Process Mining meets Abstract Interpretation

J. Carmona and J. Cortadella

Universitat Politècnica de Catalunya, Spain

Abstract. The discovery of process models out of system traces is an interesting problem that has received significant attention in the last years. In this work, a theory for the derivation of a Petri net from a set of traces is presented. The method is based on the theory of abstract interpretation, which has been applied successfully in other areas. The principal application of the theory presented is Process Mining, an area that tries to incorporate the use of formal models both in the design and use of information systems.

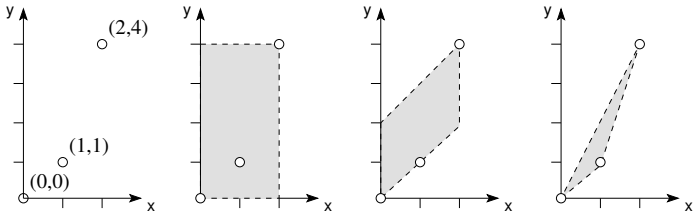
1 Introduction

Traces are everywhere: from information systems that store their continuous executions, to any type of health care applications that record each patient's history. The transformation of a set of traces into a mathematical model that can be used for a formal reasoning is therefore of great value which can save money or even human lives.

This paper proposes methods to build a process model representing the causal relations between the events in the trace, i.e., whether the event a occurs before b and after c or d. The goal is to construct a graph modeling all these orderings in a concise form. Among many of the graph formalisms that exist nowadays, we have selected Petri nets (PN) [15] for representing a set of traces. The reasons for this selection are: sound mathematical model, clear semantics, succinctness, ability of representing concurrent and conflict behavior among others.

The problem of deriving a PN out of a set of traces (called *log*) is one of the main areas of Process Mining [19]. More concretely, the goal is to obtain a PN whose behavior contains all the traces in the log, but maybe more. Within this area, several algorithms have been proposed to accomplish this task [4,5,20], most of them based on the theory of regions [11]. Informally, the theory of regions tries to map structures in the state-based or language-based representation of a system into *places* of the derived PN. However, given the well-known *state explosion problem*, algorithms that are defined at the level of the states will suffer when dealing with large systems exhibiting a high degree of concurrency.

Abstract interpretation [9] is a generic approach for the static analysis of complex systems. The underlying notion in abstract interpretation is that of *upper approximation*: to provide an abstraction of a complex behavior with less details. A property about a system such as an invariant is in some way an abstraction: it represents all the states of the system that satisfy the property.



Interval	**Octagon**	**Convex polyhedron**
$0 \leq x \leq 2$	$0 \leq x \leq 2$	$y - x \geq 0$
$0 \leq y \leq 4$	$0 \leq y - x \leq 2$	$x - 2y \geq 0$
		$3x - y \leq 2$

Fig. 1. Approximating a set of values (left) with several abstract domains

Intuitively, abstract interpretation defines a procedure to compute an upper approximation for a given behavior of a system. This definition guarantees (a) the termination of the procedure and (b) that the result is conservative. An important decision is the choice of the kind of upper approximation to be used, which is called the *abstract domain*. For a given problem, there are typically several abstract domains available. Each abstract domain provides a different trade-off between precision (proximity to the exact result) and efficiency.

There are many problems where abstract interpretation can be applied, several of them oriented towards the compile-time detection of run-time errors in software. For example, some analysis based on abstract interpretation can discover numeric invariants among the variables of a program. Also, it has been applied to extract invariants from a PN [7]. Several abstract domains can be used to describe the invariants: intervals [8], octagons [14], convex polyhedra [10], among others. These abstract domains provide different ways to approximate sets of values of numeric variables. For example, Figure 1 shows how these abstract domains can represent the set of values of a pair of variables x and y.

In this work we present an approach for deriving a PN from a log, based on the theory of abstract interpretation. The contributions can be summarized in: 1) a theory for deriving PNs out of a set of traces, 2) a technique to allow for the partitioning of the set of events into groups. The relations inside the groups and between groups can be detected and the corresponding causalities computed, 3) a sampling strategy that can be applied to detect the relations on a small set of instances instead of the whole set, and 4) a prototype tool implementing all the theory of the paper.

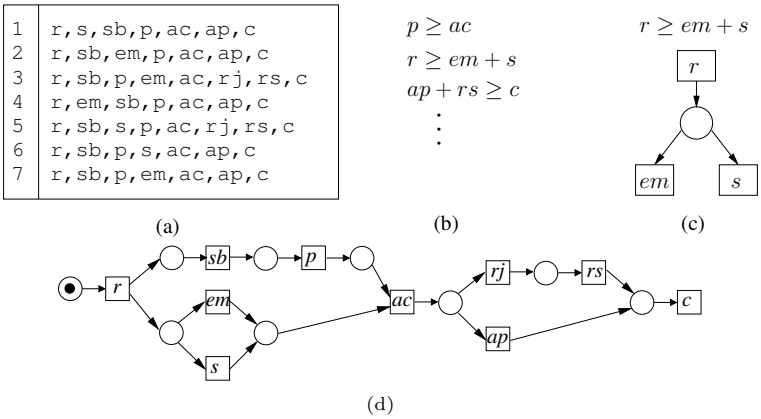


Fig. 2. Derivation of PNs using abstract interpretation: (a) log, (b) some invariants obtained, (c) from invariants to PN arcs, (d) mined Petri net.

