Incremental Process Discovery using Petri Net Synthesis

Eric Badouel, Uli Schlachter

Incremental Process Discovery using Petri Net Synthesis

Eric Badouel
Inria Rennes-Bretagne Atlantique
Campus de Beaulieu, F35042 Rennes, France
eric.badouel@inria.fr

Uli Schlachter*
Department of Computing Science,
Carl von Ossietzky Universität Oldenburg,
D-26111 Oldenburg, Germany
uli.schlachter@informatik.uni-oldenburg.de

July 10, 2017

Abstract

Process discovery aims at constructing a model from a set of observations given by execution traces (a log). Petri nets are a preferred target model in that they produce a compact description of the system by exhibiting its concurrency. This article presents a process discovery algorithm using Petri net synthesis, based on the notion of region introduced by A. Ehrenfeucht and G. Rozenberg and using techniques from linear algebra. The algorithm proceeds in three successive phases which make it possible to find a compromise between the ability to infer behaviours of the system from the set of observations while ensuring a parsimonious model, in terms of fitness, precision and simplicity. All used algorithms are incremental which means that one can modify the produced model when new observations are reported without reconstructing the model from scratch.

1 Introduction

The notion of *region* introduced by Andrzej Ehrenfeucht and Grzegorz Rozenberg [1, 2] is at the origin of numerous studies on the synthesis of Petri nets [3] from specifications given by labelled transitions systems or by languages. Algorithmic solutions for the synthesis of Petri nets using linear algebra techniques

*Supported by DFG (German Research Foundation) through grant Be 1267/15-1 ARS (Algorithms for Reengineering and Synthesis).

[4, 5, 6] and combinatorial techniques [7, 8, 9] have been used in the context of *process discovery* [10, 11]. Indeed, the goal of process discovery [12, 13] is to construct a model of a system under observation from a set of its execution traces, a so-called *log* of the system. Even though a log usually consists of a very large set of executions traces it can not exhaust all possibilities. A process discovery algorithm should therefore be able to generalize. Namely, by learning from the samples of behaviour given in the log it must be able to infer that some execution trace that was not reported in the log is nonetheless a valid behaviour of the system. But it should not generalize too much. Otherwise the model would lack in precision. On the other hand the generated model should be as simple as possible where the complexity of a Petri net would take into account the number of its places, the weights of its flow relation etc. Another key feature is the *incrementality* of the construction which means that one can amend the model when new observations are reported without reconstructing the model from scratch.

It may also be desirable for process discovery to be able to identify possible outliers in the log. Namely, it would discard executions traces that deviate from the behaviour of the intended model and are consequently interpreted as errors. However, Petri net synthesis always provides an over-approximation of the language given as input. Thus the constructed model reproduces all the execution traces reported in the log. We say that it *fits* the log. Therefore, in the Petri net synthesis approach to process discovery one tends to consider that outliers have already been eliminated during a preprocessing phase applied to a concrete version of the log. The full log contains additional informations, including word frequencies, that can be used to detect suspicious behaviours. Thus, one considers that all execution sequences in the log are valid behaviours and therefore a correct model should fit the log. A convenience of this assumption is to enable us to cleanly delineate the learning ability of process discovery. Namely, one should only infer behaviours that are shared by all the models that fit the log. A model is precise if it adds few behaviours to those obtained by generalization. Since Petri net synthesis returns a model whose language is the least over-approximation of the log by a Petri net language (at least when all minimal regions are taken into account) it provides an optimal solution in that respect. However, this solution may be unnecessarily complicated. The remaining concern is thus to find a good compromise between precision and simplicity. For that purpose we present in this paper some techniques and heuristics that can be used to extract simpler models associated with subsets of minimal regions.

Section 2 presents a self-contained account of Petri net synthesis from a prefix-closed language. Our solution toward process discovery is described in Section 3 and assessed in the concluding section.

2 Petri Net Synthesis from a Prefix-closed Language

2.1 Petri Nets

We restrict to Petri nets whose transitions correspond bijectively to a fixed set of events $E = \{e_1, \ldots, e_n\}$. A place p can then be encoded as a vector $\mathbf{p} = (p_0, \ldots, p_{2n}) \in \mathbb{N}^{1+2n}$ where $p_0 = M_0(p)$ is the value of the place in the initial marking and the other components provide the flow relations: $p_i = {}^\bullet p(e_i)$ and $p_{n+i} = p^\bullet(e_i)$. Thus $\mathbf{p} = (M_0(p); {}^\bullet p; p^\bullet)$. For instance the places of the Petri net depicted in Fig. 1 are encoded by the following vectors: $\mathbf{p_1} = (0,2,0,1,0,0,0,0,4)$, $\mathbf{p_2} = (3,0,0,1,4,2,4,0,0)$, and $\mathbf{p_3} = (2,0,2,0,0,0,0,1,0)$. Since its set of transitions is fixed we identify a Petri net with its set of places and represent it by the matrix whose columns are the vectors that encode places.

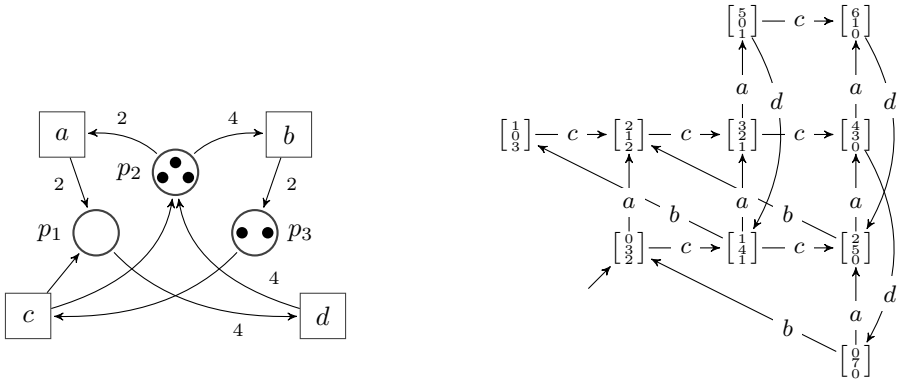

Figure 1: A Petri net and its graph of reachable markings.

