A Tour in Process Mining: From Practice to Algorithmic Challenges

Wil van der Aalst[1], Josep Carmona[2], Thomas Chatain[3], and Boudewijn van Dongen[4]

[1] Process and Data Science group, RWTH Aachen University (Germany)
[2] Computer Science Department, Universitat Politècnica de Catalunya, Barcelona (Spain)
jcarmona@cs.upc.edu
[3] LSV, ENS Paris-Saclay, CNRS, Inria, Université Paris-Saclay, Cachan (France)
chatain@lsv.fr
[4] Eindhoven University of Technology, Eindhoven, The Netherlands
b.f.v.dongen@tue.nl

Abstract. Process mining seeks the confrontation between modeled behavior and observed behavior. In recent years, process mining techniques managed to bridge the gap between traditional model-based process analysis (e.g., simulation and other business process management techniques) and data-centric analysis techniques such as machine learning and data mining. Process mining is used by many data-driven organizations as a means to improve performance or to ensure compliance. Traditionally, the focus was on the discovery of process models from event logs describing real process executions. However, process mining is not limited to process discovery and also includes conformance checking. Process models (discovered or hand-made) may deviate from reality. Therefore, we need powerful means to analyze discrepancies between models and logs. These are provided by conformance checking techniques that first align modeled and observed behavior, and then compare both. The resulting alignments are also used to enrich process models with performance related information extracted from the event log. This tutorial paper focuses on the control-flow perspective and describes a range of process discovery and conformance checking techniques. The goal of the paper is to show the algorithmic challenges in process mining. We will show that process mining provides a wealth of opportunities for people doing research on Petri nets and related models of concurrency.

1 Introduction to Process Mining

This tutorial paper is based on a tutorial given at Petri Nets 2017 in Zaragoza (Spain) on Tuesday, June 27th, 2017 and a tutorial given at Petri Nets 2018 in Bratislava (Slovakia) on Tuesday, June 26th, 2018. The goal of these two tutorials was to introduce the topic of process mining for people with a Petri nets background and to show the exciting algorithmic challenges provided by the various process mining tasks. Although process mining is widely used in

industry, there are still many open research questions that require knowledge of both process science (e.g., formal methods and concurrency theory) and data science (e.g., data mining, machine learning, and statistics). To limit the scope, we focus on control flow and Petri nets as a representation.

1.1 Opportunities Provided by Event Data

Modeling behavior is valuable, but is often based on (unrealistic) simplifying assumptions. Simulation models and specifications can be used to get process insights, but these insights depend on the assumptions and abstraction used while modeling. Fortunately, when it comes to existing processes and systems, we no longer need to rely on modeling only. There is an abundance of event data. In the book [1], the term *Internet of Events* (IoE) is used to refer to the different types of event data readily available. These include: the *Internet of Content* (traditional databases, web pages, e-books, newsfeeds, movies, music, etc.), the *Internet of People* (e-mail, Facebook, Twitter, forums, LinkedIn, etc.), the *Internet of Things* (smart homes, high-tech systems, Industry 4.0, etc.), and the *Internet of Locations* (smartphones, wearables, etc.). Events may take place inside machines, enterprise information systems, hospital information systems, social networks, and transportation systems. Events may be "life events", "machine events", or "organization events". Process mining aims to exploit event data in a meaningful way, for example, to provide insights, identify bottlenecks, anticipate problems, record policy violations, recommend countermeasures, and streamline processes. Event data are like the breadcrumbs in the fairy tale of Hansel and Gretel. When stored properly, we can use them to reconstruct the real behavior.

Table 1 shows a small fragment of a larger *event log* describing 16218 events related to 1266 cases. The events in this log correspond to the handling of orders. Each row corresponds to two *events*, i.e., the *start* of an activity instance and the *completion* of an activity instance. Each *activity instance* (i.e., row) describes the execution of an activity for a particular order. The first column refers to the order number. The second column shows the activity name. The next two columns show the start time and end time of an activity. There are also additional attributes such as the person executing the event.