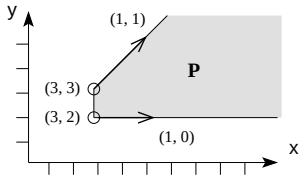
1.1 An introductory example

Let us provide a simple example to illustrate the theory of this paper. The example is taken from [17] and considers the process of handling customer orders. The starting point in Process mining is a set of traces representing the log of a system. In our example, the log contains seven traces with the following activities: *r=register*, *s=ship*, *sb=send_bill*, *p=payment*, *ac=accounting*, *ap=approved*, *c=close*, *em=express_mail*, *rj=rejected*, and *rs=resolve*. Part of these traces is shown in Figure 2(a), whilst Figure 2(b) shows some invariants that have been extracted from these traces using the theory of abstract interpretation. These inequalities can be obtained under the domain of convex polyhedra (see Figure 1), and relate the number of occurrences between events, e.g., $r \geq em + s$ indicates that the number of occurrences of r is always greater or equal than the sum of occurrences of em and s. Each invariant can be converted into a set of arcs in a PN, as it is shown in Figure 2(c). The final PN that covers all the traces in the log is presented in Figure 2(d) (see Section 2.1 for the formal semantics of a PN). It accepts the language defined by the expression[1]: $r;(sb;p)||(em|s);ac;(rj;rs)|ap;c$, where $||$, $|$ and ; denote interleaving, union and concatenation operators.

2 Preliminaries

Some mathematical notation is provided for the understanding of the paper. Given a set T, we denote $\mathcal{P}(T)$ as the powerset over T, i.e. the set of possible subsets of elements of T. A sequence $\sigma \in T^*$ is a called *trace*. Given a trace $\sigma = t_1, t_2, \ldots, t_n$, and a natural number $0 \leq k \leq n$, the trace $t_1, t_2, \ldots, t_k$ is

[1] For the reader not familiar with Petri nets: a transition (box) in a PN is enabled if every input place (circle) holds a token (black dot). If enabled, the transition can fire, removing tokens from its input places and adding tokens to its output places.



System of generators

$$P = \{\lambda_1 \cdot (3,3) + \lambda_2 \cdot (3,2) + \mu_1 \cdot (1,1) + \mu_2 \cdot (1,0) \mid \lambda_1 \geq 0, \lambda_2 \geq 0, \mu_1 \geq 0, \mu_2 \geq 0, \lambda_1 + \lambda_2 = 1\}$$

System of constraints

$$P = \{(x,y) \mid (y \geq 2) \wedge (x \geq 3) \wedge (x - y \geq 0)\}$$

Fig. 3. An example of a convex polyhedron (shaded area) and its double description

called the *prefix* of length k in σ. Given a set of traces L, we denote $Pref(L)$ the set of all prefixes for traces in L. Finally, given a trace σ, $\#(\sigma, e)$ computes the number of times that event e occurs in σ.

2.1 Logs and Petri Nets

Definition 1 (Log). *A log over a set of activities T is a set $L \in \mathcal{P}(T^*)$.*

Definition 2 (Petri net [15]). *A Petri net is a tuple (P, T, F, M_0) where P and T represent finite sets of places and transitions, respectively, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^{|P|}$ defines the initial state of the system.*

The sets of input and output transitions of place p in PN N are denoted by ${}^\bullet p$ and $p^\bullet$, respectively. A transition $t \in T$ is *enabled* in a marking M if $\forall p \in P : M[p] \geq F(p,t)$. Firing an enabled transition t in a marking M leads to the marking M' defined by $M'[p] = M[p] - F(p,t) + F(t,p)$, for $p \in P$, and is denoted by $M \xrightarrow{t} M'$. The set of all markings reachable from the initial marking m_0 is called its *Reachability Set*. The *Reachability Graph* of PN ($\mathsf{RG}(\mathsf{PN})$) is an automaton in which the set of states is the Reachability Set, the arcs are labeled with the transitions of the net and an arc (m_1, t, m_2) exists if and only if $m_1 \xrightarrow{t} m_2$. We use $L(\mathsf{PN})$ as a shortcut for $L(\mathsf{RG}(\mathsf{PN}))$. Finally, a place p in a PN is *redundant* if its removal does not changes $L(\mathsf{PN})$. Figure 2(d) contains an example of a PN such that $\sigma = r, s, sb, p, ap, c \in L(\mathsf{PN})$.

