Process Mining from a Basis of State Regions

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. A central problem in the area of Process Mining is to obtain a formal model that represents selected behavior of a system. The theory of regions has been applied to address this problem, enabling the derivation of a Petri net whose language includes a set of traces. However, when dealing with real-life systems, the available tool support for performing such task is unsatisfactory, due to the complex algorithms that are required. In this paper, the theory of regions is revisited to devise a novel technique that explores the space of regions by combining the elements of a region basis. Due to its light space requirements, the approach can represent an important step for bridging the gap between the theory of regions and its industrial application. Experimental results improve in orders of magnitude state-of-the-art tools for the same task.

1 Introduction

Nowadays the formal reasoning of a system is sometimes restricted by the difficulty of having a formal model that describes its behavior. This problem may appear at several stages of the life cycle: design, verification, and optimization. Aware of the problem, some companies have started to incorporate tools to discover formal models from executions or simulations of a system. This was the driving force that originated the area of *Process Mining*, where different subproblems are tackled, being *Process discovery* (denoted as *mining*), *Process Conformance* and *Process Extension* the three main directions of the area. In mining, the goal is to obtain a formal model (e.g., a Petri net) that includes the behavior of a system. In this work we present a novel strategy for this problem.

The *synthesis problem* [1] is related to mining: it consists in building a Petri net that has a behavior equivalent to a given transition system. The problem was first addressed by Ehrenfeucht and Rozenberg [2] introducing *regions* to model the sets of states that characterize marked places. In the area of synthesis, some techniques have been proposed to take the theory of regions in to practice. In [3] polynomial algorithms for the synthesis of bounded nets were presented. These algorithms have been recently adapted for the problem of process mining in [4]. In [5], the theory of regions was applied for the synthesis of safe Petri nets with bisimilar behavior. Recently, the theory from [5] has been extended to bounded Petri nets [6].

Mining differs from synthesis in the knowledge assumption: while in synthesis one assumes a complete description of the system, only a partial description

of the system is assumed in mining. However, synthesis can be adapted for mining in two ways: either the initial set of traces (called *log*) is encoded as a transition system (introducing state information, as described in [7]) and state-based methods for mining [8] are applied, or language-based methods are used directly on the log [4, 9]. In this paper we follow the first approach.

Due to its complexity, the theory of regions might become impractical for large inputs. In this paper, we present methods to alleviate significantly the complexity of the region-based approach. The main idea behind the strategy presented in this paper is based on the observation that the set of regions necessary for deriving a Petri net might be obtained by linear combinations of a small set of regions, i.e., from a *basis* of regions. This technique deviates from previous state-based methods for computing regions [6, 8], where the full lattice of multisets of states was explored to find the regions. The main contributions of this paper are:

- *Methods to efficiently compute a basis of regions*, based on the isomorphism between the structural and state-based representation of regions.
- *Heuristics to explore the region space*, that might be guided by meaningful population of the derived Petri net (desired bound and/or arcs) together with elaborated factors to avoid complex combinations of regions in the basis.
- *Enhancement of the technique for languages*, i.e., when the input transition system is derived from a language, the obtention of a basis and the corresponding isomorphism is shown to be simplified.
- The theory of this paper has been implemented in a tool [10]. The experimental results reported demonstrate the capacity of handling systems for which related approaches fail. Moreover, for well-known benchmarks, the quality of the derived results is shown to be similar to the one obtained for related approaches.

Organization We start by giving the necessary background in Sect. 2. Methods to compute a region basis are presented in Sect. 3, and Sect. 4 provides a strategy to explore the space of regions from a region basis to derive a Petri net. The techniques of this paper are evaluated in Sect. 5. A short description on the relations between the contributions of this paper and previous work is presented in Sect. 6. Finally, Sect. 7 concludes.

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 $s \xrightarrow{e} s'$ as a shortcut for $(s, e, s') \in A$, and we denote its transitive closure as $\xrightarrow{*}$. A state s' is said to be *reachable from state* s if $s \xrightarrow{*} s'$. We

extend the notation to transition sequences, i.e., $s_1 \xrightarrow{\sigma} s_{n+1}$ if $\sigma = e_1 \dots e_n$ and $(s_i, e_i, s_{i+1}) \in A$. We denote $\#(\sigma, e)$ 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 traces feasible from the initial state.

Definition 2 (Petri net [11]). *A Petri net (PN) is a tuple (P, T, W, M_0) where the sets P and T represent finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.*

A transition $t \in T$ is *enabled* in a marking M if $\forall p \in P : M[p] \geq W(p, t)$. Firing an enabled transition t in a marking M leads to the marking M' defined by $M'[p] = M[p] - W(p, t) + W(t, p)$, for $p \in P$, and is denoted by $M \xrightarrow{t} M'$. The set of all markings reachable from the initial marking M_0 is called its Reachability Set. The *Reachability Graph* of 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 [2, 1] 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 an homogeneous relation with respect to the set of events. Later extensions [12, 13, 6] generalize this definition to multisets, which is the notion used in this paper.

Definition 3 (Multiset, k-bounded Multiset, Subset). *Given a set S, a multiset r of S is a mapping $r : S \to \mathbb{Q}$. 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 4 (Multiset operations).

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

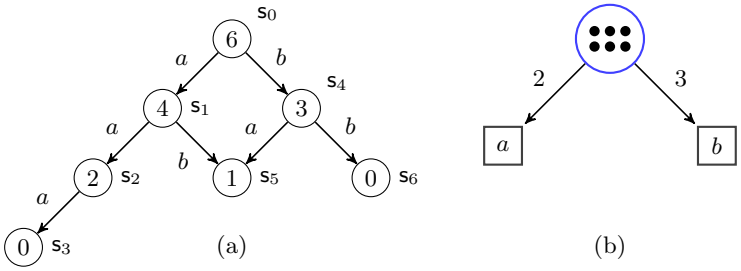


Fig. 1. (a) Region in a TS: $r(\mathsf{s}_0) = 6, r(\mathsf{s}_1) = 4, \ldots, r(\mathsf{s}_6) = 0$, (b) corresponding place in the Petri net.

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

Notice that rational multiplicities are allowed in a multiset. However, for the sake of simplicity, multisets with integer multiplicity will be used for illustrating the concepts of this paper.

Definition 5 (Gradient). *Let $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$ be a TS. Given a multiset r and a transition $s \xrightarrow{\mathsf{e}} s' \in T$, its* gradient *is defined as $\delta_r(s \xrightarrow{\mathsf{e}} s') = r(s') - r(s)$. If all the transitions of an event $\mathsf{e} \in \Sigma$ have the same gradient, we say that the event e has* constant gradient*, whose value is denoted as $\delta_r(\mathsf{e})$.*

Definition 6 (Generalized region). *A* generalized region *r is a multiset defined in a TS, in which all the events have constant gradient.*

Given that we will only use generalized regions, we simply use the term *region* from now on.

Example 1. Fig. 1(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. $\delta_r(a) = -2$ and $\delta_r(a) = -3$. 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 [14]. Fig. 1(b) shows the place corresponding to the region shown in Fig. 1(a).

