# An Iterative Algorithm for Applying the Theory of Regions in Process Mining

B.F. van Dongen[1], N. Busi[2], G.M. Pinna[3], and W.M.P. van der Aalst[1]

[1] Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna
busi@cs.unibo.it
[3] Dipartimento di Matematica e Informatica, Università di Cagliari
gmpinna@unica.it

**Abstract.** The research domain of process mining, or more specifically process discovery, aims at constructing a process model as an abstract representation of an event log. The goal is to build a model (i.e. in terms of a Petri net) that (1) can reproduce the log under consideration, and (2) does not allow for much more behaviour than shown in the log.

The Theory of Regions can be used to transform a state-based model (such as a transition system) into a Petri net that exactly mimics the behaviour of the transition system.

In this paper, we use the Theory of Regions to do process discovery, and we address two problems. First, we show how event logs that do not carry state information can be transformed into transition systems. Second, we deal with the problem of large logs, by showing that the proposed algorithm can be made *iterative* over the traces in a log, i.e. we change the complexity of the algorithm, such that it requires significantly less space, but more time.

## 1 Introduction

At the basis of process aware information systems, typically lie process models of some sort, e.g. either conceptual models or executable models. The enactment of processes by the information system, i.e. the operational process, is based on these process models, and all steps performed during enactment are typically logged in some sort of event log.

### 1.1 Event Logs

Figure 1 shows the relations between the operational process, the models that describe it and the logs generated from it. Furthermore, it shows how the research areas of process mining relates to these entities, by showing how event logs, process models and some desired or undesired properties can be used for *log-based verification*, *process verification*, *process discovery* and *conformance testing*.

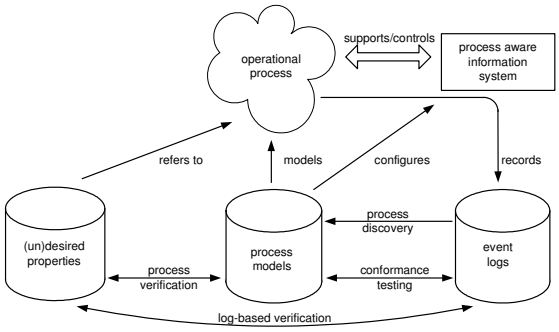


**Fig. 1.** Process Mining.

A complete overview of recent process mining research is beyond the scope of this paper. Therefore, we limit ourselves to a brief introduction to process discovery only and refer to [3, 4] and the http://www.processmining.org web page for a more complete overview of the whole research domain.

### 1.2 Process Discovery

One of the main challenges in the context of process mining is *process discovery*, i.e. how to generate a model describing a process while only looking at event logs.

Event logs such as the one shown in Table 1 are used as the starting point for process mining, and from a *process perspective* we focus on the control-flow, i.e., the ordering of activities, which is shown in terms of a Petri net (cf. [27]) in Figure 2(a). The goal of process mining from this perspective is to find a good characterization of all possible paths, e.g., expressed in terms of a Petri net or Event-driven Process Chain (EPC) [21, 22]. The *organizational perspective* focuses on the originator field, i.e., which performers are involved and how are

| case id | activity id | originator | case id | activity id | originator |
|---|---|---|---|---|---|
| case 1 | activity A | John | case 5 | activity A | Sue |
| case 2 | activity A | John | case 4 | activity C | Carol |
| case 3 | activity A | Sue | case 1 | activity D | Pete |
| case 3 | activity B | Carol | case 3 | activity C | Sue |
| case 1 | activity B | Mike | case 3 | activity D | Pete |
| case 1 | activity C | John | case 4 | activity B | Sue |
| case 2 | activity C | Mike | case 5 | activity E | Clare |
| case 4 | activity A | Sue | case 5 | activity D | Clare |
| case 2 | activity B | John | case 4 | activity D | Pete |
| case 2 | activity D | Pete | | | |

**Table 1.** An event log (audit trail).

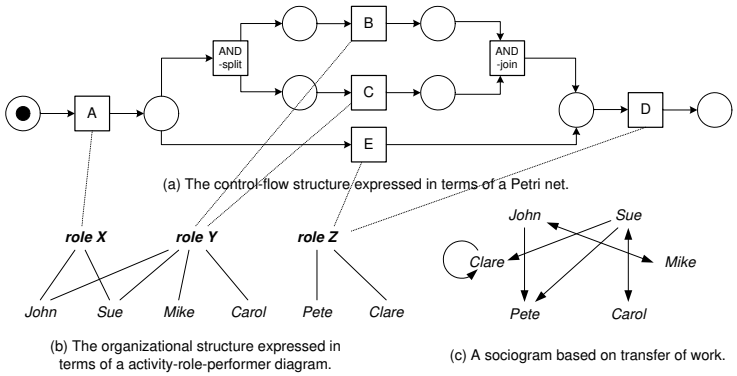


**Fig. 2.** Some mining results for the process perspective (a) and organizational (b and c) perspective based on the event log shown in Table 1.

they related. The goal is to either structure the organization by classifying people in terms of roles and organizational units (Figure 2(b)) or to show relation between individual performers (i.e., build a social network as described in [2] and references there, and as shown in Figure 2(c)). The *case perspective* focuses on properties of cases. Cases can be characterized by their path in the process or by the originators working on a case. However, cases can also be characterized by the values of the corresponding data elements. For example, if a case represents a replenishment order, it is interesting to know the supplier or the number of products ordered.

Ever since the first work on process mining emerged, researchers in the area of Petri nets were wondering how this relates to the so-called Theory of Regions. In this paper, we focus on the process perspective, i.e. we discover the control flow of a process from its event logs, using concepts from the Theory of Regions.

### 1.3 Theory of Regions

The Theory of Regions ([18, 8]) establishes a connection between transitions systems and Petri nets through so called *net synthesis*. The idea behind the Theory of Regions is that a state-based model i.e. a model describing which states a process can be in and which transitions are possible between these states, can be transformed into a Petri net, i.e. a compact representation of the state space, explicitly showing causality, concurrency and conflicts between transitions.

It is clear that the Theory of Regions shares common goals with the research area of process mining. However, there are some subtle differences:

– First of all, the starting point for net synthesis is a so-called transition system, i.e. a description of a process explicitly showing *all possible states*, whereas event logs *do not carry state information*.