The transition relation between markings is given by

$$M[e\rangle M' \iff M[e\rangle \text{ and } M' = M \bullet e$$

where $\bullet$ is a (total) action of the set of events E on the set of "extended" markings $M \in \mathbb{Z}^P$ where P is the set of places (in which we admit negative entries). It is defined via the next three points where $e \in E$ and $w \in E^*$: (1) $M \bullet \varepsilon = M$, (2) $(\forall p \in P) \quad (M \bullet e)(p) = M(p) - p^\bullet(e) + {}^\bullet p(e)$, and (3) $M \bullet (w \cdot e) = (M \bullet w) \bullet e$. Relation $M[w\rangle$ given next ensures that reachable markings contain only non-negative entries: (1) $M[\varepsilon\rangle$ always holds, (2) $M[e\rangle \iff (\forall p \in P) \quad M(p) \geq p^\bullet(e)$, and (3) $M[w \cdot e\rangle \iff M[w\rangle$ and $M \bullet w[e\rangle$ The language of a Petri net $\mathcal{L}(N) = \{w \in E^* \mid M_0[w\rangle\}$ is thus a prefix-closed language of E^*. Note that

$$(M \bullet w)(p) = M_0(p) + \Psi(w)({}^\bullet p - p^\bullet) \tag{1}$$

where $\Psi(w) \in \mathbb{N}^E$, the *Parikh vector* of w, records the number $\Psi(w)(e_i)$ of occurrences of letter e_i in w: $\Psi(w)(e_i) = \sharp_{e_i}(w)$.

2.2 Regions

If $L \subseteq E^*$ is a prefix-closed language one has $L \subseteq \mathcal{L}(N) \iff (\forall w \cdot e \in L) \quad M_0 \bullet w[e\rangle$. Hence by Equation (1): $L \subseteq \mathcal{L}(N) \iff (\forall w \cdot e \in L)(\forall p \in N) \quad M_0(p) + \Psi(w)({}^\bullet p - p^\bullet) \geq p^\bullet(e)$. Using the vector encoding of places this equivalence can be reformulated as

$$L \subseteq \mathcal{L}(N) \iff (\forall w \cdot e \in L)(\forall \mathbf{p} \in N) \quad \langle \Psi(w), e\rangle \cdot \mathbf{p} \geq 0 \tag{2}$$

where $\langle \mathbf{v}, e\rangle = (1; \mathbf{v}; -\mathbf{v} - e)$ for $\mathbf{v} \in \mathbb{N}^E$ and $e \in E$. We let $L^{\mathbf{C}} \subseteq \mathbb{N}^E \times E$ denote the set of pairs $(\mathbf{v}, e)$ for which exists $w \in L$ such that $\Psi(w) = \mathbf{v}$ and $w \cdot e \in L$. We identify $L^{\mathbf{C}}$ with the matrix whose rows are the vectors $\langle \mathbf{v}, e\rangle$.[1] One has

$$L \subseteq \mathcal{L}(N) \iff (\forall \mathbf{p} \in N) \quad L^{\mathbf{C}}\mathbf{p} \geq 0 \tag{3}$$

and then $L \subseteq \mathcal{L}(N) \iff L^{\mathbf{C}}N \geq 0$. In view of Equation (3), we let the *regions* of a prefix-closed language L be the places of the net synthesized from L, given as $\mathcal{N}(L) = \left\{\mathbf{p} \in \mathbb{N}^{1+2n} \;\middle|\; L^{\mathbf{C}}\mathbf{p} \geq 0\right\}$. Hence we have a Galois connection between prefix-closed languages and Petri nets:

$$L \subseteq \mathcal{L}(N) \iff L^{\mathbf{C}}N \geq 0 \iff N \subseteq \mathcal{N}(L) \tag{4}$$

Using Equation (3) with $L = \mathcal{L}(N)$ we get that $(\forall \mathbf{p} \in N) \quad \mathcal{L}(N)^{\mathbf{C}}\mathbf{p} \geq 0$, i.e. a place of a Petri net is a region of its language, called the *extension* of the place. Nonetheless not all regions of $L = \mathcal{L}(N)$ are extensions of places of N. They correspond to places that can be added to N without modifying its language. Due to the Galois connection the composition $\mathcal{N}(\mathcal{L}(\cdot))$ is indeed a closure operation and $\mathcal{N}(\mathcal{L}(N))$, called the *saturation* of N, is obtained by adding to N all the places that permit the execution of every word in $\mathcal{L}(N)$.[2]

2.3 Separation Problems

Applying the Galois connection (Equation 4) to an atomic Petri net $N = \{\mathbf{p}\}$ tells us that

$$L \subseteq \mathcal{L}(\{\mathbf{p}\}) \iff \mathbf{p} \in \mathcal{N}(L) \tag{5}$$

meaning that a vector $\mathbf{p} \in \mathbb{N}^{1+2n}$ is a region of L if and only if, viewed as an atomic Petri net, it allows the execution of every word in L. Note that $w \cdot e \in \mathcal{L}(\{\mathbf{p}\})$ if and only if $w \in \mathcal{L}(\{\mathbf{p}\})$ and $\langle \Psi(w), e\rangle \cdot \mathbf{p} \geq 0$. Hence for any $w \in \mathcal{L}(\{\mathbf{p}\})$ and $e \in E$ we say that *place* $\mathbf{p}$ *inhibits event* e *after reading* w when $\langle \Psi(w), e\rangle \cdot \mathbf{p} < 0$. Accordingly, for any prefix-closed language L, we let

[1] Note that $\langle \mathbf{v}, e\rangle$ determines both vector $\mathbf{v} \in \mathbb{N}^E$ and event $e \in E$ so that we shall allow ourselves to write for instance $(\exists \langle \mathbf{v}, e\rangle \in L^{\mathbf{C}}) \quad \Phi(\mathbf{v}, e)$ in place of $(\exists w \in E^*)(\exists e \in E) \quad w \cdot e \in L \;\wedge\; \Phi(\Psi(w), e)$, and accordingly to view $L^{\mathbf{C}}$ also as a set of vectors in $\mathbb{Z}^{1+2n}$, the rows of matrix $L^{\mathbf{C}}$.