We say that region r is *normalized* if $\text{minp}(r) = 0$. Any region r can become normalized by subtracting $\text{minp}(r)$ to the multiplicity of all the states:

Definition 7 (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 8 (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 9 (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:*

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

Proof. If r is a region, then $\forall \mathsf{e} \in \Sigma, \forall \mathsf{s}_1 \xrightarrow{\mathsf{e}} \mathsf{s}_2 \in T$ we have that $r(\mathsf{s}_2) - r(\mathsf{s}_1) = \delta_r(\mathsf{e})$. If $r_2 = r_1 + r_0$, we have that $\forall s \in S, r_2(s) = r_1(s) + r_0(s)$. Now for any event e $\delta_{r_2}(\mathsf{e})$ is constant (*i.e.* r_2 is a region), because $\forall \mathsf{s}_1 \xrightarrow{\mathsf{e}} \mathsf{s}_2 \in T, r_2(\mathsf{s}_2) - r_2(\mathsf{s}_1) = r_1(\mathsf{s}_2) + r_0(\mathsf{s}_2) - (r_1(\mathsf{s}_1) + r_0(\mathsf{s}_1)) = \delta_{r_1}(\mathsf{e}) + \delta_{r_0}(\mathsf{e})$. Consequently, $\Delta(r_1 + r_2) = \Delta(r_1) + \Delta(r_2)$.

The properties of the scalar product are proved similarly. First, $k \cdot r$ is a region, because, if $k = 0$, then the result is the trivial empty region, and otherwise the gradient of each event is $\delta_{k \cdot r}(\mathsf{e}) = k \cdot r(\mathsf{s}_2) - k \cdot r(\mathsf{s}_1) = k \cdot (r(\mathsf{s}_2) - r(\mathsf{s}_1)) = k \cdot \delta_r(\mathsf{e})$, for any $\mathsf{s}_1 \xrightarrow{\mathsf{e}} \mathsf{s}_2 \in T$.

The scalar sum properties are trivially proved, and the subtraction properties come from $r_1 - r_2 = r_1 + (-1)r_2$. Finally the normalized sum properties are implied by the fact that $\Delta(r_1 \oplus r_2) = \Delta(\downarrow(r_1 + r_2)) = \Delta((r_1 + r_2) - \mathrm{minp}(r_1 + r_2)) = \Delta(r_1 + r_2)$. □

Regions can be partitioned into classes using $\Delta(r)$.

Definition 10 (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 canonical 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. $\downarrow r$.

An example of canonical region is provided in Fig. 3(b), where a TS in which some regions have been shadowed is shown. The canonical region $r_1 = \{\mathsf{s}_1, \mathsf{s}_2\}$ has gradient vector $\Delta(r_1) = (+1, +1, -1)$. Under some conditions, the set of minimal canonical regions is enough to guarantee some equivalence between the TS and the derived PN [1].

Definition 11 (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 $r' \equiv \emptyset$.*

2.3 Derivation of regions from gradient vectors

A region corresponding to a gradient vector can be obtained by traversing the TS from the initial state, with an arbitrary multiplicity (0 for instance), and giving a multiplicity to each discovered state based on the multiplicity of the source state

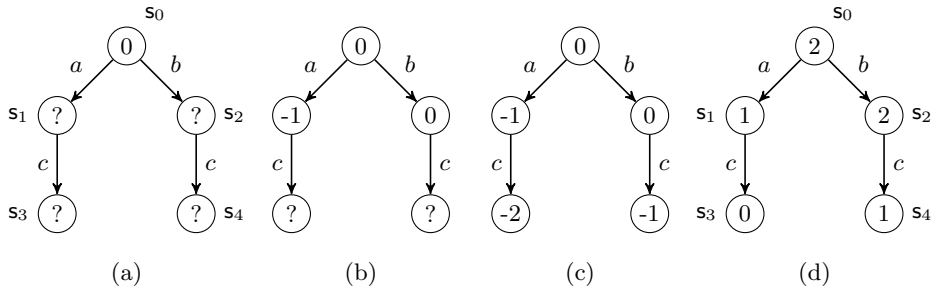


Fig. 2. The obtention of the region with gradient $(-1, 0, -1)$ in a TS, using a breadth-first search. (a) A zero multiplicity is assigned to the initial state. (b,c) Multiplicities after the first and second iterations, respectively. Some of the multiplicities are negative. To normalize them, the minimum power, in this case -2, is subtracted to all states, yielding the region in (d).

of any incoming arc and the gradient of the event that labels the arc. To obtain a normalized region, the smallest multiplicity computed during the exploration (*i.e.* the minimum power of the region) is stored, and then subtracted to all multiplicities. An example is shown in Fig. 2.

3 Finding a region basis

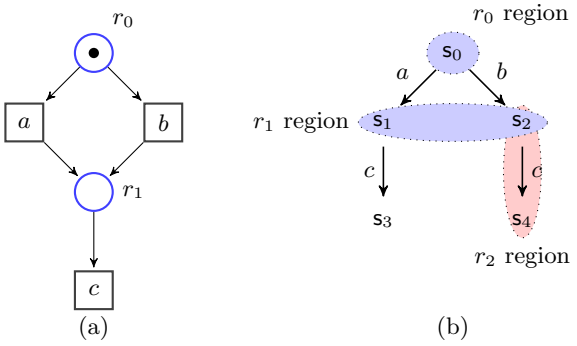
The goal of this section is to obtain a *basis* of regions, i.e. a set of linearly independent regions B that can represent any other region by linear combinations of the elements in B. In general, the size of B is significantly smaller than the number of minimal canonical regions (see Theorem 1 below).

Definition 12 (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 the elements in B (*i.e. $r = \sum_{i=1}^{n} c_i \cdot r_i$, *with $c_i \in \mathbb{Q},\ r_i \in B$).*

The set of canonical regions of a TS, together with the normalized sum operation ($\oplus$), forms a free Abelian group [13]. 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. In particular all the minimal canonical regions can be generated from the basis. As the following theorem states, the size of such a basis is bounded:

Theorem 1 ([13]). *The size of a region basis for TS $A = \langle S, \Sigma, T, \mathsf{s}_0 \rangle$ is less or equal to* $\min(|\Sigma|, |S| - 1)$.

Example 2. In TS of Fig. 3(b), the set of minimal canonical regions is formed by $r_0 = \{\mathsf{s}_0\}$, $r_1 = \{\mathsf{s}_1, \mathsf{s}_2\}$, $r_2 = \{\mathsf{s}_2, \mathsf{s}_4\}$, $r_3 = \{\mathsf{s}_1, \mathsf{s}_3\}$, $r_4 = \{\mathsf{s}_3, \mathsf{s}_4\}$. However,



Event	$\Delta(r_0)$	$\Delta(r_1)$	$\Delta(r_2)$
a	-1	+1	0
b	-1	+1	+1
c	0	-1	0

(c)

Fig. 3. PN whose places, r_0 and r_1, have regions that cannot produce, by linear combination, some of the regions present in its RG, for instance region r_3.

we can express r_3 and r_4 in terms of the other regions: $r_3 = -r_0 - r_2$ and $r_4 = -r_0 - r_1$. Note that, without normalizing the resulting regions it might be difficult to see the equivalence. For instance $-r_0 - r_1 = \{-\mathsf{s}_0, -\mathsf{s}_1, -\mathsf{s}_2\}$ which requires to subtract -1 (add 1) to each state multiplicity to obtain a normalized region, thus $\{-\mathsf{s}_0, -\mathsf{s}_1, -\mathsf{s}_2\} + 1 = \{\mathsf{s}_3, \mathsf{s}_4\} = r_4$. Since any region can be expressed as a sum of minimal canonical regions [13], and r_3 and r_4 are linear combinations of the other regions, a possible basis is formed by only three regions (as there are only three events), namely r_0, r_1 and r_2, whose gradient vectors appear in Fig. 3(c).

In the previous example we have found a basis from the set of minimal canonical regions. In Sect. 4, the opposite process will be performed: from a basis, obtain a set of minimal canonical regions. What remains in this section is to present methods to obtain a basis.