– Second, the Theory of Regions assumes the transition system to show *all possible transitions* between states, while in process mining, the assumption usually is that the logs are *not exhaustive*, i.e. they do not contain all possible sequences of events.

In Section 4, we introduce a region-based algorithm that deals with these two issues. Then, in Section 5, we show how that algorithm can be applied in an iterative way, thus reducing the space requirement of the algorithm. Before we conclude this paper with Section 6 showing the implementation of our work and the conclusions in Section 7, we first discuss some related work in Section 2 and introduce some notation in Section 3.

## 2 Related Work

### 2.1 Regions

Regions and the related theory has been developed starting from the seminal papers of Ehrenfeucht and Rozenberg ([18]) and has been successfully applied to the so called net synthesis (see, among others, [7, 16, 11]) and to the characterization of concurrency models (see, among others, [26, 20, 24]). To the best of our knowledge, the approach to process mining based on regions has not yet received great attention. In fact the use of regions in general, gives a saturated net (i.e. with many more places) and the complexity is quite high in comparison with other methods. The novelty of our approach lies in the incremental calculus of regions. Although regions of a transition systems can be combined algebraically under precise conditions ([9, 8]) the attempt to find regions of a compound transition system from the regions of the components is new. We believe that this can give better performance.

It is worth recalling here that regions have been used in many different settings, e.g. in the synthesis and verification of asynchronous circuits (e.g. [14]) or in the verification of security properties (e.g. [10]).

### 2.2 Process mining

Since the mid-nineties several groups have been working on techniques for process mining, i.e., discovering process models based on observed events. In [3, 4] an extensive overview is given of the work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [6]. Cook et al. investigated similar issues in the context of software engineering processes [12]. Herbst [19] was one of the first to tackle more complicated processes, e.g., processes containing duplicate tasks. Most of the approaches have problems dealing with concurrency. The $\alpha$ algorithm [5] is an example of a simple technique that takes concurrency as a starting point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature). Approaches based on heuristics or genetic algorithms can deal with noise [29].

The application of the Theory of Regions in the context of process mining has been addressed in [1], where the authors address process mining in the context of software engineering. One of the challenges faced in this context is to find state information in event logs. In [1], the authors propose several ways of doing so. Furthermore, their approach is implemented in ProM by making a link between the event logs of ProM and a well-known tool tailored towards the application of the Theory of Regions, called Petrify [13].

The results presented in this paper are fully implemented in the open source framework ProM. (See www.processmining.org for the latest version.) ProM serves as a testbed for our process mining research. Most of the leading process mining approaches have been implemented in ProM.

## 3 Preliminaries

In this section, we introduce the notations and concepts we use in the remainder of this paper.

### 3.1 Petri nets

Petri nets are a formalism that can be used to specify processes. Since Petri nets have a formal and executable semantics, processes modelled in terms of a Petri net can be executed by an information system. For an elaborate introduction to Petri nets, we refer to [15, 25, 27]. For sake of completeness, we mention that the Petri nets we use in this paper correspond to a classic subclass of Petri nets, namely Place/Transition nets.

A Petri net consists of two modeling elements, namely places and transitions. When a Petri net is represented visually, we draw transitions as boxes and places as circles. Furthermore, to denote the state of a process execution the concept of *tokens* is used. A token is placed inside a place to show that a certain condition holds. Each place can contain arbitrarily many of tokens. If a transition *fires*, one token is removed from each of the input places and one token is produced for each of the output places. The distribution of tokens over the places is called a *marking*.

Figure 3 shows an example of a marked P/T-net, containing 11 transitions, i.e.
$T = \{A, B, C, D, E, F, G, H, I, J, K\}$ and 10 places, of which we typically do not show the labels. Furthermore, three places are marked, i.e. they contain a token denoted by the black dot.

Formally, a Place/Transition net with some initial marking is defined as follows.

**Definition 3.1. (Place/Transition net)** $\wp = (P, T, F)$ is a place/transition net (or P/T-net) if:

- $P$ is a finite, non empty set of places,
- $T$ is a finite, non empty set of transitions, such that $P \cap T = \emptyset$ and $T \neq \emptyset$,

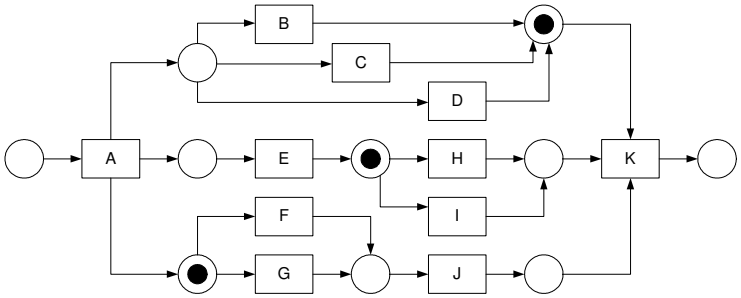


**Fig. 3.** Example of a marked Petri net with 11 transitions and 10 places.

– $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation of the net,

A marking is a *bag* over the set of places $P$ and a *marked* P/T-net is a pair $(\wp, M_0)$, where $\wp = (P, T, F)$ is a P/T-net and where $M_0$ is a bag over $P$ denoting the *marking* of the net. The set of all marked P/T-nets is denoted $\mathcal{N}$.

Note that any place/transition net $\wp = (P, T, F)$ defines a directed graph $((P \cup T), F)$. In this paper, we restrict ourselves to P/T-nets where for all transitions $t$ holds that $\bullet t \neq \emptyset$ and $t\bullet \neq \emptyset$ and for all places $p$ holds that $\bullet p \cup p\bullet \neq \emptyset$.

As we stated before, Petri nets are used to describe processes and therefore, to describe dynamic behaviour. So-far, we have only defined the static structure of a Petri net. Therefore, we now define the dynamics. The dynamics of a Petri net are defined using the concepts of a marking and a firing rule. However, first we state when a transition is *enabled*.

**Definition 3.2. (Enabled transition)** Let $\wp = ((P, T, F), M_0)$ be a marked P/T-net. Transition $t \in T$ is *enabled*, denoted $(\wp, M_0)[t\rangle$, if and only if $\bullet t \leq M_0$.