[2] The following properties are direct consequences of the Galois connection: (1) L is a Petri net language, i.e. $(\exists N) \quad L = \mathcal{L}(N)$, if and only if $L = \mathcal{L}(\mathcal{N}(L))$; (2) if N is a synthesized net, i.e. $(\exists L) \quad N = \mathcal{N}(L)$, then $N = \mathcal{N}(\mathcal{L}(N))$ and it is the largest Petri net with the given language (i.e., N is saturated); (3) $\mathcal{L}(\mathcal{N}(L))$ is the least Petri net language containing L.

$L^{\mathbf{SP}} \subseteq \mathbb{N}^E \times E$ denote the set of vectors $\langle \mathbf{v}, e \rangle$ for which exists $w \in L$ such that $\Psi(w) = \mathbf{v}$ and $w \cdot e \notin L$. An element $\mathbf{sp} \in L^{\mathbf{SP}}$ is called a *separation problem for* L. We say that place $\mathbf{p} \in \mathbb{N}^{1+2n}$ *witnesses* the separation problem $\mathbf{sp} \in L^{\mathbf{SP}}$, or that the separation problem is solved by the place, when $\mathbf{sp} \cdot \mathbf{p} < 0$. Then a language L is a Petri net language, i.e. $L = \mathcal{L}(N)$ for some Petri net N, if and only if every instance of the separation problems in $L^{\mathbf{SP}}$ is solved by some place of N. Then $N \subseteq \mathcal{N}(L)$, i.e. N is formed from a subset of regions of L. [3]

Say that a set $R \subseteq \mathcal{N}(L)$ of regions is *complete* w.r.t. a prefix-closed language L if it witnesses all solvable instances of separation problems, i.e. for every $\langle \mathbf{v}, e \rangle \in L^{\mathbf{SP}}$, when $(\exists \mathbf{p} \in \mathcal{N}(L)) \quad \langle \mathbf{v}, e \rangle \cdot \mathbf{p} < 0$, then also $(\exists \mathbf{p} \in R) \quad \langle \mathbf{v}, e \rangle \cdot \mathbf{p} < 0$. Suppose that L is a Petri net language. $R \subseteq \mathcal{N}(L)$ is complete if and only if $\mathcal{L}(R) = L$. In particular if $R \subseteq N \subseteq \mathcal{N}(\mathcal{L}(N))$ then R is complete if and only if $\mathcal{L}(R) = \mathcal{L}(N)$, i.e. places in $N \setminus R$ are redundant places that can be dropped without modifying the language. If L is not a Petri net language and $R \subseteq \mathcal{N}(L)$ is a complete set of regions then the language $\mathcal{L}(R)$ is not necessarily the least language of a Petri net that contains L. It nevertheless satisfies $\mathcal{L}(R) \cap L \cdot E = \mathcal{L}(\mathcal{N}(L)) \cap L \cdot E$[4] which states that $\mathcal{L}(R)$ differs from the optimal solution $\mathcal{L}(\mathcal{N}(L))$ only after having strictly exited language L. We call such a solution a *suboptimal* Petri net solution for L.

2.4 Minimal Regions

Two regions that differ only by a non-negative multiplicative factor, namely $\mathbf{p}$ and $\mathbf{p}'$ with $\mathbf{p}' = k \cdot \mathbf{p}$ for some $0 \neq k \in \mathbb{N}$, provide equivalent places.[5] Thus one can without loss of generality consider vectors with rational entries and consider the cone of regions given by

$$\mathcal{R}(L) = \left\{ \mathbf{r} \in \mathbb{Q}^{1+2n} \;\middle|\; \mathbf{r} \geq 0 \;\wedge\; L^{\mathbf{C}}\mathbf{r} \geq 0 \right\} \tag{6}$$

This cone is generated by its finite set of *extremal rays*. Up to a multiplication by a non-negative integer one can assume that each extremal ray has integral entries. They constitute the set $\mathcal{R}_{\min}(L) = \{\mathbf{r_1}, \ldots, \mathbf{r_K}\}$ of *minimal regions* of L. We can restrict ourselves to minimal regions because minimal regions are witnesses to all solvable separation problems (as $\mathbf{sp} \cdot (\Sigma \lambda_i \cdot \mathbf{r_i}) < 0 \quad \text{with} \quad \lambda_i \geq 0 \implies (\exists i)\ \mathbf{sp} \cdot \mathbf{r_i} < 0$) and the net synthesized from minimal regions is language equivalent to the synthesized net: $\mathcal{L}(\mathcal{N}(L)) = \mathcal{L}(\mathcal{R}_{\min}(L))$. Indeed, by the same reasoning as above, if $\mathbf{p} = \Sigma \lambda_i \cdot \mathbf{r_i}$ inhibits event e after reading w, so does some minimal region. Hence $\mathcal{L}(\{\mathbf{p}\}) \subseteq \bigcap_i \mathcal{L}(\{\mathbf{r_i}\}) = \mathcal{L}(\mathcal{R}_{\min}(L))$. Thus if each separation problem can be solved by $R \subseteq \mathcal{R}_{\min}(L)$ then L is the language of the corresponding Petri net: $L = \mathcal{L}(R)$. Otherwise, $\mathcal{L}(\mathcal{R}_{\min}(L))$ is the least Petri net language that contains L.

[3] Indeed $L = \mathcal{L}(N) = \bigcap \{\mathcal{L}(\{\mathbf{p}\}) \mid \mathbf{p} \in N\}$ iff *(1)* $(\forall \mathbf{p} \in N) \quad L \subseteq \mathcal{L}(\{\mathbf{p}\})$, i.e. $\mathbf{p} \in \mathcal{N}(L)$ and *(2)* $(\forall w \in L)\,(\forall e \in E) \quad w \cdot e \notin L \implies (\exists \mathbf{p} \in N) \quad \langle \Psi(w), e \rangle \cdot \mathbf{p} < 0$.

[4] Indeed let $u \in L$, such that $u \cdot e \notin L$ and $u \cdot e \in \mathcal{L}(R) \setminus \mathcal{L}(\mathcal{N}(L))$. Then $\mathbf{sp} = \langle \Psi(u), e \rangle$ has a solution (in $\mathcal{N}(L)$) but not in R, a contradiction.