An efficient method to compute a region basis without requiring the set of minimal canonical regions can be devised if we use the following observation: a set of regions whose corresponding gradient vectors form a basis of the universe of gradient vectors also forms a basis of the universe of regions.

Proposition 2. *Given a TS A, let CGR_A denote the set of canonical regions of A and D_A be the set of their gradient vectors. If $B_\Delta = \{d_1, d_2, \ldots, d_n\} \subseteq D_A$ is a basis of the group $(D_A, +)$, then $B = \{r_1, r_2, \ldots r_n\} \subseteq CGR_A$ such that $\Delta(r_i) = d_i$ is a basis for the group $(CGR_A, \oplus)$.*

Proof. To prove that B is a basis, two properties must be shown: first, any region can be obtained as a linear combination of the elements in B. The Δ function establishes an isomorphism between the free Abelian groups $(\mathrm{CGR}_A, \oplus)$ and $(D_A, +)$. Thus any gradient vector in D_A can be expressed as $\sum_i c_i d_i$, which is the gradient vector of the region $\bigoplus_i c_i r_i$. Since any normalized region in A has its gradient vector in D_A, and any such vector is the gradient vector of a region $\bigoplus_i c_i r_i$, then any region in CGR_A can be generated as a linear combination of the elements in B. The second required property for B to be a basis is that

the elements in B are linearly independent. Since both groups $(\mathrm{CGR}_A, \oplus)$ and $(D_A, +)$ are isomorphic, their basis has the same rank (*i.e.* have the same number of elements), implying that all elements in B are linearly independent. □

Hence, a region basis can be found by: (i) find a gradient basis, and then (ii) generate the corresponding regions as explained in Sect. 2.3. Next section shows how to do the first step.

3.1 Computing a basis of gradient vectors

We extend the concept of gradient of an event to sequences of events:

Definition 13 (Gradient of a sequence). *Let σ be a sequence of events, and r a region. The gradient of σ in r, denoted $\delta_r(\sigma)$, is $\sum_{e\in\Sigma} \#(\sigma, e) \cdot \delta_r(e)$.*

The following property is crucial for the method developed in this section to compute a gradient basis:

Property 1. Any region r of a TS has gradients such that any two paths $\mathsf{s} \xrightarrow{\sigma} \mathsf{s}'$ and $\mathsf{s} \xrightarrow{\sigma'} \mathsf{s}'$ satisfy that $\delta_r(\sigma) = \delta_r(\sigma')$.

Proof. Assume $\delta_r(\sigma) \neq \delta_r(\sigma')$. Now compute $r(\mathsf{s}')$ as $r(\mathsf{s}) + \delta_r(\sigma)$. Since $r(\mathsf{s}')$ is also $r(\mathsf{s}) + \delta_r(\sigma')$, we have that $r(\mathsf{s}') \neq r(\mathsf{s}')$, which is a contradiction. □

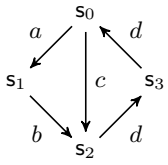
Property 1 is automatically satisfied if there is only a single path connecting any two states, or the only paths between two states fire exactly the same events the same number of times (but possibly in different order). That is, if a state has the same *Parikh vector* no matter the path used to reach it.

Definition 14 (Parikh vector, Parikh vector table). *Given a TS $A = \langle S, \Sigma, T, \mathsf{s}_0\rangle$, the* Parikh vector of a sequence σ *is a vector $p_\sigma \in \mathbb{N}^{|\Sigma|}$ such that $p_\sigma(\mathsf{e}) = \#(\sigma, e)$. The* set of Parikh vectors of a state s*, denoted P_s, contains the Parikh vectors of all sequences σ that start from s_0 and end in s. If all states in S have a single Parikh vector,* i.e. $|P_\mathsf{s}| = 1$*, the* Parikh vector table *of TS A is a table with $|S|$ columns, in which each column contains the Parikh vector of one state in S.*

If P_s contains more than one element, then Property 1 is not guaranteed. Thus, any feasible gradient of a region r must make these Parikh vectors compatible, *i.e.* the multiplicity in r of state s must be the same no matter the path taken to reach it. In particular, for any state s and any two sequences σ and σ' such that $\mathsf{s}_0 \xrightarrow{\sigma} \mathsf{s}$, $\mathsf{s}_0 \xrightarrow{\sigma'} \mathsf{s}$ and $p_\sigma \neq p_{\sigma'}$, it must be true that $\delta_r(\sigma) - \delta_r(\sigma') = 0$. This is that

$$\sum_{\mathsf{e}\in\Sigma} (p_{\sigma'}(\mathsf{e}) - p_\sigma(\mathsf{e})) \cdot \delta_r(\mathsf{e}) = 0. \tag{1}$$

We can use Eqn. 1 as a building block of an algorithm that computes a gradient basis. The algorithm comprises two phases:



(a)

Parikh vectors

$p_\epsilon = (0,0,0,0)$
$p_a = (1,0,0,0)$
$p_c = (0,0,1,0)$
$p_{ab} = (1,1,0,0)$
$p_{cd} = (0,0,1,1)$
$p_{cdd} = (0,0,1,2)$

Conflicts

$p_{ab} - p_c = (1,1,-1,0)$
$p_{cdd} - p_\epsilon = (0,0,1,2)$

Any region r satisfies

$$\begin{pmatrix} 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} \delta_r(a) \\ \delta_r(b) \\ \delta_r(c) \\ \delta_r(d) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Gradient basis: $\{(-1,1,0,0),(-2,0,-2,1)\}$

Region basis: $\{\{\mathsf{s}_0,\mathsf{s}_2,\mathsf{s}_3\},\{2\mathsf{s}_0,\mathsf{s}_3\}\}$

Fig. 4. Computing the gradient basis of a TS. The symbol ϵ is used for the empty sequence.

- Traverse the TS, computing the Parikh vectors assigned to each state and recording the conflicts (Algorithm 1 below).
- From the list of Parikh vector conflicts, build a system of equations using Eqn. 1 from which we can derive the gradient basis.

Algorithm 1 shows the exploration phase of the algorithm. The function returns a set C of Parikh vector differences. Following Eqn. 1, any feasible gradient of the system must satisfy all these differences. We can write such condition in matrix form as $M \cdot \Delta(r)^T = \mathbf{0}$, where $\Delta(r)^T$ is the gradient vector written as a column vector, and each row of matrix M contains one element of C.

Example 3. Consider the TS of Fig. 4. Starting from s_0, Parikh vectors of each state are computed. In some cases there are states that have different Parikh vectors assigned (s_2 and s_0). Such cases are recorded as conflicts, and the difference in the Parikh vectors is used to construct the matrix that enforces the equality of all conflicting Parikh vectors.

It is important to realize that in general (e.g. for cyclic TSs), P_s might be infinite but only a finite subset is needed by our algorithm: consider that in the example above the algorithm uses the conflict between p_{abdd} and p_ϵ instead of the one between p_{cdd} and p_ϵ. The result would be the same since $\delta_r(ab) = \delta_r(c)$ once the conflict between p_{ab} and p_c is solved.