In other words, a transition is enabled if each of its input places contains at least one token. In Figure 3 for example, the transitions $F$,$G$,$H$ and $I$ are all enabled. If a transition is enabled, it can fire. Transitions fire one by one using the following firing rule.

**Definition 3.3. (Firing rule)** Let $\wp = ((P, T, F), M_0)$ be a marked P/T-net. The *firing rule* $\_\,[\_\rangle\,\_ \subseteq \mathcal{N} \times T \times \mathcal{N}$ is the smallest relation satisfying for any $((P, T, F), M_0) \in \mathcal{N}$ and any $t \in T$, $(\wp, M_0)[t\rangle \Rightarrow (\wp, M_0)\,[t\rangle\,(\wp, M_0 - \bullet t + t\bullet)$.

The firing rule says that if a transition is enabled then it can fire and when it does, it removes exactly one token from each of its input places and adds exactly one token to each of its output places. If in Figure 3 transition $G$ would fire, then the input place of $G$ contains no tokens after the firing and the output place of $G$ contains one token after the firing.

The distribution of tokens over places is what we call a marking. Since the firing rule defines how one marking can be transformed into another marking, we can define a set of *reachable* markings.

**Definition 3.4. (Reachable markings)** Let $(\wp, M_0)$ be a marked P/T-net in $\mathcal{N}$. A marking $M$ is *reachable* from the initial marking $M_0$ if and only if there exists a sequence of enabled transitions whose firing leads from $M_0$ to $M$. The set of reachable markings of $(\wp, M_0)$ is denoted $[\wp, M_0\rangle$.

If we look at a Petri net with an initial marking, then each marking that is reachable, can be reached by executing some transitions in a given order. Such a sequence of transition firings is what we call a *firing sequence*

**Definition 3.5. (Firing sequence)** Let $\wp = ((P, T, F), M_0)$, be a marked P/T-net. A sequence $\sigma \in T^*$ is called a *firing sequence* of $(\wp, M_0)$, if and only if, for some natural number $n \in \mathbb{N}$, there exist markings $M_1, \ldots, M_n$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = t_1 \ldots t_n$ and, for all $i$ with $0 \leq i < n$, $(\wp, M_i)[t_{i+1}\rangle$ and $M_{i+1} = M_i - \bullet t_{i+1} + t_{i+1}\bullet$. (Note that $n = 0$ implies that $\sigma = \langle\rangle$ and that $\langle\rangle$ is a firing sequence of $(\wp, M_0)$.) Sequence $\sigma$ is said to be *enabled* in marking $M_0$, denoted $(\wp, M_0)[\sigma\rangle$. Firing the sequence $\sigma$ results in a marking $M_n$, denoted $(\wp, M_0)\,[\sigma\rangle\,(\wp, M_n)$. Furthermore, for all $i \in \{0, \ldots, n-1\}$ we use $\sigma_i = t_{i+1}$ and, we say $t \in \sigma$ if there exists an $0 \leq i < |\sigma|$ with $\sigma_i = t$.

The goal of process mining is to obtain a Petri net that can reproduce the event log under consideration, i.e. each trace in the log is a firing sequence of the resulting Petri net.

### 3.2 Process Logs

Information systems typically log all kinds of events. Unfortunately, most systems use a specific format. Therefore, we formalize the concept of process logs. The basic assumption is that the log contains information about specific *activities* executed for specific *cases* (i.e., traces).

**Definition 3.6. (Trace, Process log)** Let $T$ be a set of activities. $\sigma \in T^*$ is a *trace*, and $W \in \mathcal{P}(T^*)$ is a *process log*.[1]

In Definition 3.6, we define a log as a *set* of traces. Note that in real life, logs are *bags* of traces, i.e. the same trace may occur more than once, as shown in our example. However, in this paper, we do not have to consider occurrence frequencies of traces and therefore sets suffice for our properties and proofs.

In process mining, process logs are said to be *globally complete* if a log contains all the possible behaviour of the underlying system (i.e. it shows complete behaviour).

**Definition 3.7. (Globally complete log)** Let $T$ be a set of log events and $L \in \mathcal{P}(T^*)$ be the set of all possible traces of some model or process. Furthermore, let $W \in \mathcal{P}(T^*)$ be a process log over $T$. We say that $W$ is a globally complete log if and only if $W = L$.

[1] With $\mathcal{P}(T) = \{T' \subseteq T\}$ we denote the powerset of a set $T$, i.e. the set of all subsets of $T$ and with $T^*$ we denote the set of all sequences composed of zero or more elements of $T$

The problem with the definition of globally complete logs is that it is hard to tell whether a log is globally complete or not, if a process model of the underlying process is not available. Since the goal of process discovery is to obtain such a model from the log, we should not make the assumption that it is available beforehand. Therefore, in this paper, we say that the goal of process discovery is to find a Petri net, of which each trace in the log is a firing sequence.

### 3.3 Transition Systems

State-based models are widely used for the formal specification and verification of systems. Such models are usually called *transition systems*, i.e. models that explicitly show the states a process can be in and all possible transitions between those states.

**Definition 3.8. (Transition system)** A labelled state transition system is a triple $(S, \Lambda, \rightarrow)$, where $S$ is a set of states, $\Lambda$ is a set of labels, and $\rightarrow \subseteq S \times \Lambda \times S$ is a ternary relation. If $p, q \in S$ and $\alpha \in \Lambda$, $(p, \alpha, q) \in \rightarrow$ is usually written as $p \xrightarrow{\alpha} q$. This represents the fact that there is a transition from state $p$ to state $q$, by executing a transition (or by performing an activity) labelled $\alpha$. Furthermore, in this paper, we assume that a transition system is connected.

In this paper, we restrict ourselves to transition systems with a single initial state, i.e. one state without incoming transitions.

For the purpose of deriving Petri nets from transition systems, the concept of regions was first introduced in [18], where these regions served as intermediate objects, between a transition system on the one hand and a Petri net on the other hand. This process, to go from a state-based model to a Petri net, is called synthesis and the goal is to generate a Petri net that *exactly* mimics the behaviour of the state-based model.

