1, 0)$. A PN built from the set of minimal canonical regions has the same language as a PN built using all the regions [5], thus it yields the smallest overapproximation with respect to the language of the TS [10].

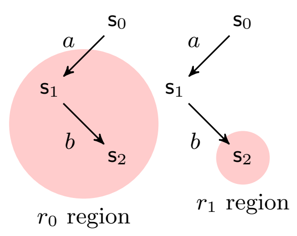
Definition 12 (Subregion, Empty region, Minimal canonical region). *r_1 is a* subregion *of r_2, denoted as $r_1 \sqsubseteq r_2$, if, for any state s, $\downarrow r_1(s) \leq \downarrow r_2(s)$. We denote by $\emptyset$ the region in which all states have zero multiplicity. A minimal canonical region r satisfies that for any other region r', if $r' \sqsubseteq r$ then $r' \equiv \emptyset$.*

3 Combining region bases

In this section we detail how region bases of different TSs can be joined, yielding a region basis of their union. We will illustrate the theory with the running example shown in Fig. 3. In this case, we assume A is a very large TS that cannot be easily handled, hence it is split into two smaller subsystems, namely A_1 and A_2, so that $A = A_1 \cup A_2$.

3.1 Basis of regions

Definition 13 (Region basis). *Given a TS, a region basis $B = \{r_1, r_2, \ldots, r_n\}$ is a minimal subset of the canonical regions of TS such that any region r can be expressed as a linear combination of B (*i.e. *$r = \sum_{i=1}^{n} c_i \cdot r_i$, with $c_i \in \mathbb{Q}$, $r_i \in B$).*



Event	$\Delta(r_0)$	$\Delta(r_1)$
a	1	0
b	0	1

Event	$\Delta(q_0)$	$\Delta(q_1)$
a	-1	-1
c	1	0
d	0	1

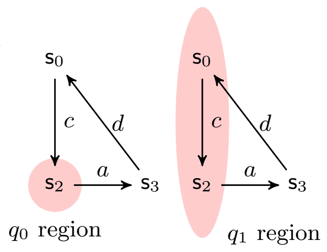


Fig. 4. Region bases for A_1 and A_2.

The set of canonical regions of a TS, together with the normalized sum operation, forms a free Abelian group [18]. Consequently, there exists a *basis* (*i.e.* subset of the group) such that every element in the group can be rewritten as a unique linear combination of the basis elements.

Region bases are interesting because, as the following theorem states, their size is usually small.

Theorem 1 ([18]). *Let $\langle S, \Sigma, T, s_0 \rangle$ be a TS. The size of the region basis is less or equal to* $\min(|\Sigma|, |S| - 1)$.

3.2 Region compatibility

Given two systems described by their region basis, we want to obtain the region basis of their union TS. The work more closely related is [16], in which it is described how the set of regions in the joined system can be obtained from the regions of the component systems. This is achieved by introducing the concept of compatible (standard) regions. In this section we first review and extend this concept of compatibility to generalized regions.

Definition 14 (Compatible TSs). *Two TSs A_1 and A_2 are compatible if they have the same initial state.*

Def. 14 is more general than the one in [16], where the number of shared states is restricted to one (the initial state). For instance, systems A_1 and A_2 of Fig. 3 are compatible according to this definition, but not using the definition in [16], since they share states s_0 and s_2.

Definition 15 (Compatible regions, offset). *Two regions r_1 and r_2 from two compatible TSs $A_1 = \langle S_1, \Sigma_1, T_1, s_0 \rangle$ and $A_2 = \langle S_2, \Sigma_2, T_2, s_0 \rangle$ are said to be* compatible *if:*

– $\forall e \in \Sigma_1 \cap \Sigma_2, \delta_{r_1}(e) = \delta_{r_2}(e)$, i.e. *they have the same gradient for all common events, and,*

– $\exists k \in \mathbb{Z} : \forall \mathsf{s} \in S_1 \cap S_2, r_2(\mathsf{s}) - r_1(\mathsf{s}) = k$, i.e. *the difference in multiplicity of each shared state is equal to a constant, that we call the* offset *between* r_1 *and* r_2*, denoted as* $\mathrm{off}(r_1, r_2)$.