Proposition 3. *If $M \cdot \Delta(r)^T = \mathbf{0}$, then $\Delta(r)$ is the gradient of a region.*

Proof. Let σ and σ' be two sequences $\mathsf{s} \xrightarrow{\sigma} \mathsf{s}'$ and $\mathsf{s} \xrightarrow{\sigma'} \mathsf{s}'$. In order to have a region it must be true that $\delta_r(\sigma) = \delta_r(\sigma')$. Rewriting this expression we obtain $(p_\sigma - p_{\sigma'}) \cdot \Delta_r^T = 0$. If both sequences have a common prefix or suffix, such that $\sigma = \alpha\omega\beta$ and $\sigma' = \alpha\omega'\beta$, then $p_\sigma - p_{\sigma'} = p_\omega - p_{\omega'}$, so without loss of generality we can assume σ and σ' have no state in common besides s and s'. Let γ be a sequence without cycles such that $\mathsf{s}_0 \xrightarrow{\gamma} \mathsf{s}$. If γ is unique, then $|P_\mathsf{s}| = 1$. Let $p_\mathsf{s} \in P_\mathsf{s}$. Now $(p_\sigma - p_{\sigma'}) \cdot \Delta_r^T = 0$ is equivalent to $((p_\mathsf{s} + p_\sigma) - (p_\mathsf{s} + p_{\sigma'})) \cdot \Delta_r^T = 0$,

Algorithm 1 find Parikh vector conflicts

```
function FIND_CONFLICTS(TS A = ⟨S, Σ, T, s0⟩)
    P_s0 ← {(0, 0, ..., 0)}                          ▷ Assign zero Parikh vector
    for all s ∈ S − {s0} do P_s0 ← ∅                  ▷ Initialize the rest
    E ← {s0}                                         ▷ Set of states to explore
    V ← ∅                          ▷ Set of visited states (whose arcs have been visited)
    while E ≠ ∅ do
        select s in E
        E ← E − {s}                  ▷ Remove s from the set of states to explore
        select p in P_s
        for all s -e-> s' ∈ T do
            p' ← p
            p'(e) ← p'(e) + 1                        ▷ p' is one of the Parikh vectors of s'
            P_s' ← P_s' ∪ {p'}                       ▷ Update set of Parikh vectors
            if s' ∉ V then E ← E ∪ {s'}
        end for
        V ← V ∪ {s}                                  ▷ Mark s as visited
    end while
    C ← ∅                                            ▷ Initialize set of conflicts
    for all s such that |P_s| > 1 do
        select p in P_s
        for all p' in P_s − {p} do C ← C ∪ {p − p'}
    end for
    return C
end function
```

thus $((p_{\gamma_\sigma})-(p_{\gamma\sigma'}))\cdot\Delta_r^T = 0$, which is an equation in $M\cdot\Delta(r)^T = \mathbf{0}$ if $p_{\gamma_\sigma})-(p_{\gamma\sigma'}$ is not already 0.

On the other hand, if γ is not unique and another $\mathsf{s}_0 \xrightarrow{\gamma'} \mathsf{s}$ exists, then it is either possible that Algorithm 1 adds $((p_{\gamma_\sigma}) - (p_{\gamma\sigma'})) \cdot \Delta_r^T = 0$, which has been already considered, or $((p_{\gamma_\sigma}) - (p_{\gamma'\sigma'})) \cdot \Delta_r^T = 0$ (or any other combination of the sequences). In such case, if $p_\gamma = p_{\gamma'}$, we are done. Otherwise, there is an equation guarantying $p_\gamma = p_{\gamma'}$ in M, since we have a conflict. The induction is possible since the γ sequences are always decreasing in size. □

So the problem reduces to finding the solutions to the homogeneous linear system $M \cdot \Delta(r)^T = \mathbf{0}$. Each solution of this equation system identifies a feasible gradient in the TS. Note that the system requires to have solutions in the integer domain because, by definition, all gradients have to be integers.

Homogeneous linear systems have one trivial solution (*i.e.* $\mathbf{0}$) and infinite non-trivial solutions. If the homogeneous linear system is represented by a matrix M, it is said that all these solutions form the *nullspace* of M. The nullspace of a matrix has a basis of solutions, that is, every solution to the homogeneous linear system can be obtained by linear combination of the solution vectors in the basis. Formally, if the basis of the nullspace of M is formed by the vectors $\{\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots\}$, then any solution $\boldsymbol{x}$ can be written as a unique linear combination $\boldsymbol{x} = \sum_i \lambda_i \boldsymbol{y_i}$, with $\lambda_i \in \mathbb{Q}$. Consequently any integer basis of the nullspace of

matrix M is a valid gradient basis, since any valid gradient can be written as a linear (rational) combination of these gradients.

There are several well-known methods to obtain a basis for the nullspace of a matrix [15]. Basically they involve performing a Gaussian elimination on matrix M. Since matrix M has $|C|$ rows and $|\Sigma|$ columns, the cost of such operation is $O(|C|^2 \cdot |\Sigma|)$. Once the basis has been computed, the only additional step to perform is to check if some of the resulting vectors contains a non-integer number. In such case, since all numbers are rational, the vector is multiplied by the minimum common multiple of all denominators to obtain an integer gradient.

In the example of Fig. 4, the basis of the nullspace of matrix M is formed by gradient vectors $(-1, 1, 0, 0)$ and $(-2, 0, -2, 1)$, from which we can obtain (using the technique shown in Sect. 2.3) the regions $\{\mathsf{s}_0, \mathsf{s}_2, \mathsf{s}_3\}$ and $\{2\mathsf{s}_0, \mathsf{s}_3\}$, respectively, which form a region basis.

The procedure presented in this section allows finding a region basis for any arbitrary TS by generating first a gradient basis. In some cases, however, this intermediate step can be avoided, as we will see in the next section.

3.2 Region basis from a language

In the area of Process Mining [16], the input object is typically a set of traces, i.e. a *language*, rather than a TS. To be able to use the theory of regions in such case two solutions have been proposed. The first one is to use the theory of regions adapted to languages (language mining) [4], and the other is to convert the language into a TS and then apply the classical theory of regions.

In [17] three types of conversions from a language to a TS were proposed. The main difference between the conversions, namely *sequence*, *multiset* and *set*, is how they decide if the occurrence of an event in a trace produces a new state in the TS or just introduces an arc to an already existing state. In this paper we will focus on the first two conversions. In the *sequence* conversion, two traces only lead to the same state if they fire the same event in exactly the same order. For instance for $L = \{abc, bad\}$, the TS obtained from this conversion is shown in Fig. 5(a). In the *multiset* conversion this condition is relaxed so that the events do not have to happen in the same order, but still it is required that the number of occurrences of each event to be equal. With the previous log, this yields the TS of Fig. 5(b).

Given a language, for the sequence and multiset conversions it is guaranteed that all paths leading to a state will have the same Parikh vector. Then, when computing the gradient basis in the resulting TSs, there will be no conflict and all the variables will be free. Hence we can chose an arbitrary set of $|\Sigma|$ linearly independent gradients as a gradient basis. For simplicity, a good option is to chose the *standard basis*, that is the basis formed by all the linearly independent vectors in which only one event has gradient one, and the rest have gradient zero (see below an example). Computing the corresponding regions is very natural when the standard basis is used, since we only need the Parikh vector of each state, which has been already computed while searching for Parikh vector conflicts.

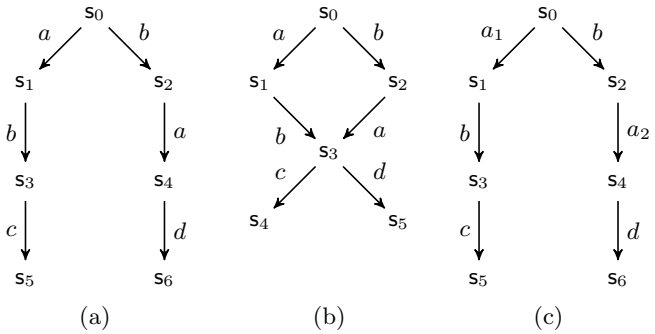


Fig. 5. (a) A TS. (b) Its quotient TS. The quotient TS accepts more traces, but their PN *without label splitting* is the same. (c) If label splitting is performed (for synthesis purposes), then the TS and it quotient TS coincide.