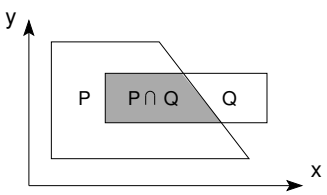
2.2 Convex Polyhedra

As suggested in Section 1.1, the convex polyhedra domain provides the necessary inequalities for the purposes of this paper. It can be described as the set of solutions of a set of *linear inequality constraints* with rational ($\mathbb{Q}$) coefficients. Let P be a polyhedron over $\mathbb{Q}^n$, then it can be represented as the solution to the system of m inequalities $P = \{X | AX \leq B\}$ where $A \in \mathbb{Q}^{m \times n}$ and $B \in \mathbb{Q}^m$. Convex polyhedra can also be characterized in a *polar* representation by means of a *system of generators*, i.e. as a linear combination of a set of vertices V (points) and a set of rays R (vectors). Figure 3 exemplifies this double description.

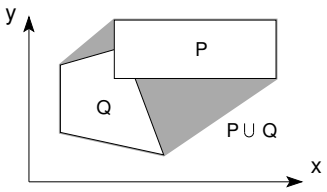
The fact that there are two representations is important, because several of the operations for convex polyhedra are computed very efficiently when the

proper representation is available. There are efficient algorithms [6,10] that translate one representation into the other. Also, the dual representations can be used to keep a *minimal* description, removing redundant constraints and generators.

The domain of convex polyhedra provides the operations required in abstract interpretation. In this paper, we will mainly use the following two operations:



Meet ($\cap$): Given convex polyhedra P and Q, computes $R = P \cap Q$. Notice that this operation is exact, e.g., the intersection of two convex polyhedra is always a convex polyhedra, implying that R does not contain any point outside $P \cap Q$.



Join ($\cup$): Given convex polyhedra P and Q, computes $R = P \cup Q$. Unfortunately the union of convex polyhedra is not necessarily a convex polyhedron. Therefore, the union of two convex polyhedra is approximated by the *convex hull*, the smallest convex polyhedron that includes both operands. The example on the left shows in gray the zone added by computing the convex hull of P and Q.

3 From logs to Petri nets via extraction of invariants

This section will set the basis for the approach presented in this paper. The underlying idea can be stated informally: for each trace of the log and each prefix of the trace, a vector describing the number of firings of each event for the prefix is computed. All these vectors are then inserted as n-dimensional points in the theory of convex polyhedra, where n is the number of events considered. Finally, a polyhedra is computed such that contains all these points, and its set of constraints represents invariants for the system.

3.1 Derivation of invariants from logs

We introduce the main element to link traces from a log L and convex polyhedra:

Definition 3 (Parikh vector). *Given a trace* $\sigma \in \{t_1, t_2, \ldots, t_n\}^*$*, the* Parikh vector *of* σ *is defined as* $\widehat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \ldots, \#(\sigma, t_n))$.

Any component of a Parikh vector can be seen as a constraint for the n-dimensional point that it defines. Hence, a Parikh vector $\widehat{\sigma} = (\#(\sigma, t_1), \#(\sigma, t_2), \ldots, \#(\sigma, t_n))$ can be seen as the following polyhedron:

$$P_{\widehat{\sigma}} = (x_1 = \#(\sigma, t_1)) \ \cap \ (x_2 = \#(\sigma, t_2)) \ \cap \ \ldots \ \cap \ (x_n = \#(\sigma, t_n))$$

where each variable x_i denotes the number of occurrences of t_i in σ. For each prefix σ of a trace in L, a convex polyhedra $P_{\widehat{\sigma}}$ can be obtained. Given all possible

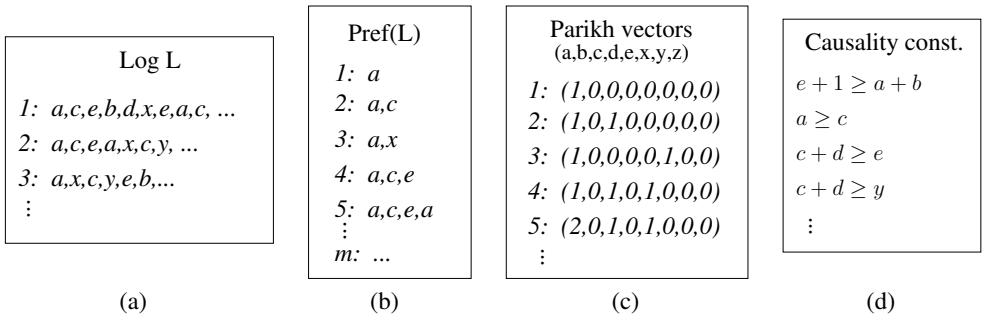


Fig. 4. From traces in the log to invariants: (a) Initial log, (b) corresponding m prefixes of the log, (c) Parikh vectors associated to the prefixes, and (d) derived causality constraints.

prefixes $\sigma_1, \sigma_2, \ldots, \sigma_m$ of traces in L, convex polyhedra $P_{\widehat{\sigma_1}}, P_{\widehat{\sigma_2}}, \ldots, P_{\widehat{\sigma_k}}$ can be found[2]. Finally, the convex polyhedra

$$P = \bigcup_{i \in \{1 \ldots m\}} P_{\widehat{\sigma_i}}$$

contains all the points defined by the convex polyhedra $P_{\widehat{\sigma_1}}, P_{\widehat{\sigma_2}}, \ldots, P_{\widehat{\sigma_m}}$, thus representing completely the behavior of the log. As Section 2.2 explains, convex polyhedra can be described as the set of solutions of a conjunction of linear inequality constraints. These constraints can be obtained from P in state-of-the-art libraries for convex polyhedra [12]. Hence from P one can obtain the set of m constraints representing it:

$$\begin{aligned} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n &\leq b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n &\leq b_2 \\ \vdots \qquad\qquad\qquad\qquad &\leq \vdots \\ a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \ldots + a_{mn} \cdot x_n &\leq b_m \end{aligned}$$

each one of these constraints models invariants that the system (i.e., the log) satisfy.