Two compatible regions are said to be directly compatible *if* $\mathrm{off}(r_1, r_2) = 0$*, a fact that we denote as* $r_1 \leftrightarrow r_2$*. Conversely, if* $\mathrm{off}(r_1, r_2) \neq 0$*, we say that the regions are* indirectly compatible *and we use the following notation* $r_1 \leftrightsquigarrow r_2$*.*

An immediate consequence of Def. 15 is that, if there is only a single shared state, then any two regions with the same gradient for all common events are compatible. This is, for instance, the case when TSs represent execution trees and only the initial state is shared among them.

Two compatible regions can be made directly compatible by adding the offset to one of them.

Definition 16 (Directly compatible region). *Given two compatible regions* r_1 *and* r_2 *with* $\mathrm{off}(r_1, r_2) > 0$*, the directly compatible region of* r_1 *with respect to* r_2 *is* $r_1{\uparrow^{r_2}} = r_1 + \mathrm{off}(r_1, r_2)$*.*

Definition 17 (Union of compatible regions). *Given two compatible regions* r_1 *and* r_2*, defined over sets of states* S_1 *and* S_2*, respectively, with* $\mathrm{off}(r_1, r_2) > 0$*, their union, denoted* $r_1 \sqcup r_2$*, is*

$$(r_1 \sqcup r_2)(\mathsf{s}) = \begin{cases} (r_1{\uparrow^{r_2}})(\mathsf{s}) & \text{if } \mathsf{s} \in S_1 \\ r_2(\mathsf{s}) & \text{otherwise} \end{cases}$$

Proposition 2 (Union of compatible regions is a region of union system). *Given two compatible regions* r_1 *and* r_2 *of two compatible* TS*s* A_1 *and* A_2*,* $r_1 \sqcup r_2$ *is a region of* $A_1 \cup A_2$*. For each shared event, its gradient is equal to the gradient in* r_1 *or* r_2*, which are equal. For non-shared events of* A_1 *(*A_2*), their gradient is the gradient in* A_1 *(*A_2*).*

Proof. Assume $\mathrm{off}(r_1, r_2) \geq 0$. Region $r = r_1 \sqcup r_2$, where $r_1{\uparrow^{r_2}} = r_1 + \mathrm{off}(r_1, r_2)$. The latter entails that, in a shared state s, the multiplicity is the same for $r_1{\uparrow^{r_2}}$ and r_2, which is the multiplicity assigned to r. For non-shared states, on the other hand, the multiplicity assigned in r is the multiplicity of the region whose TS contains the state. Thus any arc (either going from a shared to non-shared state or any other combination) has a constant gradient, equal to the gradient of that event in the corresponding region. Result transfers to r_1 because $\Delta(r_1) = \Delta(r_1{\uparrow^{r_2}})$. □

Example 2. In Fig. 5 we show one region of each subsystem of our running example. They are compatible, since the gradient of the single shared event a is the same in both regions. More specifically, they are indirectly compatible, since their offset is different than 0.

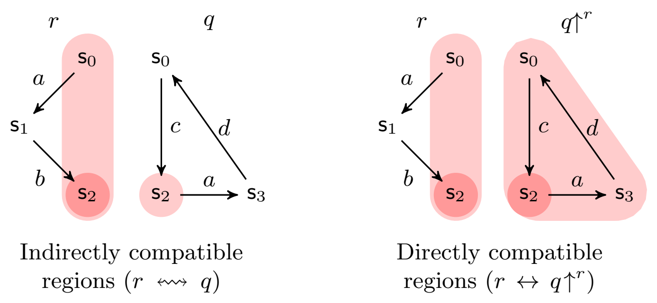

r
q
r
$q{\uparrow}^{r}$
s_0
s_1
s_2
s_3
a
b
c
d
Indirectly compatible regions ($r \leftrightsquigarrow q$)
Directly compatible regions ($r \leftrightarrow q{\uparrow}^{r}$)

Fig. 5. Region r of A_1 and q of A_2, are indirectly compatible.