Event data, as shown in Table 1, can be used to *discover process models* best describing the behavior observed. This is similar to discovering a decision tree based on labeled instances, i.e., instances having descriptive attributes and a class label. However, in process mining, the instances correspond to events referring to cases (i.e., process instances), activities, timestamps, etc. Process discovery starts from the situation without a model and just event data. Given a process model, it is also possible to *check conformance*. Conformance checking uses as input both modeled and observed behavior. Discrepancies can be detected by replaying the event data on the process model. The process model may be hand-made or discovered using some process discovery technique. Next to process discovery and conformance checking, there are many additional process mining techniques using both event data and models. For example, events data can be

Table 1. Fragment of an event log with 16218 events related to 1266 cases.

order number	activity	start time	end time	resource	product	prod-price	quantity	address
...	...	...	...	...	...	...	...	...
989	place order	10/07/2015 05:10	10/07/2015 05:25	Sophia	APPLE iPhone 6s 64 GB	858.0	2	NL-7742XG-17
984	send invoice	10/07/2015 08:47	10/07/2015 08:57	Jack	SAMSUNG Core Prime G361	135.0	2	NL-9468HG-14
810	prepare delivery	10/07/2015 08:50	10/07/2015 08:53	Emma	APPLE iPhone 6s Plus 64 GB	969.0	4	NL-7875EH-26
898	confirm payment	10/07/2015 08:56	10/07/2015 09:02	Lily	SAMSUNG Galaxy S4	329.0	4	NL-7944BB-6
858	prepare delivery	10/07/2015 09:01	10/07/2015 09:04	Emma	SAMSUNG Galaxy J5	219.99	1	NL-7823JJ-7
990	place order	10/07/2015 09:41	10/07/2015 09:51	Sophia	MOTOROLA Moto G	199.0	4	NL-9514BV-16
838	confirm payment	10/07/2015 09:56	10/07/2015 10:03	Jack	SAMSUNG Core Prime G361	135.0	3	NL-7826GD-9
869	prepare delivery	10/07/2015 10:39	10/07/2015 10:42	Emma	SAMSUNG Galaxy S4	329.0	1	NL-7742XG-17
875	prepare delivery	10/07/2015 10:42	10/07/2015 10:46	Aiden	SAMSUNG Core Prime G361	135.0	4	NL-9407EM-35
898	make delivery	10/07/2015 10:44	10/07/2015 11:06	Aubrey	SAMSUNG Galaxy S4	329.0	4	NL-7944BB-6
914	send reminder	10/07/2015 11:38	10/07/2015 11:58	Abigail	MOTOROLA Moto G	199.0	3	NL-7828AM-11a
991	place order	10/07/2015 11:41	10/07/2015 15:32	Lucas	SAMSUNG Core Prime G361	135.0	4	NL-7905BC-40a
888	pay	10/07/2015 11:49	10/07/2015 11:54	Lily	APPLE iPhone 6 16 GB	639.0	3	NL-7821AC-3
977	send invoice	10/07/2015 11:56	10/07/2015 12:01	Madison	SAMSUNG Galaxy S4	329.0	3	NL-7908XB-46
889	pay	10/07/2015 12:03	10/07/2015 12:09	Lily	SAMSUNG Galaxy S4	329.0	2	NL-7948BX-10
883	make delivery	10/07/2015 12:09	10/07/2015 12:37	Aubrey	MOTOROLA Moto G	199.0	1	NL-7751AR-19
912	send reminder	10/07/2015 12:34	10/07/2015 12:46	Abigail	APPLE iPhone 6 16 GB	639.0	1	NL-9402NV-25
823	confirm payment	10/07/2015 12:56	10/07/2015 13:04	Lily	SAMSUNG Galaxy S4	329.0	7	NL-7943MC-4
863	prepare delivery	10/07/2015 12:59	10/07/2015 15:21	Lucas	APPLE iPhone 6 16 GB	639.0	2	NL-7905AX-38
860	make delivery	10/07/2015 13:04	10/07/2015 13:32	Aubrey	SAMSUNG Galaxy S6 32 GB	543.99	3	NL-7823JJ-7
992	place order	10/07/2015 13:41	10/07/2015 13:51	Jacob	SAMSUNG Galaxy S4	329.0	3	NL-7948DN-12a
900	send reminder	10/07/2015 13:45	10/07/2015 13:59	Abigail	APPLE iPhone 6 16 GB	639.0	2	NL-9521KJ-34
905	send reminder	10/07/2015 13:52	10/07/2015 14:08	Luke	APPLE iPhone 5s 16 GB	449.0	4	NL-7942GT-2
867	prepare delivery	10/07/2015 14:36	10/07/2015 14:43	Sophia	SAMSUNG Galaxy S6 32 GB	543.99	2	NL-7908XB-46
...	...	...	...	...	...	...	...	...

used to *enrich* or *repair* process models. It is also possible to *predict* performance or compliance. See [1] for a more complete overview of process discovery. The book [2] focuses on conformance checking.