Example 1. Figure 4(a) shows part of a log containing several traces on the events a, b, c, d, e, x, y and z[3]. Once the prefixes of the traces are found (Figure 4(b)), corresponding Parikh vectors are converted into polyhedra. A unique polyhedra is derived by performing a join operation on all the polyhedra, and the related invariants are extracted, some of them shown in Figure 4(d).

[2] Here k is in practice significantly smaller than $\sum_{\sigma \in L} |\sigma|$ since many prefixes of different traces in L share the same Parikh vector.

[3] This log contains 100 traces of length 50 each. The reader can inspect the log by following the reference provided in [2].

3.2 From invariants to Petri nets

If we split the coefficients into positive and negative coefficients, constraint i can be represented in the following way:

$$\sum_{a_{ij}>0} a_{ij} \cdot x_j + \sum_{a_{ij}<0} a_{ij} \cdot x_j \leq b_i$$

that can be transformed into:

$$\sum_{a_{ij}>0} a_{ij} \cdot x_j - b_i \leq \sum_{a_{ij}<0} -a_{ij} \cdot x_j$$

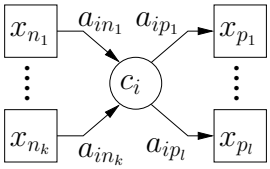
A constraint i is a *causality constraint* if the following conditions hold:

- There is at least one positive coefficient, and
- $b_i \leq 0$

Hence causality constraints can be described as:

$$\sum_{a_{ij}>0} a_{ij} \cdot x_j + c_i \leq \sum_{a_{ij}<0} -a_{ij} \cdot x_j \tag{1}$$

with $c_i = -b_i \geq 0$. The intuition behind causality constraints is that they represent real causalities observed in the log which can be explicit in the derived PN. Hence if we assume indices $n_1, \ldots n_k$ range over the indices of variables with negative coefficients and $p_1, \ldots p_l$ range over the variables with positive coefficients, (1) can be modeled in a PN as:



where c_i inside the place denotes c_i tokens, and a_{ij} in an arc represents the weighted flow relation F for the arc (see Def. 2).

Example 2. Following the example in the previous section (shown in Figure 4), causality constraints can be selected and the corresponding places and arcs introduced, deriving the Petri net shown in Figure 5. For instance the place labeled p is obtained from the constraint $c + d \geq y$.

Finally, a necessary property in the area of Process Mining that relates the set of traces possible in the PN and the ones in the log can be established:

Theorem 1. *Let* $\mathsf{PN} = (P, T, F, M_0)$ *and* L *be a Petri net and a log, respectively, such that* $L(\mathsf{PN}) \supseteq L$*, and the* i*-th causal constraint from* L *as described in (1).*

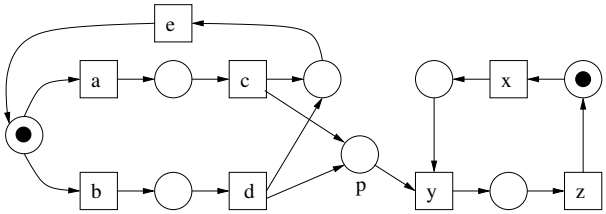


Fig. 5. Petri net derived from the causality constraints shown in Figure 4(d).

Then the $\mathsf{PN}' = (P', T, F', M_0')$ *defined as*

$$P' = P \ \cup \ \{p\}$$
$$F' = F \ \cup \ \{t_j \xrightarrow{a_{ij}} p \mid a_{ij} < 0\} \ \cup \ \{p \xrightarrow{a_{ij}} t_j \mid a_{ij} > 0\}$$
$$M_0'[q] = \begin{cases} M_0[q] & \textit{if } q \neq p \\ c_i & \textit{otherwise} \end{cases}$$

where $p \notin P$*, satisfies* $L(\mathsf{PN}) \supseteq L(\mathsf{PN}') \supseteq L$*.*