Property 1. Given two compatible non-negative regions r_1 and r_2. If they are directly compatible then $r_1 \sqcup r_2$ is a normalized region, if and only if, one of them is normalized. If they are not directly compatible, but both of them are normalized, then again $r_1 \sqcup r_2$ is a normalized region.

Proof. If one of them is normalized and they are directly compatible no modification of multiplicities is performed, so the state with 0 multiplicity will remain untouched and since regions are non-negative, then $\text{minp}(r_1 \sqcup r_2) = 0$. Conversely, if $r_1 \sqcup r_2$ is a normalized region and $r_1 \leftrightarrow r_2$, then all multiplicities of either r_1 or r_2 are greater or equal to 0, and since there is at least one 0 multiplicity, at least one of them must be normalized. If both are normalized but are not directly compatible, only one of them modifies its multiplicities and we end up in the same situation. □

3.3 Incremental algorithm for obtaining a basis

Given two TSs, A_1 and A_2. Let $\{r_1, \ldots, r_n\}$ be the region basis of A_1 and $\{q_1, \ldots, q_m\}$ the basis of A_2. The set of all regions of the union system $A_1 \cup A_2$ whose language satisfies $\mathcal{L}(A_1 \cup A_2) \supseteq \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ is obtained by finding all non-trivial solutions for x_i variables that satisfy:

$$\forall \mathsf{e}_i \in \Sigma_1 \cap \Sigma_2 : \sum_{1 \le j \le n} \delta_{r_j}(\mathsf{e}_i) \cdot x_j = \sum_{1 \le k \le m} \delta_{q_k}(\mathsf{e}_i) \cdot x_{n+k}$$

i.e. for all common events their gradient must be the same on both systems.

This system of equations can be rewritten in matrix form as $M \cdot \boldsymbol{x} = \mathbf{0}$, where $\boldsymbol{x}$ is the column vector of variables x_i and M is the following matrix:

$$M = \begin{pmatrix} \delta_{r_1}(\mathsf{e}_1) \ldots \delta_{r_n}(\mathsf{e}_1) & -\delta_{q_1}(\mathsf{e}_1) \ldots -\delta_{q_m}(\mathsf{e}_1) \\ \delta_{r_1}(\mathsf{e}_2) \ldots \delta_{r_n}(\mathsf{e}_2) & -\delta_{q_1}(\mathsf{e}_2) \ldots -\delta_{q_m}(\mathsf{e}_2) \\ \vdots & \vdots \\ \delta_{r_1}(\mathsf{e}_c) \ldots \delta_{r_n}(\mathsf{e}_c) & -\delta_{q_1}(\mathsf{e}_c) \ldots -\delta_{q_m}(\mathsf{e}_c) \end{pmatrix}$$

where c is the number of common events in the system, *i.e.* $c = |\Sigma_1 \cap \Sigma_2|$. We call this matrix the *gradient compatibility matrix* between A_1 and A_2.

Compatibility of regions demands not only the gradients of common events to be the same, but also that the offset is the same for all shared states. To enforce such condition, assume that we shift all the regions in the bases $\{r_1, \ldots, r_n\}$ and $\{q_1, \ldots, q_m\}$ so that for all r_i and q_j we have that $r_i(\mathsf{s}_0) = q_j(\mathsf{s}_0) = 0$. Now any region obtained by combining the regions in the bases will have a 0 multiplicity in the initial state. Thus, if the offset is the same in all shared states, their multiplicity must coincide in all remaining shared states.

If there is a path between s_0 and shared state s in which the same events (and the same number of times but no matter in which order) are fired in both TSs A_1 and A_2, then the condition is automatically satisfied. Only in the case where all paths are different, we say that shared state s is *in conflict*, and then we must explicitly enforce the equality of the multiplicity of s in both systems.

This condition can be expressed in matrix form using a row for each shared state in conflict (different than s_0 since this state is never in conflict). For a shared state s in conflict, its corresponding row will be of the form $(r_1(\mathsf{s}) \ldots r_n(\mathsf{s}) - q_1(\mathsf{s}) \ldots q_m(\mathsf{s}))$, where all regions r_i and q_j have been shifted, as said before, so that $r_i(\mathsf{s}_0) = q_j(\mathsf{s}_0) = 0$. Let us name as C the matrix containing such rows, we will call it the *shared state conflict matrix* between A_1 and A_2. Now since the multiplicity of all these states must be the same, their subtraction must be 0. Thus, $C \cdot \boldsymbol{x} = \mathbf{0}$.