[5] in the sense that one place can be replaced by the other in any Petri net without modifying its behaviour.

3 Incrementality of the Synthesis

The Petri net synthesized from a log (a prefix-closed language $L \subseteq E^*$) is constructed from minimal regions that are associated with the extremal rays of the cone of regions (Equation 6). We show that this construction is incremental. Namely, it can be presented as an online process that updates the constructed Petri net model whenever new execution sequences are added to the log which are not executable in the current model. At each point in time the constructed Petri net model is such that its language is an over-approximation of the current log. For ease of presentation we decompose this process into three parts. A preprocessing phase turns the log into a so-called *specification graph* from which the set of linear constraints $L^{\mathbf{C}}$ characterizing the set of regions and the set $L^{\mathbf{SP}}$ of separation problems can be extracted. Then we compute the set of minimal regions based on $L^{\mathbf{C}}$. In a post-processing phase we extract a complete set of places from the full set of minimal regions via $L^{\mathbf{SP}}$.

3.1 Generating the Set of Linear Constraints and Separation Problems

The specification graph of a prefix closed language L defines an over-approximation $L^{\Psi,I}$ of L that should be part of the language of any intended Petri net realization of L. As such it specifies the generalization that process discovery is expected to perform. First, we define the *Parikh-Closure* L^{Ψ} of L that characterizes all Petri net models of L and then we refine it to $L^{\Psi,I}$ when additional information on cyclic behaviours are available. The specification graph allows to produce the set of linear constraints and the separation problems that are used to generate a Petri net model.

3.1.1 Parikh-Closure of a Language

The linear constraints that define the cone of regions are given by the vectors in $L^{\mathbf{C}}$:

$$\big(\forall \mathbf{p} \in \mathbb{N}^{1+2n}\big) \quad \mathbf{p} \in \mathcal{N}(L) \iff (\forall \mathbf{c} \in L^{\mathbf{C}}) \quad \mathbf{c} \cdot \mathbf{p} \geq 0 \tag{7}$$

Nonetheless if $w \cdot a$ and $w' \cdot a$ are two nonempty words of L where w and w' have the same Parikh image $\mathbf{v}$ then they lead to the same constraint $\langle \mathbf{v}, a \rangle \in L^{\mathbf{C}}$. This is due to the fact that, by Equation (1), a Petri net language is *Parikh-closed*, meaning that it is a prefix-closed language such that $(\forall w, w' \in L) \quad \Psi(w) = \Psi(w') \Rightarrow [(\forall e \in E) \quad w \cdot e \in L \Leftrightarrow w' \cdot e \in L]$. The *Parikh closure* L^{Ψ}, i.e. the least Parikh-closed language containing a prefix-closed language L, is given by the labelled transition system $TS^{\Psi}(L)$ with states $S = \{\Psi(w) \mid w \in L\}$, initial state $s_0 = \Psi(\varepsilon)$, and transitions $T = \big\{(\mathbf{v}, e, \mathbf{v} + e) \;\big|\; \langle \mathbf{v}, e \rangle \in L^{\mathbf{C}}\big\}$. Hence $\mathbf{v} \xrightarrow{e} \mathbf{v}'$ in $TS^{\Psi}(L)$ if and only if $\mathbf{v}' = \mathbf{v} + e$ and there is a $w \in E^*$ such that $\Psi(w) = \mathbf{v}$ and $w \cdot e \in L$.

A prefix-closed language and its Parikh closure define the same set of linear constraints, namely $L^{\mathbf{C}} = (L^{\Psi})^{\mathbf{C}}$, and thus $\mathcal{N}(L) = \mathcal{N}(L^{\Psi})$. By Equation (4) it

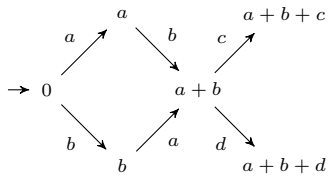

Figure 2: Abstraction $TS^{\Psi}(L)$ of the log $L = \mathrm{Pref}(\{abc, bad\})$. It defines the language $L^{\Psi} = \mathrm{Pref}(\{abc, bad, abd, bac\})$.

follows that $L^{\Psi} \subseteq \mathcal{L}(N)$ for every Petri net such that $L \subseteq \mathcal{L}(N)$. Note that the set of constraints $L^{\mathbf{C}}$ and the set of separation problems $L^{\mathbf{SP}}$ of a Parikh-closed language L are disjoint and determine each other.

3.1.2 Taking cyclic behaviours into account

The transition system $TS^{\Psi}(L)$ allows to produce each constraint $\mathbf{c} \in L^{\mathbf{C}}$ and separation problem $\mathbf{sp} \in L^{\mathbf{SP}}$ only once and its size is expected to be much smaller than the size of the log L. It can be seen as an approximation of the reachability graph of the Petri net to be synthesized. Nonetheless, this transition system is acyclic whereas the reachability graph of the Petri net synthesized from the log will usually contain cycles. We obtain a much more accurate specification of the expected model if we have knowledge about the cyclic behaviours. For instance one often considers that a complete transaction, or treatment of a case, in a workflow system does not affect the internal state of the system. This property, called *soundness* [12], means that after the treatment of a case the system recovers its initial state where it can initiate a new case. Each execution sequence is related to some case and we can assume to be able to distinguish a completed execution from a partial execution of a case. For that purpose let us assume that L is the prefix-closure of its set $L_{\max}$ of maximal elements (for the prefix order relation). Words in $L_{\max}$ would correspond to complete executions of a case and their Parikh vectors should be T-invariants for the Petri net to be synthesized, whose prefix language should therefore contain the prefix-closure of the ω-language $L^{\omega}_{\max}$.