Example. The TS of Fig. 3(b) contains no Parikh vector conflict. Thus we will use the regions with gradients $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$, *i.e.* the standard basis, as the basis. To compute their regions we use the Parikh vector in each state, shown in the *Parikh vector table* to the right. Each row corresponds to one of the regions, so that region with gradient $(1,0,0)$ is $\{\mathsf{s}_1, \mathsf{s}_3\}$, region with gradient $(0,1,0)$ is $\{\mathsf{s}_2, \mathsf{s}_4\}$, and gradient $(0,0,1)$ belongs to region $\{\mathsf{s}_3, \mathsf{s}_4\}$.

Event	s_0	s_1	s_2	s_3	s_4
a	0	1	0	1	0
b	0	0	1	0	1
c	0	0	0	1	1

Proposition 4. *In a TS without Parikh vector conflicts, a row in the Parikh vector table corresponds to a region.*

Proof. For an event e, the value of the Parikh vector table for the row assigned to event e and the column of state s is $p_{\mathsf{s}}(\mathsf{e}) = \#(\sigma, e)$, thus the multiplicity of the corresponding region r is $r(\mathsf{s}) = \#(\sigma, e)$. We prove that r is a region by proving that the gradient of all events is constant. First of all, the value assigned to a state s is the same no matter which sequence σ is used to reach s, because the TS has no Parikh vector conflicts. For an event $a \neq e$ the gradient is 0, since any arc $\mathsf{s} \xrightarrow{\mathsf{a}} \mathsf{s}'$ has a gradient $r(\mathsf{s}') - r(\mathsf{s}) = 0$ as, if σ leads to s, then $\#(\sigma a, e) = \#(\sigma, e)$. Similarly, event e has gradient 1, because $\#(\sigma e, e) = \#(\sigma, e) + 1$. □

Regions for sequential and multiset language representations

There can be significant differences when using either sequential or multiset conversions, since typically the sequential conversion yields TSs with much more states. So it is a relevant question to decide whether there is some advantage of the sequential conversion over the multiset conversion. As we will prove, as far as regions are concerned, there is no difference between both approaches.

Definition 15. *Two states s and s' in a TS are said to be equivalent, $s \equiv s'$, if, for all region r of the TS, $r(s) = r(s')$. We denote the equivalence class of s as $[s]$.*

Proposition 5. *Two states s and s' in a TS A are equivalent if, for all region r in the region basis of A, $r(s) = r(s')$.*

Proof. A region basis can generate any canonical region r as $r = \downarrow \sum_i c_i r_i$. Thus, for state s we have $r(\mathsf{s}) = \downarrow \sum_i c_i r_i(\mathsf{s}) = \downarrow \sum_i c_i r_i(\mathsf{s}') = r(\mathsf{s}')$. □

The state equivalence relation partitions the set of states in equivalence classes. The TS that abstracts the behavior of a given TS at the level of the equivalence classes is called the *quotient TS* .

Definition 16 (Quotient TS). *Let $A = \langle S, \Sigma, T, s_0 \rangle$ be a TS. The* quotient TS *of A, denoted $A_{/\equiv} = \langle S_{/\equiv}, \Sigma, T_{/\equiv}, [s_0] \rangle$, is the TS that results from A by merging all the states in an equivalence class. Formally, $S_{/\equiv} = \{[s] \mid s \in S\}$ and $T_{/\equiv} = \{[s] \xrightarrow{\mathsf{e}} [s'] \mid s \xrightarrow{\mathsf{e}} s' \in T\}$.*

Let us construct the quotient TS of Fig. 5(a). To determine which states are equivalent, we use Proposition 5 on a basis, which can be obtained by the method shown in the previous section. Since the TS is acyclic and has no conflicts, by Proposition 4, each row of the Parikh vector table below corresponds to one of the regions in the standard basis.

Using Proposition 5 any two states that have the same multiplicity in all the regions of the basis must have the same columns in the table. The only two states fulfilling this condition are s_3 and s_4, which can be merged obtaining the TS shown in Fig. 5(b).

Event	s_0	s_1	s_2	s_3	s_4	s_5	s_6
a	0	1	0	1	1	1	1
b	0	0	1	1	1	1	1
c	0	0	0	0	0	1	0
d	0	0	0	0	0	0	1

Theorem 2. *Let $A = \langle S, \Sigma, T, s_0 \rangle$ be a TS, and $A_{/\equiv} = \langle S_{/\equiv}, \Sigma, T_{/\equiv}, [s_0] \rangle$ be the quotient TS. For any canonical region r of A there is a canonical region r' in $A_{/\equiv}$ such that $\Delta(r) = \Delta(r')$ and $r(s_0) = r'(s_0)$, and vice versa.*

Proof. Consider region r from A. Since it is a region we have that $\forall \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T, r(\mathsf{s}') - r(\mathsf{s}) = \delta_r(\mathsf{e})$. And because it is normalized we have that $\exists \mathsf{s} : r(\mathsf{s}) = 0$ and $\forall \mathsf{s}\ r(\mathsf{s}) \geq 0$. Consider the multiset r' such that $r'([\mathsf{s}]) = r(\mathsf{s})$, we will prove that it is a canonical region with the same gradient and multiplicity in the initial state as r. Consider an arc $[\mathsf{s}] \xrightarrow{\mathsf{e}} [\mathsf{s}']$ in $T_{/\equiv}$, clearly $r'([\mathsf{s}']) - r'([\mathsf{s}]) = r(\mathsf{s}') - r(\mathsf{s}) = \delta_r(\mathsf{e})$. Thus all gradients are constant and r' is a region. Moreover, since all equivalent states $\mathsf{s} \in [\mathsf{s}]$ have the same multiplicity in any region, then $\exists [\mathsf{s}] : r([\mathsf{s}]) = 0$ and $\forall [\mathsf{s}]\ r([\mathsf{s}]) \geq 0$, which proves that r' is a canonical region. The proof in the other direction follows the same reasoning. □

Proposition 6. *Let L be a language and A_s and A_m be two TSs obtained from L using the sequence conversion and the multiset conversion, respectively. Then $A_m = A_{s/\equiv}$.*

Proof. We will prove that all the states of A_s that are equivalent are the ones that fire the same multiset of events. Consider the standard region basis for A_s and build its Parikh vector table. By definition two states are equivalent if they have the same multiplicity for all the regions in the basis. That is, if they have the same columns in the Parikh vector table, *i.e.* the two states have the same Parikh vector. As the Parikh vector is a representation of the multiset of events fired to reach the state, both states will be the same in A_m. □

An important consequence of Proposition 6 and Theorem 2 is that multiset conversion provides the same information, in terms of regions, as the sequence conversion, but using less states. This is relevant because the performance of some tools is specially affected by the number of states in the TS and its shape. This result might seem surprising considering that the quotient TS may contain more traces than the original TS. For instance, sequence *abd* is possible in Fig. 5(b), but not in Fig. 5(a). However Theorem 2 shows that the derived PNs are the same. Note that if label splitting [14] is allowed, then no extra behavior might be accepted by the quotient TS, as illustrated in (c).