Theorem 2. *Let A_1 and A_2 be two compatible TSs with region bases $\{r_1, \ldots, r_n\}$ and $\{q_1, \ldots, q_m\}$ respectively. Let M be their gradient compatibility matrix, C be the shared state conflict matrix, and $\boldsymbol{x}$ the column vector of variables x_1 to x_{n+m}. Consider an assignment to the variables in $\boldsymbol{x}$ such that $\binom{M}{C} \cdot \boldsymbol{x} = \mathbf{0}$ and some $x_i \neq 0$. This assignment identifies a region $r \sqcup q = \sum_{1 \leq i \leq n} r_i x_i \sqcup \sum_{1 \leq j \leq m} q_j x_{n+j}$ in $A_1 \cup A_2$.*

Proof. A non-trivial solution $\boldsymbol{x}$ defines two compatible regions, namely $r = \sum_{1 \leq i \leq n} r_i x_i$ and $q = \sum_{1 \leq j \leq m} q_j x_{n+j}$, in A_1 and A_2 respectively. These two regions are compatible according to Definition 15 if $M \cdot \boldsymbol{x} = \mathbf{0}$, because for common events the gradient is the same and the offset is the same in all shared states if $C \cdot \boldsymbol{x} = 0$. By Proposition 2, $r \sqcup q$ is a region of $A_1 \cup A_2$. □

So the problem reduces to finding the solutions to a homogeneous linear system. Note that the system does not require to have solutions in the integer domain. In fact all the solutions are in $\mathbb{Q}$, since all the gradients are integers. Homogeneous linear systems have one trivial solution (*i.e.* $\mathbf{0}$) and infinite non-trivial solutions. Let $\boldsymbol{y_i}$ be all the non-redundant solution vectors, then any possible solution of the system can be obtained by linear combination of these solution vectors, since adding or subtracting $\mathbf{0}$ from $\mathbf{0}$ does not change its value. These $\boldsymbol{y_i}$ are a basis of the nullspace of $\binom{M}{C}$, and any solution $\boldsymbol{x}$ can be written as a unique linear combination $\boldsymbol{x} = \sum_i \lambda_i \boldsymbol{y_i}$, with $\lambda_i \in \mathbb{Q}$.

There are several well-known methods to obtain a basis for the nullspace [19], one of the easiest is to put the matrix in reduced row echelon form, determine the

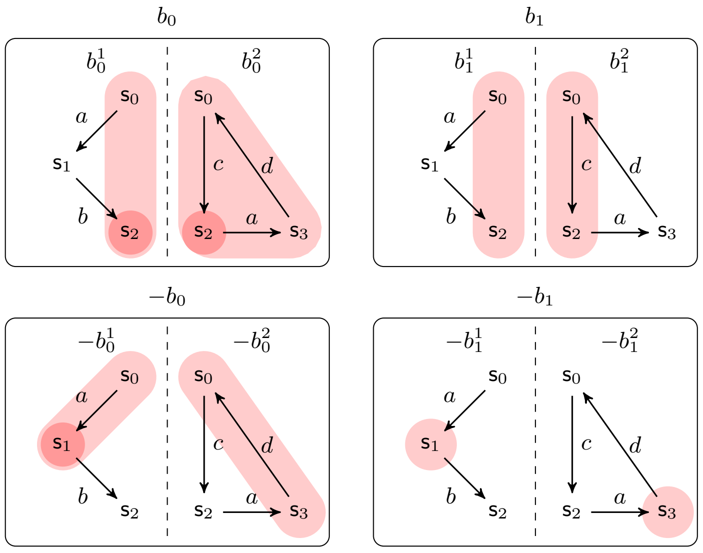


Fig. 6. Region basis $\{b_0, b_1\}$ (and their coregions $\{-b_0, -b_1\}$) for system $A_1 \cup A_2$. Regions are partitioned so that $b_0 = b_0^1 \cup b_0^2$, where b_0^i is the part of b_0 in A_i.

free variables, and then, for each free variable x_i, derive a vector of the basis by assigning 1 to x_i and 0 to the rest of free variables. The $\boldsymbol{y_i}$ columns correspond to the combination of regions of both system that have the same gradient, and the resulting combined region is a region basis for the union TS.