**Definition 3.9. (Region in a transition system)** Let $TS = (S, \Lambda, \rightarrow)$ be a transition system. We say that $R \subseteq S$ is a region of $TS$ if and only if for all $(p, \alpha, q), (p', \alpha, q') \in \rightarrow$ holds that:

- if $p \in R$ and $q \notin R$ then $p' \in R$ and $q' \notin R$, i.e. all transitions labelled $\alpha$ exit the region, and we say that $R$ is a *pre-region* of $\alpha$,
- if $p \notin R$ and $q \in R$ then $p' \notin R$ and $q' \in R$, i.e. all transitions labelled $\alpha$ enter the region, and we say that $R$ is a *post-region* of $\alpha$,
- if $(p \in R) = (q \in R)$ then $(p' \in R) = (q' \in R)$, i.e. all transitions labelled $\alpha$ *do not cross* the region.

It is easy to see that there are two *trivial* regions, i.e. $\emptyset \subseteq S$ and $S \subseteq S$ are regions. The collection of all regions of a transition system $TS$ is called $\Re(TS)$. A region $R \in \Re(TS)$ is said to be *minimal* if and only if for all $R' \subset R$ with $R' \neq \emptyset$ holds that $R' \notin \Re(TS)$. The set of all minimal regions is denoted by $\Re^{min}(TS)$. Furthermore, it is important to note that regions do not depend on one label $\alpha$, i.e. they always depend on the entire set of labels in the transition system.

When reasoning about regions and events, we use a generic notation for retrieving pre-regions and post-regions of events and entering and exiting activities of regions.

**Definition 3.10. (Pre-set and post-set for events and regions)** Let $TS = (S, \Lambda, \rightarrow)$ be a transition system and $a \in \Lambda$ an activity label. The pre-region set and the post-region set of $a$ are the sets of regions defined as follows:

- ${}^{TS}\circ a = \{R \in \Re(TS) \mid \forall (s, a, s') \in \rightarrow : s \in R \wedge s' \notin R\}$ and
- $a {}^{TS}\circ = \{R \in \Re(TS) \mid \forall (s, a, s') \in \rightarrow : s \notin R \wedge s' \in R\}$

Given a region $R \in \Re(TS)$, ${}^{TS}\circ R = \{a \in \Lambda \mid R \in a {}^{TS}\circ\}$ and $R {}^{TS}\circ = \{a \in \Lambda \mid R \in {}^{TS}\circ a\}$. Note that if the context is clear, we omit the superscript $TS$, i.e. $\circ R$ and $R\circ$.

As we explained before, in process discovery, we do not have a model to start with. However, the Theory of Regions is still highly relevant. Assume that we have a process log and we want to obtain a Petri net that exactly describes what we have seen in the log, i.e. not only do we require that each trace in the log is a firing sequence of the Petri net, but also that all possible firing sequences in the Petri net are the traces in the log.

Obviously, if our log would be a state-based model, the Theory of Regions would apply directly. However, there are two issues with our logs:

- Our process instances are sequences of events and do not carry any state information, so there is no relation between different process instances,
- We will never know if our process log is large enough to exhibit all possible behaviour of the underlying process.

## 4 Region-based Process Discovery

The first assumption about the log that we need to make in order to be able to use the Theory of Regions is about its completeness. Since the Petri net resulting from the synthesis will exactly mimic the behaviour shown in the log, we assume that the log shows *all possible* behaviour, i.e. we *assume* it is *globally complete* as defined in Definition 3.7.

### 4.1 From Process Logs to Transition Systems

To apply the Theory of Regions, we need a transition system, i.e. we need a way to transform process logs into transition systems. For this, we need to identify states, which are not contained in the log. To solve this problem, we take a naive approach, i.e. we assume that there is only one known state, i.e. the initial state. Furthermore, we assume that there is a unique activity that is the first activity in all sequences of events in the log.

The assumption that the initial state is the same for all traces is an intuitive one, whereas the assumption that all traces start with the same transition is not

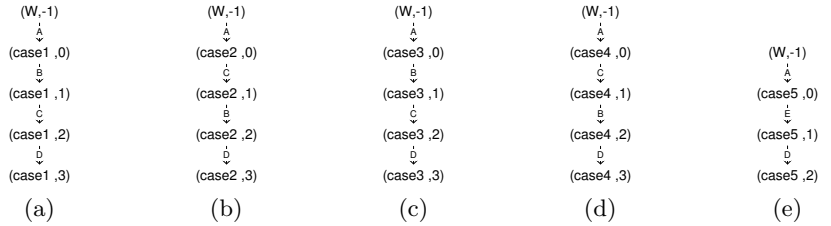


**Fig. 4.** The 5 cases of Table 1 as transition systems.

so trivial. In practice, traces can easily start with many different alternatives (for example different activities for first-time customers or recurrent customers). However, this can easily be solved by adding an artificial activity to the start of each trace and therefore the assumption is not restrictive.

The first step in region based process mining is to convert each trace of the process log into a transition system. This translation is rather trivial, and Figure 4 gives the 5 translations for the 5 cases of Table 1

**Definition 4.1. (Trace to transition System)** Let $T$ be a set of log events and let $W$ be a globally complete process log over $T$, i.e., $W \in \mathcal{P}(T^*)$. Furthermore, let $\sigma \in W$ be an arbitrary trace. We define $TS(\sigma) = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)$ to be a transition system, such that:

- $S_\sigma = \{(\sigma, i) \in \{\sigma\} \times \mathbb{N} \mid 0 \leq i < |\sigma|\} \cup \{(W, -1)\}$, i.e. the set of states consists of all indices in all process instances, as well as a global state $(W, -1)$, which is the initial state,
- $\Lambda_\sigma = \{t \mid t \in \sigma\}$, i.e. the set of labels is the set of log events,
- $\rightarrow_\sigma = \{((\sigma, i), \sigma_{i+1}, (\sigma, i+1)) \in S \times T \times S)\}$, i.e. the trace is represented as a sequence of state transitions, starting in the initial state. The transitions between each two states are defined as the activity at the given position in the trace.