Beyond the multiset language representation

In the previous section we have seen conditions allowing to reduce the number of states of a TS while obtaining the same net. This section proposes a more powerful reduction technique that considerably diminishes the size of the TS at the cost of forbidding some specific regions.

The technique, named *common final marking* (CFM) reduction, has two steps:

- From a TS A obtained by multiset conversion, create a TS A' by merging all *sink states* (states without outgoing arcs) into a single state. We say that A' is the *single sink version of* A.
- Merge equivalent states in A', by merging all states that are either reachable from a state s through the same event or reach the same state through a common event, until no further state is mergeable.

Theorem 3. *Let A be a TS and A' its single sink version. Consider a TS A'' obtained from A' by merging all states that are either reachable from a state s through the same event or reach the same state through a common event, until no further state is mergeable. Let N' and N'' be the PNs including all the regions of A' and A'', respectively. Then, $\mathcal{L}(N') \supseteq \mathcal{L}(A)$ and $N' = N''$.*

Proof. Merging the sink states of A does not introduce any new trace in the language, so $\mathcal{L}(A') = \mathcal{L}(A)$. However, A has no conflict while A' can contain some of them. This yields a smaller region basis, thus some regions of A are no longer feasible in A', consequently PN N' obtained from A' satisfies $\mathcal{L}(N') \supseteq \mathcal{L}(A)$. Now consider two states s_1 and s_2 such that the transitions $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}_1$ and $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}_2$. In any possible region r, both states will have the same multiplicity since $r(\mathsf{s}_1) = r(\mathsf{s}) + \delta_r(\mathsf{e}) = r(\mathsf{s}_2)$. Thus, by Proposition 5, $\mathsf{s}_1 \equiv \mathsf{s}_2$, and both states

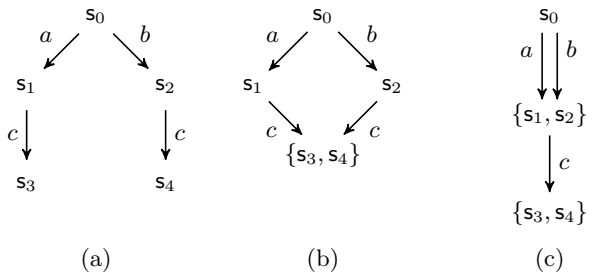


Fig. 6. (a) A TS. (b) single sink version of (a), (c) Merge of states s_1 and s_2 is possible since they are equivalent.

can be merged. The same applies if $\mathsf{s}_1 \xrightarrow{\mathsf{e}} \mathsf{s}$ and $\mathsf{s}_2 \xrightarrow{\mathsf{e}} \mathsf{s}$. Since A'' is simply A' but merging equivalent states, by Theorem 2 N'' and N' must be the same. □

For instance in Fig. 6 we can see a TS (from Fig. 3), that could come from $L = \{ac, bc\}$. Both the sequential and the multiset conversion yield the same TS, shown in (a). This TS has two sink states s_3 and s_4, which can be safely merged obtaining a TS, depicted in (b), with the same language. This merged state has two incoming arcs with the same label, thus, the predecessors of such arcs, namely s_1 and s_2 can be also safely merged, since they will be assigned the same multiplicity in every possible region.

4 Generating a PN from a basis

Once the region basis is available, we can generate a PN from it. A naive strategy would be to use a brute-force approach and generate some amount of regions, and then remove the redundant ones among them using for instance the techniques in [4]. However this approach is clearly inefficient.

An alternative generation scheme is to try to find the minimal canonical regions. The straightforward approach would be to have a set of the currently minimal regions found so far in the exploration, and every time a new region is generated, check against all the regions in the set if it is minimal or not. However, this method requires to perform many subregion checks per region, and most of the times the regions checked are not minimal.

Our proposal (Algorithm 2), denoted *minimal canonical region search*, prevents from checking the minimality of regions that are guaranteed not to be minimal. If $B = \{r_1, \ldots, r_n\}$ is the region basis, the algorithm computes the set of minimal canonical regions R in a DFS manner. It starts with the empty region $\emptyset$ to whom normalized basis regions $\downarrow(c_i \cdot r_i)$, with $c_i \neq 0$, are added. The first addition creates a normalized region $r = \downarrow(c_i \cdot r_i)$ from a single basis region. These type of regions are always checked for minimality since $size$ is always 1

in such case. Thus, they are added into R if no smaller region is present in R (line 14). This guarantees that R contains either r or one of its subregions.

From that point on, combinations including r are explored, by adding other normalized basis regions (line 19). Let $r' = \downarrow r + \downarrow(c_j \cdot r_j)$ be one of such explored regions. It is trivially true that $r' \supseteq \downarrow r$. Since $\downarrow r$ is a normalized region, if r' is also normalized, it follows that $\downarrow r' \supseteq \downarrow r$, thus $r' \sqsupseteq r$ and r' is not minimal.

Consequently, the algorithm is devised to detect whether the addition of some region basis $(c_i \cdot r_i)$ to a region r produces a non-normalized region. This check is performed in line 3, based on the fact that, if r is already normalized, then the multiplicity of any state must be the same in r or $\downarrow r$. In line 3 normalization of r is tested in the initial state s_0 with the condition $\downarrow r(\mathsf{s}_0) \neq r(\mathsf{s}_0)$. Only the regions satisfying this condition are possible minimal canonical regions, and are checked against all the regions in the current set R.

Algorithm 2 mcr_search

```
 1: procedure MCR_RECURSIVE(r, size, pos)
 2:     nr ← ↓r                                              ▷ Normalize r
 3:     if size = 1 ∨ nr(s0) ≠ r(s0) then      ▷ Check r = c_i · r_i, or r non-normalized
 4:         if ∃e : δ_r(e) < 0 then          ▷ Regions with positive gradients are useless
 5:             useful ← true                              ▷ Initially consider nr is minimal
 6:             for all mr ∈ R do
 7:                 if mr ⊆ nr then
 8:                     useful ← false                     ▷ If nr is not minimal discard it
 9:                     break
10:                 else
11:                     if nr ⊆ mr then R ← R − {mr}       ▷ Remove mr (not minimal)
12:                 end if
13:             end for
14:             if useful then R ← R ∪ {nr}                ▷ Add nr as minimal region
15:         end if
16:     end if
17:     if size < agg then                                 ▷ Check aggregation factor
18:         for all i ∈ [pos, |B|] and j ∈ [minval, maxval] − {0} do
19:             mcr_recursive(nr + ↓(j · r_i), size + 1, pos + 1)
20:         end for
21:     end if
22: end procedure
23:
24: function MCR_SEARCH
25:     R ← ∅                                              ▷ Set of minimal canonical regions
26:     mcr_recursive( ∅, 0, 1 )                           ▷ Call recursive function
27:     return R
28: end function
```

The algorithm uses the following global variables:

- R is the set of minimal canonical regions encountered so far.