Proof. The inclusion $L(\mathsf{PN}) \supseteq L(\mathsf{PN}')$ is well-known in Petri net theory from the fact that $P \subset P'$, $F \subset F'$ and $M_0 \leq M_0'$. The inclusion $L(\mathsf{PN}') \supseteq L$ can be shown by induction on the length of traces in L, and we sketch here the proof. First, if a trace $\sigma = \sigma' t \in L$ satisfies $\sigma' \in L(\mathsf{PN}')$ but $\sigma \notin L(\mathsf{PN}')$, then $t \in p^\bullet$ because transitions not in the postset of the new place inserted p will also be enabled by firing σ' in PN'. Second, the induction can now be used to prove that p will have enough tokens to also enable t, hence contradicting the hypothesis $\sigma \notin L(\mathsf{PN}')$. For $|\sigma| = 1$ it trivially holds. Assume it is true for $|\sigma| \leq n-1$, let us consider $|\sigma| = n$, with $\sigma = \sigma' x t$. If $x \notin {}^\bullet p$ or $t \notin p^\bullet$, applying the induction hypothesis on σ' the statement on p holds. If $x \in {}^\bullet p$ and $t \in p^\bullet$, the induction hypothesis guarantees that after σ', either some other place $q \neq p$ is disabling t or t is enabled. Hence, by firing x the enabling state of t cannot change, contradicting the disabling of t after σ' in PN'. □

The addition of places and arcs corresponding to causality constraints is applied starting from the net $\mathsf{PN}_{init} \stackrel{def}{=} (\emptyset, T, \emptyset, \emptyset)$, which accepts the language T^*. In summary, the flow for Process Mining will follow the steps Log $\xrightarrow{abstract}$ $\xrightarrow{interpretation}$ Convex Polyhedra $\xrightarrow{causality}$ $\xrightarrow{constraints}$ PN. The next corollary follows from Theorem 1 and $L \subseteq L(\mathsf{PN}_{init})$:

Corollary 1. *Let* PN *be the net obtained after adding to* PN_{init} *all the places and arcs corresponding to causality constraints in the convex polyhedra* P *derived from* L*. Then* $L(\mathsf{PN}) \supseteq L$*.*

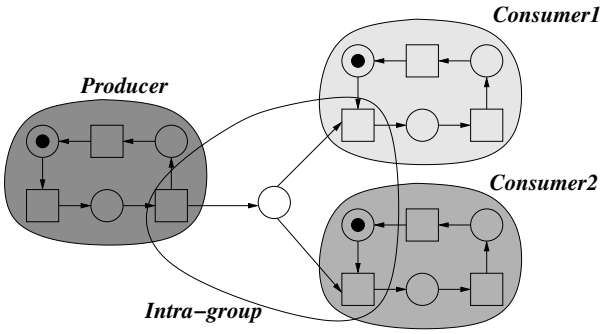


Fig. 6. Detection of groups of related events: *Producer*, *Consumer1* and *Consumer2* are tightly related, whereas the transitions within the *Intra-group* area are loosely related .

3.3 Derivation of unbounded places

Perhaps one of the main theoretical results of this work has been already presented in the example of the previous section. Informally, the derivation of places and arcs from causality constraints may produce *unbounded* places in the Petri net, i.e. places where no bound is possible on their number of tokens. For instance, the place p in Figure 5 may have k tokens when k firings of the sequence ac occur and no firing of y occurs, for any natural number k.

The approaches in the literature for deriving general (unrestricted) Petri nets from logs are [4, 5, 20]. They are based in the *theory of regions* [11], which associates places with regions. These methods cannot compute unbounded regions, and therefore are restricted to model behaviors without these type of places.

4 Process mining of large logs

The approach presented in the previous section cannot be applied for logs extracted from industrial/real-life applications, where either the number of events or the number of Parikh vectors in the traces or both might be too large for growing polyhedra straightaway. For these situations, a divide-and-conquer strategy is required. A possible strategy is presented in this section: instead of a blind search for causality constraints on the whole set of events T, groups of events that are tightly related are identified, and causality constraints are divided into *inter-group* and *intra-group*. For instance, on a log representing a producer and a pair of consumers, inter-group relations might provide the causalities within the three gray zones depicted in Figure 6, whereas intra-group relations might derive the causalities within the corresponding area shown in the figure.

4.1 Identification of groups of tightly coupled events

For determining the partition of T into groups, several techniques can be applied. In this paper, two different techniques are used:

– *Principal Component Analysis (PCA) [13]* is an exploratory data analysis technique that, given a data set of possibly correlated variables, tries to select a subset of variables that is uncorrelated (called principal components) and which accounts for as much of the variability in the data as possible.
– *Firing causalities* is an ad hoc technique to extract causalities between two events from the Parikh vectors considered in the previous section.

In the remainder of this section we explain them in detail:

Principal Component Analysis can be applied to select the partition on $T = \{t_1, \ldots, t_n\}$. The steps are i) the set of Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$ is transformed to the set $\widehat{\sigma'_1}, \ldots, \widehat{\sigma'_m}$ so that $\widehat{\sigma'_i} = (\#(\sigma_i, t_1)/\overline{t_1}, \ldots, \#(\sigma_i, t_n)/\overline{t_n})$, where $\overline{t_i}$ is the mean for number of occurrences of t_i in the set of Parikh vectors of L, ii) compute the *correlation matrix* $A \in [-1 \ldots + 1]^{n \times n}$ using the data set found at i) [13]. This matrix measures the amount of correlation between variables t_i and t_j: when $|A(i,j)| \backsimeq 1$ then both variables are highly correlated. Finally, iii) the number of groups is decided by finding the *eigenvalues* and *eigenvectors* of A: the eigenvalues are sorted according to their value (the highest eigenvalue explains the highest correlation and so on), and only the most important (those that explain the important amount of correlation) are taken. For each selected eigenvalue λ_i, we can select the *leader* of the group for λ_i by looking at the corresponding eigenvector $\alpha_1 \cdot x_1 + \ldots + \alpha_n \cdot x_n$: the leader will be the transition t_i for which absolute value of the coefficient α_i is maximal [13]. The set of transitions t_j such that $|A(i,j)| \backsimeq 1$ will be incorporated to the group leaded by t_i.