Using the translation from a single trace to a transition system, we can translate an entire log to a transition system. Again this is a straightforward translation and the result of our example log is shown in Figure 5.

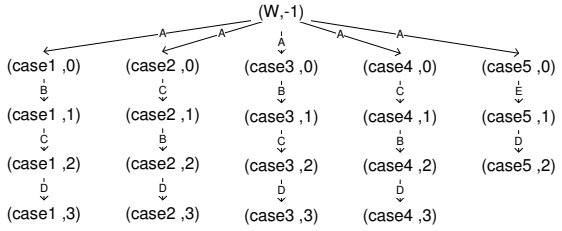


**Fig. 5.** Combining the 5 transition systems of Figure 4 into one transition system.

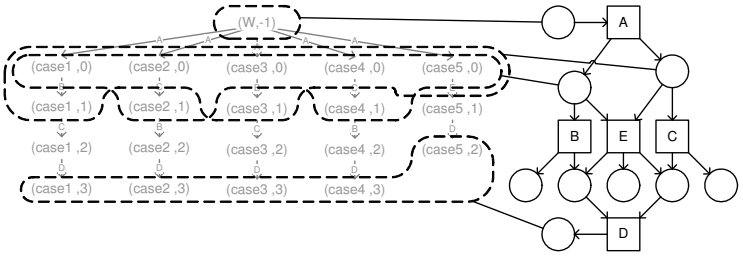


**Fig. 6.** Some minimal regions of the transition system in Figure 5 and the generated Petri net.

**Definition 4.2. (Process log to transition system)** Let $T$ be a set of log events and let $W$ be a globally complete process log over $T$, i.e., $W \in \mathcal{P}(T^*)$. We define $TS(W) = (S, \Lambda, \rightarrow)$ to be a transition system, such that:

- $S = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} S_\sigma$, i.e. the union over the states of the transition system translations of each individual trace,
- $\Lambda = T$, i.e. the set of labels is the set of log events,
- $\rightarrow = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} \rightarrow_\sigma$, i.e. each trace is represented as a sequence of state transitions, starting from the common initial state. The transition between each two states is made by the activity at the given position in the trace.

It is important to note that this algorithm presented in Definition 4.3 has been proven to work on so-called *elementary transition systems* only, i.e. on transition systems where each two *different* states have to belong to two *different* sets of regions, and if a state $s$ is included in all pre-regions of an event $e$, then that event $e$ must be enabled in $s$. Our translation from process logs to transition systems does not enforce this, i.e. the resulting transition system is not necessarily an elementary transition system. However, for now, we assume that this is the case and in Section 6, we shown an example where the transition system is not elementary, but the approach still works.

Once a process log is converted into the transition system, we can use the Theory of Regions to generate a Petri net from it. The idea is that each log event from the log is represented as a transition in the transition system and therefore, we know that each log event has a set of pre-regions and post-regions in the transition system, which may represent the input and output places in the Petri net. Note that many algorithms in the area of Petri net synthesis have been developed, to come up with smaller Petri nets in terms of the number of places, or with free-choice Petri nets by introducing multiple transitions with the same name [8]. However, in this paper, we present the most basic algorithm, presented in [16] and [18], where minimal regions are translated into places. In Figure 6, we show the result of the synthesis on our example of Figure 5.

**Definition 4.3. (Region-based mining algorithm)** Let $T$ be a set of log events, $W$ a process log over $T$ with $t_i \in T$ an initial event in $W$ and $TS(W) =$

$(S, \Lambda, \rightarrow)$ to be the transition system generated from that using Definition 4.2. We define a marked Petri net $\wp = ((P, \Lambda, F), M_0)$, synthesized from $TS(W)$, as follows:

- $P = \Re^{min}(TS(W))$, i.e. each place corresponds to a minimal region in the transition system,
- $F = \{(R, t) \in P \times T \mid R \in \circ t\} \cup \{(t, R) \in T \times P \mid R \in t\circ\}$, i.e. each transition is connected to an input place if the corresponding region is a pre-region and to an output place if the corresponding region is a post-region,
- $M_0 = \overset{\circ}{\bullet} t_i$, i.e. the initial transition has all its input places marked, with one token.

The definition of Petri net synthesis results in a Petri net whose state space is bisimilar to the original transition system. Without going into details about bisimilarity, it is enough to realize that this implies that the state space is trace equivalent with the original transition system and hence the Petri net can generate exactly those traces we observed in the log.

It is important to note that this algorithm presented in Definition 4.3 has been proven to work on so-called *elementary transition systems* only, i.e. on transition systems where each two *different* states have to belong to two *different* sets of regions, and if a state $s$ is included in all pre-regions of an event $e$, then that event $e$ must be enabled in $s$. Our translation from process logs to transition systems does not enforce this, i.e. the resulting transition system is not necessarily an elementary transition system. However, for now, we assume that this is the case and at the end of Subsection **??**, we show an example where the transition system is not elementary and the approach still works.

Although the algorithm presented in Definition 4.3 results in a large number of places, there are many ways to reduce that number, for example by just looking at the minimal regions, or by removing redundant places (or regions). Another way of reducing the number of places is by making a better transition system out of the log. However, that would require us to have state information in the log, or to estimate such information, for example by saying that the same sequences of events in different traces lead to the same global state. Under the assumption of a globally complete log, such estimations will never introduce new traces that we did not observe in the log, however it does reduce the transition system and therefore the possible number of regions.

### 4.2 Mining Quality

It is straightforward to see that the approach presented here indeed leads to a Petri net and due to the fact that the state space of that net is bisimilar to the original transition system, the state space is also trace equivalent with the transition system. Furthermore, since the transition system contains exactly those traces that are represented in the event log, the Petri net can reproduce the event log exactly.

At first sight, it seems that the Theory of Regions provides the answer to process discovery, i.e. the resulting Petri net can reproduce the event log and that was our goal. However, there are some down sides to this result.

First of all, we still do not know whether our log is globally complete, i.e. we are not sure if the resulting Petri net should allow for *more* behaviour than it currently does. Taking a larger log could help in solving this issue, but the answer to whether or not a log is globally complete can never be answered.