We obtain the specification graph $TS^{\Psi,I}(L)$ as a quotient of $TS^{\Psi}(L)$ using the linear transformation induced by the Gaussian elimination. More precisely we let $EQ = EQ_1 \cup EQ_2$ where initially EQ_1 consists of equations $\mathbf{v} = 0$ for each generator $\mathbf{v}$ of I and $EQ_2 = \emptyset$ and we transform this system into equivalent systems of equations until $EQ_1 = \emptyset$. At each stage we choose an equation $\sum \lambda_i a_i = 0$ in EQ_1 and an index i such that $\lambda_i \neq 0$. We remove this equation from EQ_1 and add it to EQ_2 in the form $a_i = \sum_{j \neq i} \frac{-\lambda_j}{\lambda_i} a_j$, and replace all occurrences of a_i in all the other equations of EQ by the corresponding value. We suppress equations of the form $0 = 0$ that may appear in EQ_1 on that occasion. One ends up with a system of equations of the form $a_i = \sum_{j \in J} q_{i,j} a_j$ for every $i \notin J$ with rational coefficients $q_{i,j}$. This system can then be presented as equations with integral coefficients consisting of equations $a_i = \sum_{j \in J} n_{i,j} a'_j$ for $i \notin J$ and $a_j = n_j a'_j$ for $j \in J$. It defines a linear transformation π_I from $\mathbf{Z}^E$ to $\mathbf{Z}^{E'}$ where $E' = \left\{ a'_j \mid j \in J \right\}$. Two states of $TS^{\Psi}(L)$, or more generally

two vectors of $\mathbb{Z}^E$, are said to be *equivalent modulo* I, in notation $\mathbf{v} \equiv_I \mathbf{v}'$ when $\pi_I(\mathbf{v}) = \pi_I(\mathbf{v}')$. Thus the resulting quotient graph $TS^{\Psi,I}(L)$ embeds into the lattice $\mathbf{Z}^{E'}$.

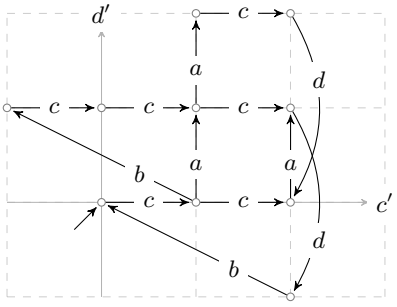

Figure 3: Illustration of the construction of $TS^{\Psi,I}(L)$ with $L_{\max} = \{ccadb, cbcccdb, caacdadb\}$. I is generated by the vectors $a+b+2c+d$, $2b+4c+d$, and $3a+b+2c+2d$. By Gaussian elimination the system $I = 0$ is transformed into the following system: $a = d'$, $b = -2c' + d'$, $c = c'$, and $d = -2d'$.

Say that a prefix-closed language L is I-saturated where I is a finitely generated subgroup of $\mathbb{Z}^E$ if $(\forall w, w' \in L) \quad \Psi(w') \equiv_I \Psi(w) \implies w \equiv_L w'$ where $w \equiv_L w' \iff w^{-1}L = (w')^{-1}L$ is the Nerode equivalence. The language $L^{\Psi,I}$ of $TS^{\Psi,I}(L)$ is the least I-saturated language containing L, and one has $L^{\Psi,I} \subseteq \mathcal{L}(N) \iff N \subseteq \mathcal{N}(L)$ for every Petri net N such that each element of I is an invariant of N, i.e. such that $\mathbf{I} \cdot N = 0$ where $\mathbf{I}$ is the matrix with row vectors $\langle \mathbf{v} \rangle = (0; \mathbf{v}, -\mathbf{v})$ for $\mathbf{v}$ ranging over a set of generators for I.

Note that, even though one considers only the invariants associated with the completed executions, by Gaussian elimination we can also retrieve invariants associated with "inner" cyclic computations. For instance if ad and $abcd$ are both in $L_{\max}$ we deduce that $b + c$ is an invariant and thus infer that any word in $a(bc)^*d$ belongs to every Petri net model that fits the log. More generally if $L_{\max} \subseteq L'_{\max}$ is a finite but sufficiently large and representative subset of the set $L'_{\max}$ of maximal execution sequences of a Petri net (with language $L'^{\omega}_{\max}$) then $L_{\max}$ will allow us to infer all the invariants of the net and by executing words in $L_{\max}$ in the model we will take all the transitions of its graph of reachable markings. In that case the specification graph $TS^{\Psi,I}(L)$ will be isomorphic to the graph of reachable markings.

3.1.3 Incremental computation of the specification graph

The specification graph $SG = TS^{\Psi,I}(L)$ can be constructed incrementally. Indeed, when a new sequence w is added to $L_{\max}$ we "run" this sequence w in SG adding as many new states and transitions as necessary. Each new transition induces an additional constraint in $L^{\mathbf{C}}$ (which should be withdrawn from the set of separation problems) and each new state s induces the additional separation problems associated with the events e that are not allowed in this new state.[6]

[6]More precisely the new states are $\Delta(S, w) = \{\pi_I(\Psi(w')) \mid w' \preceq w\} \setminus S$, and the new transitions are $\Delta(T, w) = \{(\pi_I(\Psi(w')), e, \pi_I(\Psi(w') + e) \mid w' \cdot e \preceq w\} \setminus T$ where $w' \preceq w$ means that w' is some prefix of w. A new transition (s, e, s') induces a new constraint $\langle \mathbf{v}, e \rangle$ and each new state s and event e not allowed in s induces a new separation problem $\{\langle \mathbf{v}, e \rangle \mid e \in E\}$ where $\mathbf{v}$ is the Parikh image of some path from the initial state to state s.

Finally we also add the constraint $\Psi(w) = 0$ to $L^{\mathbf{C}}$ (by introducing the two inequalities $\Psi(w) \geq 0$ and $-\Psi(w) \geq 0$), and we apply Gaussian elimination to update the representation of the specification graph. It is important to note that the production of constraints is monotonous (we add constraints). However, at each stage one can produce new separation problems and suppress some others at the same time.

3.2 Computing the Minimal Regions

The computation of the set of extremal rays of the cone of regions is given in Algorithm (1).