Firing causalities between two events t_i and t_j can be extracted by considering the maximal distance (in number of firings) between both events in any possible Parikh vector. Formally, we build the matrix $M \in \mathbb{Z}^{n \times n}$ such that $M(i,j) = \max\{\#(\sigma_k, t_i) - \#(\sigma_k, t_j) \mid 1 \leq k \leq m\}$. There is a causality between t_i and t_j if $M(i,j) > 0$ and $M(j,i) \leq 0$.

4.2 Inter-group causality constraints

The information obtained from the two previous techniques can be combined to form the groups. Intuitively, events t_i and t_j will belong to the same group if

– t_i leads a group and has a high correlation with t_j or vice versa, or
– there is a firing causality relating t_i and t_j

Once a group is identified, the Parikh vectors can be projected into the events of the group and the technique presented in Section 3 can be applied for the projected Parikh vectors.

Example 3. Following with the running example used in the previous section (see the resulting PN in Figure 5), we will show how the same Petri net can be obtained by the hierarchical approach presented in this section. Using the firing

causalities, we will find the pairwise causalities $a \rightarrow c$, $b \rightarrow d$, $x \rightarrow y$ and $y \rightarrow z$. With PCA, more complex relations will be detected: e related with a and b, and also e related with c and d. Hence, two groups are selected: $g_1 = \{a, b, c, d, e\}$ and $g_2 = \{x, y, z\}$. Projecting the Parikh vectors into each group of events will give the causality constraints only relating the events in the group, e.g., for group g_1 the constraints $a \leq c$, $b \leq d$, $e + 1 \leq a + b$ and $c + d \leq e$ will be obtained. These constraints correspond to the subnet to the left of place p in Figure 5. The right subnet correspond to group g_2.

4.3 Intra-group causality constraints

The causalities between different groups might be detected by applying a hierarchical approach: for each group $g_i = \{t^i_1, \ldots, t^i_{|g_i|}\}$, a new variable h_i is created such that represents the sum of firings of the transitions in the group for each Parikh vector. By using the sum of the firings, relations between group's firings might be revealed. Afterwards, the same strategy of Section 3 can be applied to detect causalities between these new variables introduced.

Formally, given a set of groups detected $g_1, \ldots, g_k$ and the set of Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$, a new set of hierarchical Parikh vectors $\widehat{\sigma^h_1}, \ldots, \widehat{\sigma^h_m}$ is created such that

$$\widehat{\sigma^h_m} = (\sum_{t \in g_1} \#(\sigma_1, t), \ldots, \sum_{t \in g_k} \#(\sigma_m, t))$$

and now convex polyhedra can be created representing the hierarchical Parikh vectors:

$$P_{\widehat{\sigma^h_i}} = (h_1 = \sum_{t \in g_1} \#(\sigma_i, t)) \ \cap \ \ldots \ \cap \ (h_k = \sum_{t \in g_k} \#(\sigma_i, t))$$

And in the same way as Section 3.1, a set of invariants can be extracted from the union of the m polyhedra build as explained above.

$$\begin{aligned} a_{11} \cdot h_1 + a_{12} \cdot h_2 + \ldots + a_{1k} \cdot h_k &\leq b_1 \\ a_{21} \cdot h_1 + a_{22} \cdot h_2 + \ldots + a_{2k} \cdot h_k &\leq b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad &\leq \vdots \\ a_{m1} \cdot h_1 + a_{m2} \cdot h_2 + \ldots + a_{mk} \cdot h_k &\leq b_m \end{aligned}$$

These invariants provide relations between groups of variables. Intuitively, invariants where the constant b_i is small denote relevant causalities between groups, whilst invariants with a large constant represent loose causalities possibly originated by the length of the traces in the log. Hence, the invariants are sorted in increasing order according to their constant and only these invariants with small constant are used[4].

[4] Several threshold criteria can be applied to limit the number of invariants to consider. For instance, one can greedily take invariants as far as the constant lies within the order of the previous one.

Algorithm 1: GroupMining

Input: Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$,
Output: Invariant set I containing inter and intra-group causality constraints

```
1  begin
2      I = ∅
3      {g_1, ..., g_k} = ComputeGroups(σ̂_1, ..., σ̂_m)
4      foreach group g_i do
5          I = I ∪ InvariantMining(σ̂_1|_{g_i}, ..., σ̂_m|_{g_i})
6      end
7      H = SelectLowConstant(InvariantMining(σ̂^h_1, ..., σ̂^h_m))
8      foreach invariant i ∈ H do
9          {g_1, ..., g_l} = NonZeroCoefs(i)
10         I = I ∪ InvariantMining(σ̂_1|_{g_1,...,g_l}, ..., σ̂_m|_{g_1,...,g_l})
11     end
12 end
```

When a set of groups are identified to be related, the same technique of Section 4.2 applied for a group can be now applied for the set of groups: the Parikh vectors are projected into the variables that belong to any of the groups related, and causality constraints that relate these variables can be extracted.