An even greater limitation is that the algorithm requires the transition system to be built before the calculation of regions. Especially for large, complex logs, this is not feasible since the resulting transition system would be too big for memory, i.e. there is a *space limitation*. Therefore, we take advantage of the structure of our transition system to introduce an iterative approach.

## 5 Iterative Region Calculation

Our naive approach towards using the Theory of Regions in the context of process discovery has the problem that it requires the entire transition system to be built in memory. Since process logs can easily contain thousands of cases, referring to hundreds of events each, the resulting transition system may be too large to be stored in computer memory. Therefore, in this section, we use the structure of our transition system to introduce an iterative approach.

Recall that the transition system we built from the process logs in Definition 4.2 is basically a straightforward sum over a set of sequential transition systems with a known initial state. In this section, we show that if we have two transition systems with equal initial states, we can calculate the regions of the combination of these two transition systems without constructing the transition system itself. Finally, after iterating over all traces individually, we translate the resulting set of regions to a Petri net.

In Definition 3.9, we defined a region as a set of states, such that each transition in the transition system either *enters*, *exits* or *does not cross* the region. Since our aim is to obtain the regions of an unknown transition system by combining the regions of two smaller transition systems with similar initial states, we introduce the concept of compatible transition systems and compatible regions.

**Definition 5.1. (Compatible transition systems)** Let $TS_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $TS_2 = (S_2, \Lambda_2, \rightarrow_2)$ be two transition systems. We say that $TS_1$ and $TS_2$ are compatible if and only if:

- $|S_1 \cap S_2| = 1$, i.e. there is only one common state and,
- For $s \in S_1 \cap S_2$ holds that there is no $p \in S_1 \cup S_2$ and $\alpha \in \Lambda_1 \cup \Lambda_2$ with $(p, \alpha, s) \in \rightarrow_1 \cup \rightarrow_2$, i.e. the common state is an initial state.

Two transition systems are compatible if they share a common initial state, but no other states. It is easily seen that the translation of two traces from one process log to two transition systems yields two compatible transition systems.

For compatible transition systems, we define compatible regions.

**Definition 5.2. (Compatible regions)** Let $TS_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $TS_2 = (S_2, \Lambda_2, \rightarrow_2)$ be two compatible transition systems. Let $R_1 \in \Re(TS_1)$ and $R_2 \in \Re(TS_2)$. We say that $R_1$ is *compatible* with $R_2$, denoted by $R_1 \leftrightarrow R_2$ if and only if

- $({}^{TS_1}_{\circ} R_1 \setminus {}^{TS_2}_{\circ} R_2) \cap \Lambda_2 = \emptyset$, and,
- $({}^{TS_2}_{\circ} R_2 \setminus {}^{TS_1}_{\circ} R_1) \cap \Lambda_1 = \emptyset$, and,
- $(R_1 {}^{TS_1}_{\circ} \setminus R_2 {}^{TS_2}_{\circ}) \cap \Lambda_2 = \emptyset$, and,
- $(R_2 {}^{TS_2}_{\circ} \setminus R_1 {}^{TS_1}_{\circ}) \cap \Lambda_1 = \emptyset$, and,
- $\forall_{s \in S_1 \cap S_2}\ s \in R_1$ if and only if $s \in R_2$

A region of one transition system is compatible with a region of another transition system if all transitions that enter the first region also enter the second region, or do not appear at all in the second transition system. Similarly, this has to hold for all exiting transitions. Furthermore, if a common state appears in one region, it should appear in the other region as well.

The first step towards our iterative approach is to define how to add two compatible transition systems, where we use the earlier translation of a process log to a transition system, i.e. Definition 4.2.

**Definition 5.3. (Adding compatible transition systems)** Let $TS_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $TS_2 = (S_2, \Lambda_2, \rightarrow_2)$ be two compatible transition systems. We define $TS = (S, \Lambda, \rightarrow)$ as the sum of the two transition system, denoted by $TS = TS_1 \oplus TS_2$, such that:

- $S = S_1 \cup S_2$, i.e. the union over the states of both transition systems,
- $\Lambda = \Lambda_1 \cup \Lambda_2$, i.e. the union over the labels of both transition systems,
- $\rightarrow = \rightarrow_1 \cup \rightarrow_2$, i.e. the union over the transitions of both transition systems.

**Property 5.4. (Adding yields a compatible transition system)** Let $TS_1 = (S_1, \Lambda_1, \rightarrow_1)$, $TS_2 = (S_2, \Lambda_2, \rightarrow_2)$ and $TS_3 = (S_3, \Lambda_3, \rightarrow_3)$ be three compatible transition systems and let $TS = (S, \Lambda, \rightarrow) = TS_1 \oplus TS_2$ be the sum over the first two. $TS$ is compatible with $TS_3$.

**Proof.** For $TS$ and $TS_3$ to be compatible, we need to show that there is one common initial state. Let $s_i \in S_1 \cap S_2$ be the common initial state of $TS_1$ and $TS_2$. Since there is only one initial state, we know that this is the initial state of both $TS_3$ and of $TS$, hence $TS$ and $TS_3$ share one initial state and hence $TS$ and $TS_3$ are compatible. □

It remains to be shown that we are able to calculate the set of regions of the sum of two transition systems from the sets of regions of the transition systems we are adding.

**Property 5.5. (Region summation is possible)** Let $TS_1 = (S_1, \Lambda_1, \rightarrow_1)$ and $TS_2 = (S_2, \Lambda_2, \rightarrow_2)$ be two compatible transition systems. Furthermore, let $TS = (S, \Lambda, \rightarrow) = TS_1 \oplus TS_2$ be the sum over both transition systems. We show that $\Re(TS) = D$, where $D = \{R \in \mathcal{P}(S) \mid \exists_{R_1 \in \Re(TS_1)} \exists_{R_2 \in \Re(TS_2)} R_2 \leftrightarrow R_1 \wedge R = R_1 \cup R_2\}$