Algorithm 1 Computation of the minimal regions

Invariant $\mathcal{R}(L) = \left\{\mathbf{x} \in \mathbb{Q}^{1+2n} \;\middle|\; \mathbf{x} \geq 0 \;\wedge\; L^{\mathbf{C}}\mathbf{x} \geq 0\right\} = \{\mathbf{y}R \mid \mathbf{y} \geq 0\}$
where the columns of R are the extremal rays of $\mathcal{R}(L)$
Initially $L^{\mathbf{C}} = \emptyset$ and $R = I_{1+2n}$
if some update of the log induces new constraints **then**
  $L^{\mathbf{C}} \leftarrow L^{\mathbf{C}} \cup C$ where C are new constraints
  $\mathcal{R}(L) = \left\{\mathbf{x} \in \mathbb{Q}^{1+2n} \mid \exists \mathbf{y} \geq 0 \quad \mathbf{x} = \mathbf{y}R \;\wedge\; C\mathbf{x} \geq 0\right\}$
  for all constraint (row) $\mathbf{c}$ of C **do**
    We update R by intersecting the cone $\mathcal{R} = \{\mathbf{y}R \mid \mathbf{y} \geq 0\}$
    with the half-space $\mathcal{H}_{\mathbf{c}} = \{\mathbf{x} \in \mathbb{Q}^n \mid \mathbf{c} \cdot \mathbf{x} \geq 0\}$
    $R^{>0} \leftarrow \{\mathbf{r} \in R \mid \mathbf{c} \cdot \mathbf{r} > 0\}$
    $R^{<0} \leftarrow \{\mathbf{r} \in R \mid \mathbf{c} \cdot \mathbf{r} < 0\}$
    $R \leftarrow R \setminus R^{<0}$
    for all $\mathbf{r}^+ \in R^{>0}$ and $\mathbf{r}^- \in R^{<0}$ with $\mathbf{r}^+$ and $\mathbf{r}^-$ adjacent in $\mathcal{R}$ **do**
      let $\mathbf{r} = (-\mathbf{c} \cdot \mathbf{r}^-)\mathbf{r}^+ + (\mathbf{c} \cdot \mathbf{r}^+)\mathbf{r}^-$
      $\mathbf{r}$ is the ray of the face generated by $\mathbf{r}^+$ and $\mathbf{r}^-$ that lies in the hyperplane $\left\{\mathbf{x} \in \mathbb{Q}^{1+2n} \mid \mathbf{c} \cdot \mathbf{x} = 0\right\}$
      $R \leftarrow R \cup \{\mathbf{r}\}$
    end for
  end for
end if

It is a slight adaptation of the corresponding algorithm given in [3, p. 208] that emphasizes its incrementality.

In the course of this algorithm we have to compute the adjacency relation between the extremal rays. We recall the following alternative characterization of this relation: Given a matrix A, the corresponding *polyhedral cone* is $\mathcal{C}(A) = \{\mathbf{x} \in \mathbb{Q}^n \mid A\mathbf{x} \geq 0\}$. Two extremal rays $\mathbf{r}_1$ and $\mathbf{r}_2$ are adjacent in $\mathcal{R} = \{\mathbf{y}R \mid \mathbf{y} \geq 0\}$ if and only if there does not exist another extremal ray $\mathbf{r}_3$ (i.e. distinct from $\mathbf{r}_1$ and $\mathbf{r}_2$) with $\mathrm{supp}_A(\mathbf{r}_3) \subseteq \mathrm{supp}_A(\mathbf{r}_1) \cup \mathrm{supp}_A(\mathbf{r}_2)$ where $\mathrm{supp}_A(\mathbf{r}) = \{\mathbf{a} \in A \mid \mathbf{a} \cdot \mathbf{r} > 0\}$. If A' is obtained from A by adding a new

constraint $\mathbf{c}$ and $\mathbf{r} \in R \setminus R^{<0}$ then

$$\begin{array}{rcll} \mathrm{supp}_{A'}(\mathbf{r}) & = & \mathrm{supp}_A(\mathbf{r}) \cup \{\mathbf{c}\} & \text{if} \quad \mathbf{r} \in R^{>0} \\ & = & \mathrm{supp}_A(\mathbf{r}) & \text{if} \quad \mathbf{c} \cdot \mathbf{r} = 0 \end{array}$$

We further compute $\mathrm{supp}_{A'}(\mathbf{r})$ for the newly introduced extremal rays. We then use the above characterization to update the adjacency relation, knowing that two extremal rays $\mathbf{r}_1$ and $\mathbf{r}_2$ in $R \setminus R^{<0}$ are adjacent in $\mathcal{C}(A)$ if and only if they are adjacent in $\mathcal{C}(A')$.

In addition to the above incremental computation of the set of the minimal regions we also need to update the value of a Boolean matrix **SEP** that records those regions solving each separation problem. The rows of matrix **SEP** are indexed by the extremal rays $\mathbf{r}$, its columns by the separation problems $\mathbf{sp} \in L^{\mathbf{SP}}$ and $\mathbf{SEP}(\mathbf{r}, \mathbf{sp}) = \mathbf{true} \iff \mathbf{r} \cdot \mathbf{sp} < 0$. When language L is upgraded to $L' \supset L$ we suppress (respectively add) the rows corresponding to the extremal rays that disappear (resp. appear) according to Algorithm (1). Similarly we suppress the columns corresponding to the elements of $(L')^{\mathbf{C}} \cap L^{\mathbf{SP}}$ (separation problems that become constraints) and add new columns associated with $(L')^{\mathbf{SP}} \setminus L^{\mathbf{SP}}$ (new separation problems). Finally we compute the entries associated either with a new extremal ray or a new separation problem.

3.3 Extracting a Petri Net Model

The method can be adapted to *pure* Petri net by encoding a place p as the vector $\mathbf{p} = (M_0(p); {}^\bullet p - p^\bullet)$, the definition of matrix $L^{\mathbf{C}}$ is not modified but we replace the constraint $\mathbf{r} \geq 0$ in the definition of the cone of regions (Equation 6) by $r_0 \geq 0$. The Petri net associated with the whole set of minimal regions provides the model in this class whose language is the least over-approximation of the log. However as illustrated by the example given in Figure 4 this solution may contain redundant places.