Example 3. We will compute the region basis of the union of TSs A_1 and A_2 shown in Fig. 3. Two possible region basis for these systems are $\{r_0, r_1\}$ and $\{q_0, q_1\}$, show in Fig. 4. The matrix M in this case is

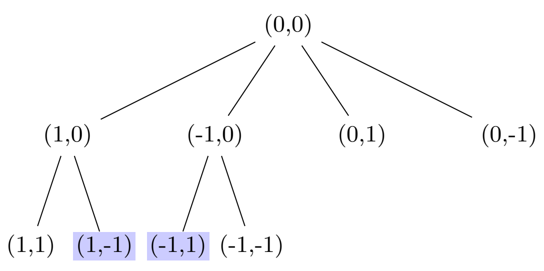
$$M = \begin{pmatrix} 1 \; 0 \; 1 \; 1 \end{pmatrix}$$

where columns (from left to right) correspond to regions r_0, r_1, q_0, q_1 and there is only a single row that corresponds to the only shared event between A_1 and A_2, namely event a.

The set of shared states is $\{\mathsf{s}_0, \mathsf{s}_2\}$, thus the shared state conflict matrix C has only one row because only state s_2 is reachable from s_0 by firing a different multiset of events. Consequently $C = \big(r_0(\mathsf{s}_2) \; r_1(\mathsf{s}_2) \; {-q_0}(\mathsf{s}_2) \; {-q_1}(\mathsf{s}_2)\big)$. With matrices M and C we can now build

$$\begin{pmatrix} M \\ C \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & -1 & 0 \end{pmatrix} \xrightarrow[\text{echelon form}]{\text{reduced row}} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -2 & -1 \end{pmatrix}$$

which is an indeterminate system with 2 degrees of freedom. We can write $x_1 = 2x_2 + x_3$ and $x_0 = -x_2 - x_3$, so by changing the values of variables x_2 and x_3



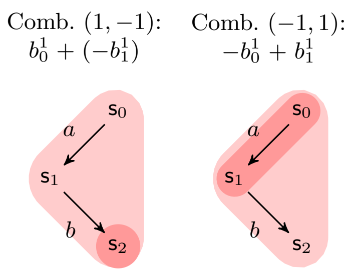

Comb. $(1,-1)$: $b_0^1 + (-b_1^1)$
Comb. $(-1,1)$: $-b_0^1 + b_1^1$
s_0
a
s_1
b
s_2
s_0
a
s_1
b
s_2

Fig. 7. Exploration of the region space. Each node represents a combination of the basis $\{b_0, b_1\}$ that will be explored. The combinations shaded in blue are the ones for which A_1 yields a non-normalized region (shown on right), thus only these combinations would be checked in A_2.

we can generate all the possible solutions. Given these parameters the region solution will be $((-x_2 - x_3) \cdot r_0 + (2x_2 + x_3) \cdot r_1) \sqcup (x_2 \cdot q_0 + x_3 \cdot q_1)$. Using the parameter values $(1, 0)$, and $(0, 1)$ for vector (x_2, x_3) we do obtain the following regions in the joined system: $b_0 = (-r_0 + 2r_1) \sqcup q_0$ and $b_1 = (-r_0 + r_1) \sqcup q_1$. The set $\{b_0, b_1\}$ forms a region basis for the $A_1 \cup A_2$ system (see Fig. 6).

4 Generating a PN from a basis

In [14] an algorithm was presented that allows finding minimal regions by careful exploration of the region space defined by the region basis. The fundamental idea is that the regions in the basis are initially assumed to be minimal, and then combinations of these regions can only yield a minimal region if the resulting region is non-normalized, since otherwise the region is a superregion of any of its component regions.