$$r_0 = \{\mathsf{s}_0, \mathsf{s}_2, \mathsf{s}_3\} \qquad \downarrow(-r_0) = \{\mathsf{s}_1\}$$
$$r_1 = \{2\mathsf{s}_0, \mathsf{s}_3\} \qquad \downarrow(-r_1) = \{2\mathsf{s}_1, 2\mathsf{s}_2, \mathsf{s}_3\}$$

$$\downarrow(-r_0) + \downarrow(-r_1) = \{3\mathsf{s}_1, 2\mathsf{s}_2, \mathsf{s}_3\} \sqsupseteq \downarrow(-r_0)$$
$$\downarrow(-r_0) + r_1 = \{2\mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_3\} \sqsupseteq \downarrow(-r_0)$$
$$\downarrow(-r_0) + 2r_1 = \{2\mathsf{s}_0, 2\mathsf{s}_1, \mathsf{s}_3\} \sqsupseteq \downarrow(-r_0)$$
$$r_0 + \downarrow(-r_1) = \{\mathsf{s}_0, 2\mathsf{s}_1, 3\mathsf{s}_2, 2\mathsf{s}_3\} \not\sqsupseteq \downarrow(r_0 + \downarrow(-r_1))$$
$$\downarrow(r_0 + \downarrow(-r_1)) = \{\mathbf{s_1}, \mathbf{2s_2}, \mathbf{s_3}\} \sqsupseteq \downarrow(-\mathbf{r_0})$$
$$2r_0 + \downarrow(-r_1) = \{2\mathsf{s}_0, 2\mathsf{s}_1, 4\mathsf{s}_2, 3\mathsf{s}_3\} \not\sqsupseteq \downarrow(2r_0 + \downarrow(-r_1))$$
$$\downarrow(2r_0 + \downarrow(-r_1)) = \{\mathbf{2s_2}, \mathbf{s_3}\} \not\sqsupseteq \downarrow(-\mathbf{r_0}) \Rightarrow \text{added to } R$$

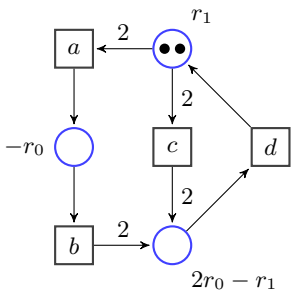


Fig. 7. Some of the regions explored by the algorithm mining the TS in Fig. 4 (only the subregion checks marked in bold are actually performed), and the final PN obtained.

- B is the region basis of the input TS, found by the methods described in Sect. 3.
- agg is an aggregation factor that bounds the number of different basis regions that can be used to obtain a new region.
- $minval \leq -1$ is the amount of times that a *coregion* of the basis can be used.
- Similarly, $maxval \geq 1$ bounds the number of times that a basis region can appear in a combination.

The last three variables are user defined parameters, that allow the user to control the amount of exploration performed in the region space that takes place in lines 17 to 19 of the algorithm. Although the algorithm does not produce k-bounded nets, it is not difficult to adapt it to fulfill such requirement.

To illustrate the behavior of the algorithm, we will follow the first steps of the PN generation using the region basis obtained in Fig. 4. To exemplify the impact of some of the parameters, we will use $minval = -1$ and $maxval = 2$, without limiting the possible combinations of the regions in the basis (*i.e.* using the size of the basis, in this case 2, as the aggregation factor agg). Some of the regions explored and the final PN can be seen in Fig. 7. The values of the $minval$ and $maxval$ parameters are loosely related to the weights in the arcs one would expect in the set of minimal regions. In this example, using $maxval = 1$ would prevent the algorithm from finding on of the places in the net.

Let us name r_0 and r_1 the two region basis in the example, namely $\{\mathsf{s}_0, \mathsf{s}_2, \mathsf{s}_3\}$ and $\{2\mathsf{s}_0, \mathsf{s}_3\}$. Since r_0 and r_1 are already normalized and to ease the notation, we will simply write $k \cdot r_0$, when k is a positive scalar, instead of $\downarrow(k \cdot r_0)$, since these regions are normalized too. First region explored is $\downarrow(-r_0)$, which is added into the R set of minimal canonical regions, because all regions formed using a single region basis are always tested against the regions in R (variable $size$ is always 1 in such cases, and the condition in line 3 evaluates to true) and R is initially empty. After that, regions $\downarrow(-r_0) + \downarrow(-r_1)$, $\downarrow(-r_0) + r_1$ and $\downarrow(-r_0) + 2r_1$

are explored. None of them is non-normalized (see Fig. 7), thus are discarded without been checked for minimality.

Next r_0 is checked, and goes into the R list since it is obviously not a superset of $\downarrow(-r_0)$. From the following combinations $r_0 + \downarrow(-r_1)$, $r_0 + r_1$ and $r_0 + 2r_1$, only the first one corresponds to a non-normalized region. However when checked against the regions in R, it turns out that it is a superregion of $\downarrow -r_0$, thus it is not included in the list. The $2r_0$ region (not shown in the figure) has the trivial subregion r_0, but one of its descendants, $2r_0 + \downarrow(-r_1)$ is minimal. Finally, regions $\downarrow(-r_1)$, r_1 and $2r_1$ are checked and discarded with the exception of r_1. The exploration concludes after generating 15 regions with a list of four minimal canonical regions. From this set, using the simplification techniques described in [4], one of them is removed, yielding the PN shown in Fig. 7.

Proposition 7. *Given a TS A, if a basis of its regions is used in Algorithm 2, then the set of regions returned by the algorithm correspond to a PN N such that $\mathcal{L}(N) \supseteq \mathcal{L}(A)$.*

Proof. The algorithm only computes regions using combinations of the regions in the basis of TS A, thus all the computed regions are regions of A. Since the infinite PN N' built using all the regions of A satisfies $\mathcal{L}(N') \supseteq \mathcal{L}(A)$, and the algorithm only returns a finite subset of these regions, then $\mathcal{L}(N) \supseteq \mathcal{L}(A)$. □

5 Experiments

All the results were obtained on a PC with an Intel Core Duo at 2.10Ghz and 2Gb of RAM, running the 2.6 Linux kernel. For the experiments we limited the amount of memory and time that could be used by the tools to 1Gb and 10000 seconds respectively. Table 1 shows some relevant information of the logs used in the experiments. For each benchmark we give the number of traces it contains (*#cases*), the number of different events present in the log ($|\Sigma|$), the number of states of the corresponding execution tree obtained by the sequential conversion ($|S_s|$), the number of states of the TS obtained by the multiset conversion ($|S_m|$), and the number of states after CFM reduction ($|S_c|$). The time required to build the TS by each type of conversion is given in columns T_s, T_m and T_c, respectively. Column $|C|$ indicates the number of conflicts present in the TS obtained by CFM reduction, while $|B|$ is the size of the corresponding region basis. All the logs have been taken from [9].

As expected the number of states produced by the multiset conversion is inferior to the sequential conversion. The reduction is dramatic in some cases, like the `t32f0n00_5` log. Since we have shown that all conversions are equivalent if no label splitting is allowed, we have used the TSs obtained by the multiset conversion with CFM reduction.

We compare the performance and the quality of three tools: the `Parikh` miner in the ProM tool, `genet` and `rbminer`. The `Parikh` miner [9] uses the language-based theory of regions combined with ILP, `genet` implements the classical TS-based approach with a symbolic representation of the TSs, and the `rbminer`

Log	#cases	$\|\Sigma\|$	$\|S_s\|$	$\|S_m\|$	$\|S_c\|$	T_s	T_m	T_c	$\|C\|$	$\|B\|$
a12f0n00_1	200	12	25	18	13	0	0	0	2	10
a12f0n00_5	1800	12	25	18	13	0	0	0	2	10
a22f0n00_1	100	22	1309	751	86	0	0	0.1	10	16
a22f0n00_5	900	22	9867	3291	80	0.1	0.1	0.3	6	16
a32f0n00_1	100	32	2011	1378	614	0	0.1	0.1	28	26
a32f0n00_5	900	32	16921	5544	481	0.1	0.3	0.4	10	26
t32f0n00_1	200	33	7717	7167	5846	0.1	0.4	0.4	119	27
t32f0n00_5	1800	33	64829	50436	2870	0.4	3.6	4.8	35	27
a42f0n00_1	100	42	2865	2568	1864	0	0.1	0.2	74	35
a42f0n00_5	900	42	24366	15816	8221	0.1	1.1	1.1	192	35