The general algorithm is presented as Algorithm 1. The functions used in the algorithm are next defined:

- InvariantMining implements the invariant derivation technique explained in Section 3.1.
- ComputeGroups implements the group derivation technique explained in Section 4.1.
- SelectLowConstant is a function that given a set of invariants, chooses those ones having a small constant.
- NonZeroCoefs is a function that given an invariant, return these variables that have non-zero coefficients, i.e., the variables that define the invariant.

Example 4. Let us show the relation between the two only groups g_1 and g_2 found in Example 3. By creating two sum variables h_1 and h_2 as explained in Section 4.3 and building the polyhedra that corresponds to the union of the polyhedra representing the projection of the Parikh vectors into these variables, the constraint $h_2 \leq h_1$ is detected, meaning that the number of firings in the group g_2 is always less or equal than the number of firings of group g_1. By projecting now the Parikh vectors into these groups and extracting the causality constraints that relate both groups of variables, the constraint $y \leq c + d$ will be extracted, which corresponds to the place p shown in Figure 5. Notice that although for this toy example we ended up by building polyhedra for the whole set of events, in general this will not be the case for real systems. For instance, we experimented with several systems like the one used in our running example, working in parallel. The approach presented in this paper was able to find the intra and inter-group relations for each individual system, thus avoiding to project into the whole set of events. In section 6 we provide such experiments.

Algorithm 2: Sampling

Input: Parikh vectors $\widehat{\sigma_1}, \ldots, \widehat{\sigma_m}$, number of samplings p, sampling size s
Output: Invariant set I

```
1  begin
2      I = ∅
3      for i ← 1 to p do
4          P = empty domain
5          for j ← 1 to s do
6              r =Random(1 ... m)
7              compute P_σ̂r
8              P = P ∪ P_σ̂r
9          end
10         I_1 = Invariants(P)
11         foreach invariant i ∈ I_1 do
12             if i satisfies σ̂_1, ..., σ̂_m then I = I ∪ {i}
13         end
14     end
15 end
```

5 Sampling

Orthogonal to the approach presented in the previous section, this section introduces a technique to avoid dealing with a large number of polyhedra and use instead a limited amount that might be enough for extracting the important relations between the events. For instance, if the log contains ten thousand traces of length a hundred, then in the worst case the techniques presented in the previous sections will be dealing with a million of polyhedra that must be joined, a scenario that often can not be completed successfully with existing libraries for abstract interpretation.

The general algorithm for applying sampling is shown as Algorithm 2. In order to avoid operations with a large number of polyhedra, one can randomly select with uniform probability a small set (s) of Parikh vectors that will be converted to polyhedra and joined (lines 5-9). Once the join operation for the s vectors has been done, the set of invariants *that denote properties for the Parikh vectors considered* must be verified on each one of the Parikh vectors not considered in the join, and only those invariants that are true for all the Parikh vectors will be accepted (lines 10-13). This sampling technique can be applied more than once, i.e., one can apply p samplings in order to find the relations on a set of events (external loop starting at line 3).

Sampling and the strategy presented in the previous section can be applied jointly. This will be accomplished by simply substituting the calls to Invariant-Mining in Algorithm 1 by calls to the function Sampling with a user-defined sampling size and number of samplings. In the experiments, this joint use of these strategies has enabled dealing with large specifications.

	Log Information			genet		Parikh		aim	
Log	$\|T\|$	#traces	#Parikh	P/F	Time	P/F	Time	P/F	Time
a12f0n00_1	12	200	17	11/25	0.1	11/25	1	11/27	0
a12f0n00_5	12	1800	17	11/25	0.1	11/25	0.7	12/30	0
a22f0n00_1	22	100	750	19/49	0.3	19/49	3	19/48	20
a22f0n00_5	22	900	3290	19/49	0.3	19/49	23	16/38	2
a32f0n00_1	32	100	1377	32/75	718	31/73	25	34/84	33
a32f0n00_5	32	900	5543	31/73	1	31/73	112	31/68	6
a42f0n00_1	42	100	1211	memout		44/109	154	41/88	16
a42f0n00_5	42	900	4326	timeout		44/101	1557	49/118	77

Table 1. PN derivation from logs.

6 Experiments

As a proof of concept, the theory of this paper has been implemented in a prototype tool. It is written in `C/C++` and uses the `Apron` library for Convex Polyhedra manipulation [12]. For the PCA method which requires computation of eigenvalues and eigenvectors, the `ALGLIB` library [1] was used. Some conclusions can be drawn from applying the tool on some well-known benchmarks within the Process Mining domain.