However this approach cannot be directly used if the number of states in the monolithic TS is too high to easily perform region operations in memory. The alternative we propose in this paper is to partition the region operations into the different component TSs, so that each time only the information of one of the systems is accessed.

To achieve this partitioning, consider the region basis $\{b_0, \ldots, b_n\}$, we denote by b_i^j the part of region b_i that belongs to subsystem j. All the regions in the basis are assumed to be normalized, but the different b_i^j could be non-normalized, since they are defined so that, given i, all b_i^j are directly compatible (cf. Def. 15). For instance region b_0 in Fig. 6 is normalized, however b_0^2 it is not.

Consequently a combination of the basis $\sum_i c_i \cdot b_i$ yields a non-normalized region only if, for all subsystem j, $\sum_i c_i \cdot b_i^j$ is a non-normalized region (by

Partial order $\subseteq$ in A_1

$$\begin{tikzcd}
& (0,0) \arrow[dl] \arrow[dr] & \\
(1,-1) \arrow[d] & & (0,-1) \arrow[d] \\
(0,1) \arrow[d] & & (-1,1) \arrow[d] \\
(1,0) & & (-1,0)
\end{tikzcd}$$

Partial order $\subseteq$ in A_2

$$\begin{tikzcd}
& (0,0) \arrow[dl] \arrow[dr] & \\
(-1,1) \arrow[dd] \arrow[dr] & & (0,-1) \arrow[dl] \arrow[dd] \\
& (-1,0) \arrow[dd] & \\
(0,1) \arrow[dr] & & (1,-1) \arrow[dl] \\
& (1,0) &
\end{tikzcd}$$

Fig. 8. Partial orders between combinations of regions in each subsystem, according to the subset relation on multisets ($\subseteq$).

Property 1)[3]. Thus, the strategy would be to test the basis combinations in the first subsystem, keeping only the combinations producing non-normalized regions. Then only these combinations will be tested in the second subsystem, discarding the ones that yield a normalized region, and the process will continue until all subsystems have been checked.

Finally, with only the surviving combinations, the subregion test will be performed to guarantee that only the minimal regions are found. Again this test can be distributed among the subsystems, since $\sum_i c_i \cdot b_i \subseteq \sum_i c'_i \cdot b_i$ if, and only if, for all subsystem j, $\sum_i c_i \cdot b_i^j \subseteq \sum_i c'_i \cdot b_i^j$.

We will illustrate all this process using our running example. In Fig. 7 we can see a tree of combinations of the $\{b_0, b_1\}$ basis. Each node is a tuple (c_0, c_1) describing the coefficients used to obtain a region r as $c_0 \cdot b_0 + c_1 \cdot b_1$. The second level of the tree corresponds to the regions in the basis and their coregions (as shown in Fig. 6). To bound the search space, assume we arbitrarily fix that the coefficients are only allowed to take values in the set $\{-1, 0, 1\}$.

From the four possible combinations in the third level, two of them yield already normalized regions in A_1, namely combinations $(1,1)$ and $(-1,-1)$. On the other hand, combinations $(1,-1)$ and $(-1,1)$ correspond to non-normalized regions in A_1, as shown in the figure. Consequently only these two combinations will be checked in subsystem A_2. In this case, both regions, namely, $b_0^2 + (-b_1^2)$ and $-b_0^2 + b_1^2$, are also non-normalized in A_2. Thus these combinations correspond to non-normalized regions in $A_1 \cup A_2$, which means that they might be minimal regions (once normalized).

[3] Since Property 1 only holds for non-negative regions, negative c_i coefficients are treated as markers for summing the normalized coregions of b_i. For instance $2b_1 - 3b_2$ will be actually computed as $2 \cdot b_1 + 3 \cdot {\downarrow}(-b_2)$. This way all regions are always non-negative and Property 1 can be safely used.

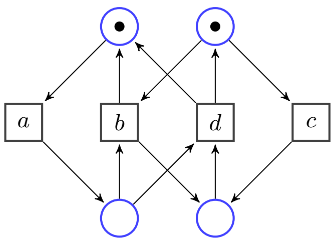


Fig. 9. Mined PN.