Simple heuristics can be put forward to construct a complete subset of minimal regions. For instance one can iteratively select some region r that solves the largest number of separation problems, and suppress from matrix **SEP** the columns that corresponds to separation problems solved by r, and the row associated with r. The procedure stops when no separation problems remains (matrix **SEP** is empty) or if none of the remaining separation problems can be solved by the remaining regions (all entries of **SEP** are zeroes). Even though in practice this algorithm performs quite well it does not necessarily produce the simplest Petri net over-approximation of the log. We can improve the constructed model by applying the following linear programming problem to that initial solution. Let the *weight* $w(\mathbf{r}) \in \mathbb{N}$ of a region $\mathbf{r} \in \mathcal{R}(L)$ describe how complicated the region $\mathbf{r}$ is. For example, it can be measured by its initial value together with the sum of its flow arc inscriptions: $w(\mathbf{r}) = \sum_{i=0}^{2n} r_i$. For $\mathbf{r} \in \mathcal{R}(L)$, the variable $x_{\mathbf{r}} \in \{0, 1\}$ has value 1 iff the region $\mathbf{r}$ is part of the solution $R = \{\mathbf{r} \in \mathcal{R}(L) \mid x_{\mathbf{r}} = 1\}$. Similarly, for $\mathbf{sp} \in L^{\mathbf{SP}}$, the variable $y_{\mathbf{sp}} \in \{0, 1\}$ has value 1 only if some region $\mathbf{r} \in R$ solves $\mathbf{sp}$. A pair $(\mathbf{x}, \mathbf{y})$ is a *potential solution* if it satisfies $\mathbf{y} \leq \mathbf{x} \cdot \mathbf{SEP}$, which means that for each separation problem

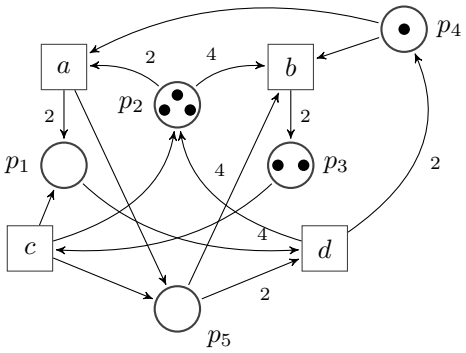

Figure 4: Petri net associated with the set of minimal *pure* regions computed from the specification graph of Fig. 3. Places p_4 and p_5 are redundant and can be removed. The corresponding regions do not solve any separation problem that cannot already be solved by the other three places. The restriction to the set $\{p_1, p_2, p_3\}$ is the Petri net of Fig. 1 whose marking graph can be compared to the specification graph of Fig. 3. This Petri net thus provides the least over-approximation of the log with a pure Petri net language. However, without the restriction to pure Petri nets, this specification graph would be realisable exactly.

$\mathbf{sp}$ which is supposed to be solved ($y_{\mathbf{sp}} = 1$), there is indeed a region $\mathbf{r}$ in the solution ($x_{\mathbf{r}} = 1$) that solves it ($\mathbf{SP}(\mathbf{r}, \mathbf{sp}) = 1$). The weight of a potential solution $(\mathbf{x}, \mathbf{y})$ is then the value $\sum_{x_{\mathbf{r}}=1} w(\mathbf{r}) = \mathbf{x} \cdot \mathbf{w}$. Solving the optimization problem

$$\text{min\_weight}(\mathbf{y}) = \arg \min_{\mathbf{x}} \{\mathbf{x} \cdot \mathbf{w} \mid \mathbf{y} \leq \mathbf{x} \cdot \mathbf{SEP}\} \tag{8}$$

leads to a set of regions with minimal weight that solves every separation problem $\mathbf{sp}$ such that $y_{\mathbf{sp}} = 1$. In particular if vector $\mathbf{y}$ encodes the set of solvable separation problems we obtain a complete set of regions with a minimal weight. By doing so we put emphasis on *precision*, by finding optimal models that provide a minimal over-approximation and then on *simplicity* by selecting among these models one with a minimal weight. Priority is thus given to precision over simplicity.

We may be interested to find alternative models that are less precise but simpler. For that purpose we quantify the penalty of not solving a separation problem by introducing the *cost* $c(\mathbf{sp}) \in \mathbb{N}$ of a separation problem $\mathbf{sp} \in L^{\mathbf{SP}}$. For instance, one can count the number of words in the log that lead to some state s of the specification graph. If only few words reach state s, then the cost of a separation problem $\mathbf{sp} = \langle \mathbf{v}, e \rangle$ is low. This is in particular the case when state s has just been introduced due to an update of the log. The cost of a solution $(\mathbf{x}, \mathbf{y})$ is given by $\sum_{y_{\mathbf{sp}}=0} c(\mathbf{sp}) = (\mathbf{1} - \mathbf{y}) \cdot \mathbf{c}$. We are interested in solutions $(\mathbf{x}, \mathbf{y})$ which have a low weight and a low cost. Since these two goals are conflicting, there is no unique best solution, but a Pareto front of pairwise incomparable solutions. To find all solutions in the Pareto front, we solve a series of integer linear programming problems. We start from a complete set of regions with an optimal weight computed as indicated above. Thus $\mathbf{y}_0$ encodes

the set of solvable separation problems and $\mathbf{x}_0 = \text{min\_weight}(\mathbf{y}_0)$ is a complete set of regions of minimal weight. Given a solution $(\mathbf{x}_i, \mathbf{y}_i)$, we can find the next solution $(\mathbf{x}_{i+1}, \mathbf{y}_{i+1})$ that is part of the Pareto front and has lower weight via:

$$\mathbf{y}_{i+1} = \arg\min_{\mathbf{y}} \{(\mathbf{1} - \mathbf{y}) \cdot \mathbf{c} \mid (\exists \mathbf{x}) \quad \mathbf{y} \leq \mathbf{x} \cdot \mathbf{SEP} \text{ and } \mathbf{x} \cdot \mathbf{w} < \mathbf{x}_i \cdot \mathbf{w}\} \tag{9}$$

and $\mathbf{x}_{i+1} = \text{min\_weight}(\mathbf{y}_{i+1})$. Solving the previous problem with $i = 0$ provides a solution $(\mathbf{x}_1, \mathbf{y}_1)$ that has slightly higher cost than $(\mathbf{x}_0, \mathbf{y}_0)$, but its weight is better. Iterating this procedure provides a series of points on the Pareto front. This can be repeated until we reach the solution with an empty set of regions and the maximal cost, $\text{max\_cost} = \sum c(\mathbf{sp})$, at which point the complete Pareto front was found. More realistically we will stop when the cost reaches a threshold indicating that the solution generalises too much, or we will limit ourselves to the first two or three solutions.