Table 1. Logs used in the experiments.

tool implements the methodology described in this paper. For each method we provide the number of places and arcs (column P/F) of the mined PN (the number of transitions coincides with $|\Sigma|$ since no label splitting is performed), the time in seconds to obtain the PN from the TS, and the well-known quality measure called *appropriateness* [18]. This metric quantifies to which extent the model describes the observed behavior, combined with the clarity degree of the model. It is normalized to be a real number between 0 (low) and 1 (high). All

	`genet`			`Parikh`			`rbminer`		
Log	P/F	Time	App.	P/F	Time	App.	P/F	Time	App.
a12f0n00_1	11/25	0.1	1.0	11/25	1	1.0	11/25	0.1	1.0
a12f0n00_5	11/25	0.1	1.0	11/25	0.7	1.0	11/25	0.1	1.0
a22f0n00_1	19/49	0.3	0.95	19/49	3	0.95	19/49	0.1	0.93
a22f0n00_5	19/49	0.3	0.94	19/49	23	0.95	19/49	0.1	0.94
a32f0n00_1	32/75	718	0.94	31/73	25	0.93	32/75	2	0.94
a32f0n00_5	31/73	1	0.95	31/73	112	0.93	31/73	2	0.95
t32f0n00_1	memout			30/72	288	0.99	31/74	8	0.92
t32f0n00_5	memout			30/72	9208	0.99	30/72	5	0.92
a42f0n00_1	memout			44/109	154	1.0	52/131	10	1.0
a42f0n00_5	timeout			44/101	1557	1.0	46/107	33	1.0

Table 2. Mining of large logs.

the benchmarks were mined using an aggregation factor of 4 for `rbminer`, with $minval = -1$, $maxval = 1$ and $k = 1$.

The benefits of using basis of regions are twofold. On the one hand, the memory consumption is very low. For instance, in all the experiments the maximum amount of memory used by `rbminer` was 10Mb. This is a clear advantage over other approaches, notably `genet`, which is very memory demanding. On the other hand the running times are, in general, much lower than in the other tools. For some benchmark even three orders of magnitude (`t32f0n00_5`), although most of the times the improvement is between one and two orders of magnitude. The crucial step in obtaining such improvements is the use of the CFM reduction. We have repeated the experiments using the multiset conversion alone, and the running times became of the same order of magnitude as the ones obtained by the `Parikh` tool.

In terms of quality the results are quite similar across all tools. Note that in some cases PNs with the same number of places and arcs have different appropriateness because they are not identical.

In addition to the experiments on mining logs, which always yield acyclic TSs, we have conducted a number of additional experiments on cyclic TSs. The goal of these experiments is to illustrate the capability in reproducing complex behaviors and are shown in Table 3. We have used the same set of benchmarks as in [6], namely

- A model for n processes competing for m shared resources, denoted SR(n,m), where $n > m$.
- A model for n producers and n consumers, denoted PC(n,m), where $n > m$.
- A 2-bounded pipeline of n processes, denoted BP(n).

A comparison of the mining capabilities of the `genet` and `rbminer` tools[3] shows that, in both cases, synthesis was achieved (for the cases in which genet could complete). However the resulting PNs were very different in terms of compactness. In all cases `rbminer` could reconstruct the original model from which the TSs were derived. The aggregation factor was set in each case to the value where synthesis was achieved. Note that for some benchmarks the values are quite low, showing that in many cases a very shallow partial exploration of the region space is enough to obtain remarkable results.

					`genet`		`rbminer`		
Bench.	$\|S\|$	$\|\Sigma\|$	$\|C\|$	$\|B\|$	P/F	Time	Agg	P/F	Time
PC(8,3)	1024	17	8	9	27/86	2	9	18/50	0.1
PC(8,5)	1536	17	8	9	42/158	83	9	18/50	0.1
PC(9,6)	3584	19	9	10	62/256	332	10	20/56	1
SR(6,4)	4077	24	6	18	89/490	20	6	25/60	32
SR(7,5)	16362	28	7	21	241/1865	1190	7	29/70	1565
BP(8)	6561	10	1	8	16/32	1320	2	16/32	0
BP(9)	19683	11	1	9	18/36	4561	2	18/36	0
BP(10)	59049	12	1	10	timeout		2	20/40	0.1

Table 3. Mining of cyclic TSs.

6 Related work

The work presented has some relations with the theory developed in [13], being the contributions of this paper algorithms built on top of that theory.

Related approaches based on the language-based theory of regions are [4, 9], which are based in the seminal work presented in [3]. Informally, these approaches build also a basis of regions (but only allowing positive combinations, thus finding a basis formed by all minimal canonical regions) by solving an homogeneous linear equation system that is proportional to the set of words (and prefixes of the words) that appear in the language, and some of them are also proportional

[3] The Parikh miner cannot handle cyclic TSs, thus not appearing in Table 3.

with the set of *wrong continuations* of words, i.e. words not appearing in the language. This makes the language-based approach to suffer for large inputs, as it is demonstrated in the previous section. However, the TS that arises from a language can be greatly simplified (as explained in Sections 3.2 and 3.2), which in turn alleviates drastically the size of the object needed for deriving a region basis, thus making the approach presented in this paper a good candidate for handling large inputs.

7 Conclusions

This paper presents a fresh look at the problem of deriving a PN from a TS using the theory of regions. By combining ideas that have been applied in the language-based theory of regions (i.e. the generation of a region basis) with drastic simplifications of the input TS, the approach has proven to be superior to any of the existing approaches for Process Mining.

All the theory developed in this paper has been implemented in a tool, which we expect to extend in the future by incorporating enhanced methods to explore efficiently the region space, in particular combining the basis of regions with causality information.

References

1. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4) (1996) 297–315
2. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica **27** (1990) 315–368
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. Bergenthum, R., Desel, J., Lorenz, R., S.Mauser: Process mining based on regions of languages. In: Proc. 5th Int. Conf. on Business Process Management. (September 2007) 375–383
5. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Transactions on Computers **47**(8) (August 1998) 859–882
6. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Transactions on Computers (2009)
7. 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. Technical Report BPM-08-01, BPM Center (2008)
8. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In Dumas, M., Reichert, M., Shan, M.C., eds.: BPM. Volume 5240 of Lecture Notes in Computer Science., Springer (2008) 358–373
9. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In van Hee, K.M., Valk, R., eds.: Petri Nets. Volume 5062 of Lecture Notes in Computer Science., Springer (2008) 368–387

10. Solé, M.: rbminer. http://www.lsi.upc.edu/∼jcarmona/rbminer/rbminer.html
11. Murata, T.: Petri Nets: Properties, analysis and applications. Proceedings of the IEEE (April 1989) 541–580
12. Mukund, M.: Petri nets and step transition systems. Int. Journal of Foundations of Computer Science **3**(4) (1992) 443–478
13. Bernardinello, L., Michelis, G.D., Petruni, K., Vigna, S.: On the synchronic structure of transition systems. In Desel, J., ed.: Structures in Concurrency Theory, Proceedings of the International Workshop on Structures in Concurrency Theory (STRICT), Berlin, 11-13 May 1995. (1995) 69–84
14. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. In: 29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency. (June 2008)
15. Kalman, D.: Basic null space calculations. The College Mathematics Journal **15**(1) (1984) 42–47
16. 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
17. 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)
18. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1) (2008) 64–95