**Proof.** Assume that $R = R_1 \cup R_2$ with $R_1 = R \cap S_1$ and $R_2 = R \cap S_2$. It is easy to see that $R_1 \in \Re(TS_1)$ and $R_2 \in \Re(TS_2)$. The question remains whether $R_1 \leftrightarrow R_2$, however since $R \in \Re(TS)$, we know that all events entering $R$ also enter $R_1$, or do not appear in $\Lambda_1$, i.e. for all $e \in {}^{TS}\circ R$ holds that either $e \in {}^{TS_1}\circ R_1$ or $e \notin \Lambda_2$, hence $({}^{TS}\circ R \setminus {}^{TS_1}\circ R_1) \cap \Lambda_2 = \emptyset$. Since ${}^{TS_1}\circ R_1 \subseteq {}^{TS}\circ R$, we know that $({}^{TS_1}\circ R_1 \setminus {}^{TS_1}\circ R_1) \cap \Lambda_2 = \emptyset$. Similarly, all events entering $R$ also enter $R_2$, or do not appear in $\Lambda_2$, as well as for the exiting events. Hence $R_1 \leftrightarrow R_2$ and therefore $R \in D$, which contradicts our initial assumption. □

In Property 5.5, we have shown that if we have two compatible transition systems $TS_1$ and $TS_2$, then we can calculate the regions of the sum of $TS_1$ and $TS_2$, using the regions of the individual transition systems. If we have a large collection of compatible transition systems, then adding two of them up to a new one yields a transition system which is compatible with all others as shown in Property 5.4. Since we have provided an algorithm in Definition 4.1 to translate all traces of a process log to a collection of compatible transition systems, we have constructed an iterative algorithm for the calculations presented in Section 4.

With Property 5.6, we conclude this section. Property 5.6 shows that the iterative algorithm yields the same transition system as Definition 4.2. Combining this with Property 5.4 and Property 5.5 leads to the conclusion that the set of regions resulting from our iterative approach indeed yields the set of regions of the transition system obtained by applying Definition 4.2 directly.

**Property 5.6. (Iterative approach works)** Let $T$ be a set of log events, let $W$ be a globally complete process log over $T$, i.e., $W \in \mathcal{P}(T^*)$ and let $TS(W) = (S, \Lambda, \rightarrow)$ be a transition system. Furthermore let $TS' = \bigoplus_{\sigma \in W} TS(\sigma) = (S', \Lambda', \rightarrow')$ be the transition system gained by adding all transition systems corresponding to each instance. We show that $TS = TS'$.

**Proof.**

- From Definition 4.2, we know that $S = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} S_\sigma$ and,
  from Definition 5.3, we know that $S' = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} S_\sigma$, hence $S = S'$,
- From Definition 4.2, we know that $\Lambda = T$. Furthermore, from Definition 5.3, we know that $\Lambda' = \bigcup_{\sigma \in W} \{t \mid t \in \sigma\}$. Assuming that for all $t \in T$ there is at least one $\sigma \in W$, such that $t \in \sigma$, we know that $\Lambda = \Lambda'$,
- From Definition 4.2, we know that $\rightarrow = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} \rightarrow_\sigma$ and,
  from Definition 5.3, we know that $\rightarrow' = \bigcup_{\substack{\sigma \in W \\ TS_\sigma = (S_\sigma, \Lambda_\sigma, \rightarrow_\sigma)}} \rightarrow_\sigma$, hence $\rightarrow = \rightarrow'$.

□

Note that by showing that the iterative approach works in Property 5.6, we can apply the synthesis algorithm of Definition 4.3 on the set of regions that results from the iterative algorithm.

### 5.1 Complexity

In Subsection 4.2, we mentioned that the synthesis algorithm of Definition 4.3 requires the full transition system to be built in memory. The space required to do so is obviously linear in the size of the log. However, our experience with process mining has shown that typical process logs found in practice are too big to be stored in memory.

In process mining, all algorithms have to make a trade-off between computation time on the one hand and space-requirements on the other. In [23] for example, a genetic-algorithm approach toward process mining is shown, where the algorithm scales linearly in the size of the log, i.e. if a log contains twice the number of cases, but the same number of different events, the algorithm is twice as slow, but requires the same amount of memory.

However, when a full transition system is stored in memory, the calculation of only *minimal* regions is simpler than in our iterative approach, i.e. using a breadth-first search, all minimal regions could be found, without considering larger regions. Our iterative approach requires all regions to be calculated, after which the minimal regions need to be found in the set of all regions. Therefore, the computation *time* is larger with our iterative approach.

In fact, it is not too hard to see that our iterative approach scales linearly in the size of the log, i.e. by adding more cases, while keeping the number of different activities equal, we need more processing time, but not more memory.

## 6 Tool Support in ProM

The (Pro)cess (M)ining framework *ProM* has been developed as a completely plugable environment for process mining and related topics. It can be extended by simply adding plug-ins, and currently, more than 140 plug-ins have been added. The ProM framework has been described before in [17, 28] and of course the web site www.processmining.org.

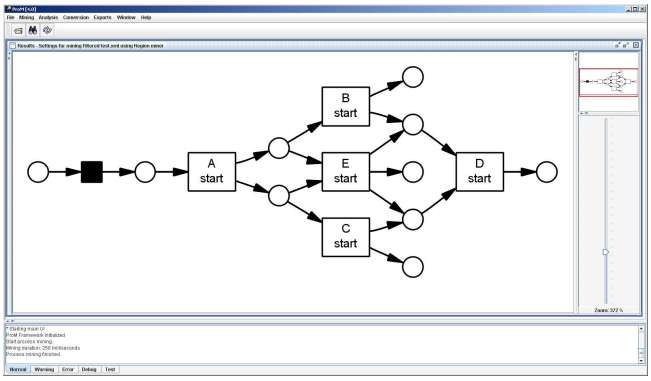


**Fig. 7.** A screenshot of ProM, showing the Petri net of Figure 6.

In the context of this paper, the region miner was developed, that implements the iterative algorithm described in Section 5. Figure 7 shows a screenshot of ProM, clearly showing the Petri net of Figure 6 that was derived from the process log of Table 1 using the region miner. Note that the plugin automatically inserts a common first step, which is shown as a black transition in Figure 7. Note that the model of Figure 7 contains many output places. These places could be removed using an algorithm tailored toward removing so-called *implicit places*, i.e. places that do not contribute to the behaviour of the Petri net.