The benchmarks applied are logs publicly available within the website [3]. These logs have been used by other algorithms and therefore will be considered in this paper to perform a comparison with two other tools for the same purpose. The tools are: `genet`, which implements algorithms based on the theory of regions and supports the mining of k-bounded PNs [5], and the `Parikh Miner`, that uses the language version of the theory of regions for the same purpose [20]. For using `genet`, an automaton representing all the traces is the input of the tool. Several algorithms exists to transform the log into an automaton [18]. For both tools we used the default parameters.

The comparison is shown in Table 1. For each log, we report the number of events ($|T|$), the number of traces and the number of Parikh vectors obtained after removing repetitions. The number of places discovered (P) and the number of arcs (F) is then provided for each one of the tools, together with the CPU time (measured in a desktop computer) in seconds. For testing each tool, we limited the amount of memory and time that could be used to 1Gb and 10000 seconds respectively. We report the results obtained by the tool of this paper in the columns under `aim`.

For the experiments, we run the tool applying 5 samplings with sampling size a number between 50 and 100, depending on the log. This light sampling application allowed to derive PNs sometimes within two orders of magnitude less CPU time than other methods. Notice that `genet` has both memory (`memout`) and time (`timeout`) problems with the last two logs. On the other hand, `aim` invests considerably more time in deriving a PN for a22f0n00_1, which may be due

	Log Information			genet		Parikh		aim	
Log	$\|T\|$	#traces	#Parikh	P/F	Time	P/F	Time	P/F	Time
ProdCons_1	8	50	3756	7/16	14	0/0	5	8/19	7
ProdCons_3	24	50	4910	timeout		0/0	182	24/57	36

Table 2. PN derivation from two logs obtained from a Producers/Consumers system.

to the particular structure of the polyhedra built on that log. Notice however that the degradation in CPU time is not as significant as in the opposite direction.

A second point to consider is the quality of the information obtained. The PNs derived with `aim` most of the time have the same arcs and places of the other tools. Sometimes extra causalities might be obtained like in a12f0n00_1 or a42f0n00_5. These denote redundant causalities that can be removed by a final application of well-known PN methods for redundant places removal [16].

Table 2 reports experiments with two logs that represent the activity of a system of producers and consumers where components are synchronized through unbounded places (see Figure 6). For ProdCons_1, the PN derived by `aim` is the one shown in Figure 5. The traces for ProdCons_3 contain the interleaving of three independent instances of PNs like the one in Figure 5. Both `genet` and the `Parikh Miner` have problems in dealing with these logs: `genet` cannot derive the unbounded place in ProdCons_1 and received a `timeout` for ProdCons_3, whereas the `Parikh Miner` did not obtain any relation between the activities of the log[5]. In contrast, `aim` was able to discover the exact PN in both logs.

7 Conclusions and future work

A novel theory for deriving a PN from a set of traces has been presented. The results obtained are promising when compared with some of the approaches in the literature for the same task. The current work is mainly focused in obtaining a mature implementation of the first prototype. Also, other strategies to complement the ones described in this paper will be investigated. Finally, the derivation of other graph formalisms will be explored.

Acknowledgements

This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

1. ALGLIB library. http://www.alglib.net.

[5] By changing the default parameters of the `Parikh Miner`, 5 places and 11 arcs are derived for ProdCons_1, but for ProdCons_3 the net is degraded (49 places, 239 arcs), with many places and arcs being redundant.

2. Example log. http://www.lsi.upc.edu/∼jcarmona/prodcons1000.tr.
3. Process mining. www.processmining.org.
4. R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser. Process mining based on regions of languages. In *Proc. 5th Int. Conf. on Business Process Management*, pages 375–383, Sept. 2007.
5. J. Carmona, J. Cortadella, and M. Kishinevsky. New region-based algorithms for deriving bounded Petri nets. *IEEE Transactions on Computers*, 59(3):371–384, 2009.
6. N. Chernikova. Algoritm for discovering the set of all solutions of a linear programming problem. *USSR Computational Mathematics and Mathematical Physics*, 6(8):282–293, 1964.
7. R. Clarisó, E. Rodríguez-Carbonell, and J. Cortadella. Derivation of non-structural invariants of petri nets using abstract interpretation. In G. Ciardo and P. Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 188–207. Springer, 2005.
8. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. of the 2nd Int. Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 84–97. ACM Press, New York, 1978.
11. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part I, II. *Acta Informatica*, 27:315–368, 1990.
12. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 661–667. Springer, 2009.
13. I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.
14. A. Miné. The octagon abstract domain. In *Analysis, Slicing and Tranformation (in Working Conference on Reverse Engineering)*, IEEE, pages 310–319. IEEE CS Press, Oct. 2001.
15. T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.
16. M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place or transition net systems. In *Petri Nets*, volume 1491 of *LNCS*, pages 309–373. Springer Verlag, 1996.
17. W. M. P. van der Aalst and C. W. Günther. Finding structure in unstructured processes: The case for process mining. In T. Basten, G. Juhás, and S. K. Shukla, editors, *ACSD*, pages 3–12. IEEE Computer Society, 2007.
18. W. M. P. van der Aalst, V. Rubin, H. M. W. E. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 2009.
19. W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.
20. J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In K. M. van Hee and R. Valk, editors, *Petri Nets*, volume 5062 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2008.