At this point we must check for minimal regions. The list of candidates includes all the regions in the basis (and their coregions), that is all the combinations in the second level, as well as combinations $(1, -1)$ and $(-1, 1)$ that have been found during the exploration. To check for minimality, we create a partial order among all these combinations in each subsystem (see Fig. 8). A region is not minimal if there is a combination, different than $(0, 0)$, that appears before it in all the partial orders. In our example, combinations $(1, 0)$ and $(-1, 0)$ are not minimal. Conversely, for instance $(-1, 1)$ is minimal because, although combination $(0, -1)$ precedes it in A_1 (i.e. $-b_1^1 \subseteq -b_0^1 + b_1^1$), it is not longer true in A_2 (i.e. $-b_1^2 \nsubseteq -b_0^2 + b_1^2$).

With the minimal regions found we build the mined PN of Fig. 9.

For a set of subsystems $A_1, \ldots, A_n$ the algorithm can be summarized as follows:

- Take the first subsystem (A_1) and explore region space until either combinations are exhausted (according to the user defined bounds on the coefficients) or the border of non-normalized regions is found.
- Pass to next subsystem the set of combinations of non-normalized regions and check for normality in this other subsystem.
  - If region is normalized, then discard, but mark sibling combinations to be explored in the current subsystem (since all sibling combinations in the first subsystem will remain to be non-normalized).
  - On the other hand, if region is non-normalized and we are not in the last subsystem, then add it to the list of regions that conform the border of non-normalized regions that will be passed to next subsystem. If we are already in the last subsystem, then add the combination to the list of candidate minimal regions, normalize it, and then check the normalization status in all subsystems. Sibling combinations are scheduled to be explored in the first subsystem whose subpart of the region is normalized.

5 Conclusions

In this paper we have extended the theory developed in [14] to devise an incremental algorithm for process mining. Given that space requirements are often the real bottleneck of some of the region-based techniques in the literature, methods like the one presented in this paper might represent a crucial step into making the theory applicable in industrial scenarios.

References

1. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica **27** (1990) 315–368
2. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142
3. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. Lecture Notes in Computer Science **915** (1995) 364–383
4. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general petri nets. Theor. Comput. Sci. **153**(1&2) (1996) 129–170
5. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4) (1996) 297–315
6. Badouel, E., Darondeau, P.: Theory of regions. In Reisig, W., Rozenberg, G., eds.: Petri Nets. Volume 1491 of LNCS., Springer (1998) 529–586
7. Mukund, M.: Petri nets and step transition systems. Int. Journal of Foundations of Computer Science **3**(4) (1992) 443–478
8. Darondeau, P.: Deriving unbounded petri nets from formal languages. In Sangiorgi, D., de Simone, R., eds.: CONCUR. Volume 1466 of Lecture Notes in Computer Science., Springer (1998) 533–548
9. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of petri nets from finite partial languages. Fundam. Inform. **88**(4) (2008) 437–468
10. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Transactions on Computers **59**(3) (2009)
11. Caillaud, B.: Synet : A synthesizer of distributable bounded Petri-nets from finite automata. http://www.irisa.fr/s4/tools/synet/ (2002)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Trans. on Computers **47**(8) (1998) 859–882
13. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. In: Application and Theory of Petri Nets and Other Models of Concurrency. (2008)
14. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Application and Theory of Petri Nets and other Models of Concurrency. LNCS, Springer (2010) 226–245
15. van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Günther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (2009)
16. Dongen, B.F.V., Busi, N., Pinna, G.M., van der Aalst, W.: An iterative algorithm for applying the theory of regions in process mining. In: Proceedings of the Workshop on Formal Approaches to Business Processes and Web Services (FABPWS'07). (2007) 36–55
17. Murata, T.: Petri Nets: Properties, analysis and applications. Proceedings of the IEEE (April 1989) 541–580
18. Bernardinello, L., Michelis, G.D., Petruni, K., Vigna, S.: On the synchronic structure of transition systems. In Desel, J., ed.: Structures in Concurrency Theory, Proc. of the Int. Workshop on Structures in Concurrency Theory. (1995) 69–84

19. Kalman, D.: Basic null space calculations. The College Mathematics Journal **15**(1) (1984) 42–47