Even though our algorithm does not guarantee that the resulting transition system is an elementary transition system, the result can still be very insightful. Consider for example a log with three cases, i.e. *case*1: $A$,$B$,$D$; *case*2: $A$,$C$,$D$ and *case*3: $A$,$B$,$C$,$D$. Figure 8 shows the result of both the $\alpha$-algorithm (top) and the region miner (bottom) on this log in ProM. Note that the result of the $\alpha$-algorithm can only replay 1 out of three cases, whereas the result of the region miner can replay all cases.

Figure 9 shows why the log with three instances does not result in an elementary transition system, i.e. the three highlighted states all appear in all pre-regions of transition $D$ (there is only one minimal pre-region of $D$, which is highlighted as well). Therefore, in all the highlighted states $(case1, 0)$, $(case2, 0)$ and $(case3, 0)$ transition $D$ is assumed to be enabled.

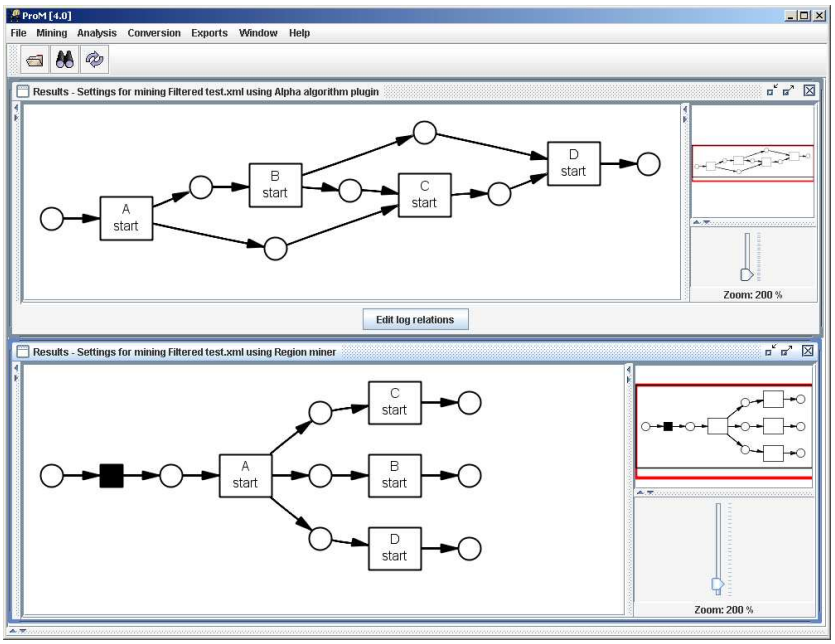


**Fig. 8.** A screenshot of ProM, showing the result of the $\alpha$-algorithm and the region approach.

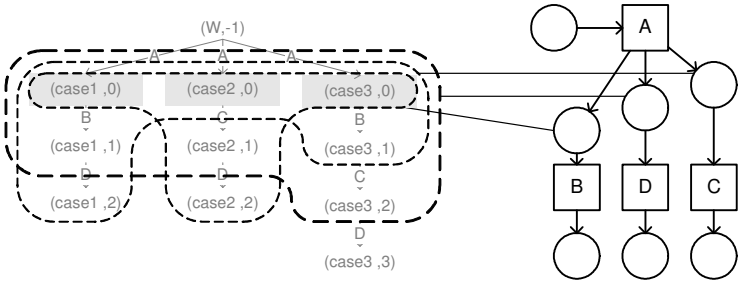


**Fig. 9.** A non-elementary transition system with some of its regions.

## 7 Conclusions and Future Work

Using the algorithm described in this paper, we can use the Theory of Regions in process discovery. The result is that, given an event log, a Petri net is obtained for which (1) each trace in the log is a firing sequence in the Petri net and (2) each firing sequence in the Petri net is a trace in the log. Furthermore, we have shown that the algorithm can be applied iteratively, thus reducing the space-complexity, which is commonly accepted to be the bottle neck for many process mining algorithms.

The biggest problem of the Theory of Regions in the context of process discovery corresponds to an advantage in the Theory of Regions domain. The resulting Petri net mimics the behaviour of the log *exactly*, i.e. the Theory of Regions was developed to generate a *compact* representation of an elementary transition system in terms of a Petri net. Although the result of our approach on non-elementary transition system does imply that the resulting Petri net allows for more behaviour, this is still very different from the goal of process mining, i.e. to generate a process model that is an *abstract* representation of the process log.

Despite this downside however, we see many future applications for the Theory of Regions in the context of process mining [1]. In our current approach for example, we have assumed that only the initial state of the transition systems is known. However, we foresee many other ways to derive state information from a process log, i.e. for example by assuming that the state is determined by the set of activities performed within a case so-far.

## References

1. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Gunther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.
2. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume

3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.

3. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
4. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
5. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
6. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
7. E. Badouel and P. Darondeau. Trace nets and process automata. *Acta Informatica*, 32:647–679, 1995.
8. E. Badouel and Ph. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets Part 1 (Basic Models)*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer-Verlag, Berlin, 1998.
9. L. Bernardinello. Synthesis of net systems. In M. Ajmone Marsan, editor, *Proceedings of the 14th Confrence on the Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 89–105. Springer Verlag, 1993.
10. N. Busi and R. Gorrieri. A survey on non-interference with petri nets. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 328–344. Springer, 2004.
11. N. Busi and G.M. Pinna. Synthesis of nets with inhibitor arcs. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR'97 Conference Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 151–165. Springer Verlag, 1997.
12. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
13. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
14. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A region-based theory for state assignment in speed- independent circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 16(8):793–812, 1997.
15. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
16. J. Desel and W. Reisig. The synthesis problem of petri nets. *Acta informatica*, 33:297–315, 1996.
17. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
18. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.

19. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
20. P. W. Hoogers, H. C. M. Kleijn, and P. S. Thiagarajan. An event structure semantics for general Petri nets. *Theoretical Computer Science*, 153(1–2):129–170, 1996.
21. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com, 2002.
22. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
23. A.K.A. de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.
24. M. Mukund. Petri nets and step transition systems. *International Journal on Foundation of Computing Science*, 3(4):443–478, 1992.
25. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
26. M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96(1):3–33, 1992.
27. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
28. H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM Framework. In *EMOI workshop*, 2006.
29. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.