Note that this process is incremental, because algorithms that solve integer linear optimisation problems work by having a current solution that is improved upon. Thus, if the problem to solve changes, the optimal solution that was previously found provides a good initial solution to start from.

4 Conclusion

As mentioned in the introduction Process Discovery puts emphasis on the following criteria: *Replay-fitness* states that the model is able to reproduce the executions in the log, *Generalization* is the ability to learn behaviours of the system from the set of samples found in the log, *Precision* stipulates that the generated model does not generalize too much, and finally *Simplicity* states that the model has a small size and complexity.

According to the principle of parsimony (a.k.a. Occam's Razor) generalization makes sense when the produced model provides the simplest explanation for the observed behaviour. Hence it should maximize Replay-fitness and Simplicity. The log does not contain all the behaviors of the intended model mainly because of concurrency and the existence of cyclic behaviours. The construction of the specification graph takes these two aspects into account to obtain the optimal level of generality: it characterizes the Petri net models that fit to the log and whose invariants contain the Parikh images of cycles. A Petri net system synthesized from a set of minimal regions always provides an over-approximation of the language given as input. We thus have *perfect replay-fitness*. The approximation is the more precise as we consider more regions. Hence we obtain the more precise model by considering the whole set of minimal regions. Nonetheless optimality w.r.t. language inclusion does not guarantee simplicity of the model. For that purpose we presented some heuristics to extract a *complete* subset of minimal regions that minimize the size and complexity of the model. Completeness means that the selected regions witness all solvable instances of separation problems. For that reason it produces a Petri net with a *suboptimal* language. The optimal language is obtained when all separations problems can

be solved. Then one can use integer linear programming to find such a sub-optimal solution with a minimal weight, hence a model as simple as possible amongst this set of best precise candidates. Finally one can also generate some alternative solutions that are simpler but less precise. To sum up the presented method provides a good compromise between the ability to infer behaviours of the system from the set of observations while ensuring a parsimonious model, in terms of fitness, precision and simplicity.

All used algorithms are incremental which means that one can modify the produced model when new observations are reported without reconstructing the model from scratch. Incrementality has been overlooked in the context of process discovery because a log is usually a very large, and hopefully representative, set of executions sequences and therefore we do not expect to gain new information from the observation of extra runs. We think that incrementality is nevertheless an important feature when the process under observation can evolve and produce new behaviours due to some modifications in the organization. In that respect, it might also be interesting to account for behaviours that disappear. For that purpose we might choose to discard executions sequences in log after a certain amount of time. However, in that situation, reminiscent of the *the logic of theory change* [14], we lose monotony and it is not at all clear that our method can easily be amended for that purpose. This question may be the subject of future research.

References

[1] Ehrenfeucht A, Rozenberg G. Partial 2-structures; Part I: Basic Notions and the Representation Problem. Acta Informatica. 1990;27:315–342. doi:10.1007/BF00264611.

[2] Ehrenfeucht A, Rozenberg G. Partial 2-structures; Part II: State Spaces of Concurrent Systems. Acta Informatica. 1990;27:343–368. doi:10.1007/BF00264612.

[3] Badouel E, Bernardinello L, Darondeau P. Petri Net Synthesis. Springer; 2015. doi:10.1007/978-3-662-47967-4.

[4] Badouel E, Bernardinello L, Darondeau P. Polynomial Algorithms for the Synthesis of Bounded Nets. In: Mosses PD, Nielsen M, Schwartzbach MI, editors. TAPSOFT'95. vol. 915 of Lecture Notes in Computer Science. Springer; 1995. p. 364–378. doi:10.1007/3-540-59293-8_207.

[5] Darondeau P. Deriving Unbounded Petri Nets from Formal Languages. In: CONCUR'98 Concurrency Theory. vol. 1466 of Lecture Notes in Computer Science. Springer; 1998. p. 533–548. doi:10.1007/BFb0055646.

[6] Lorenz R, Juhás G, Mauser S. How to synthesize nets from languages - A Survey. In: Proceedings of the 2007 Winter Simulation Conference. IEEE; 2007. p. 637–647. doi:10.1109/WSC.2007.4419657.

[7] Carmona J, Cortadella J, Kishinevsky M. A region-based algorithm for discovering Petri nets from event logs. In: M Dumas MS M Reichest, editor. BPM 2008. vol. 5240 of Lecture Notes in Computer Science. Springer; 2008. p. 358–373. doi:10.1007/978-3-540-85758-7_26.

[8] Best E, Devillers R. Synthesis and reengineering of persistent systems. Acta Informatica. 2015;52(1):35–60. doi:10.1007/s00236-014-0209-7.

[9] Best E, Devillers R. Characterisation of the state spaces of marked graph Petri nets. Information and Computation. 2017;253:399–410. doi:10.1016/j.ic.2016.06.006.

[10] van der Werf JMEM, van Dongen BF, Hurkens CAJ, Serebrenik A. Process discovery using integer linear programming. Fundamenta Informaticae. 2009;94(3-4):387–412. Available from: `http://dl.acm.org/citation.cfm?id=1662594.1662600`.

[11] van Zelst SJ, van Dongen BF, van der Aalst WMP. ILP-Based Process Discovery Using Hybrid Regions. In: Proceedings of ATAED 2015; 2015. p. 47–61. Available from: `http://ceur-ws.org/Vol-1371/paper04.pdf`.

[12] van der Aalst WMP. Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer; 2001. doi:10.1007/978-3-642-19345-3.

[13] van der Aalst WMP. Process Mining - Data Science in Action. Springer; 2016. doi:10.1007/978-3-662-49851-4.

[14] Alchourrón CE, Gärdenfors P, Makinson D. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. J Symb Log. 1985;50(2):510–530. doi:10.2307/2274239.