1.2 Control Flow as the Backbone of Any Process

In this paper, we focus on *control flow*. This means that we ignore most of the attributes in Table 1. Initially, we are only interested in the ordering of activities within a *case* (i.e., process instance). To illustrate this, consider a particular order, e.g., order 889. There are six activity instances in Table 1 related to this order. Each activity instance corresponds to two events: start and complete. Suppose we focus on the start events. This means that order 889 is described by six events (rather than twelve). Table 2 shows these six events. Note that we only show the first three columns. Table 2 describes a sequence of activities for one case, namely: $\langle place\ order, send\ invoice, pay, prepare\ delivery, make\ delivery, confirm\ payment \rangle$. Note that the timestamps are only used to order the events.

Table 2. The events of type "start" for case 889.

order number	activity	timestamp
889	place order	23/06/2015 19:45
889	send invoice	24/06/2015 18:23
889	pay	10/07/2015 12:03
889	prepare delivery	21/07/2015 11:22
889	make delivery	22/07/2015 13:00
889	confirm payment	23/07/2015 10:53

Each of the 1266 cases in Table 1 can be described as such a *trace*. There are 503 cases that correspond to the trace $\langle place\ order, send\ invoice, pay, prepare\ delivery, make\ delivery, confirm\ payment \rangle$ (including order 889). Table 3 shows the frequency of each observed trace. As we will see later, the control-flow aspect of an event log Table 3 can be formally represented as a mulsetti-set of activity sequences.

Clearly, Table 3 contains only a fraction of the information in the original event log (compare with the fragment in Table 1). However, this information is sufficient to construct a control-flow model, e.g., a Petri net. To further simplify things, assume that we abstract from activity $sr = send\ reminder$, i.e., we create a new log without this activity and feed it to various discovery algorithms. Figure 1 shows a few discovered process models that will be discussed later.

Control-flow models such as the ones shown in Figure 1 are just the starting point for process mining. By replaying the event log on the discovered model, one can show bottlenecks, etc. Actually, the control-flow model may be extended with additional perspectives: the organizational perspective ("What are the organizational roles and which resources are performing particular activities?"), the case

Table 3. Event log represented as a multi-set of traces using the short names $po = place\ order$, $si = send\ invoice$, $py = pay$, $pd = prepare\ delivery$, $md = make\ delivery$, $cp = confirm\ payment$, $co = cancel\ order$, $sr = send\ reminder$.

trace	frequency
$\langle po, si, py, pd, md, cp \rangle$	503
$\langle po, si, sr, py, pd, md, cp \rangle$	247
$\langle po, si, sr, sr, co \rangle$	141
$\langle po, si, sr, sr, py, pd, md, cp \rangle$	139
$\langle po, si, py, pd, cp, md \rangle$	135
$\langle po, si, sr, py, pd, cp, md \rangle$	57
$\langle po, si, sr, sr, py, pd, cp, md \rangle$	36
$\langle po, py, si, pd, md, cp \rangle$	6
$\langle po, py, si, pd, cp, md \rangle$	2
total	1266

perspective ("Which characteristics of a case influence a particular decision?"), and the time perspective ("Where are the bottlenecks in my process?"). These additional perspectives are very important from a practical point of view.

Analysts applying process mining are interested in questions such as:

– What is the main bottleneck in the process and is this caused by particular resources?
– What do the incompliant cases have in common? Do they lead to higher costs?
– Will we be able to handle 95% of the cases in two hours in the coming days?
– Does the workload have an effect on the durations of activities?

For all of these questions, we first need to have a control-flow model. Moreover, we need to be able to link events in the log to activities in the process model in order to discuss bottlenecks, deviations, decisions, etc. Transitional machine learning, data mining, and optimization techniques can be used once the control-flow model is in place and the event log is aligned with the model. However, these techniques cannot be used to handle the control-flow perspective. Therefore, this paper focuses on control-flow.

1.3 Process Discovery ≠ Synthesis

Data science techniques need to deal with uncertainty and incompleteness [3]. Suppose that we learn a decision tree that adequately predicts that one is less likely to claim insurance when being middle-aged and female and that one is more likely to claim insurance when being young and male. However, even a properly constructed decision tree will not be able to classify things correctly and we accept that there will be young males not claiming insurance or middle-aged females that do claim insurance. Moreover, we do not expect to see all combinations in our input data. For example, when customers are described by 10 different attributes age, gender, income, region, brand, etc., we cannot
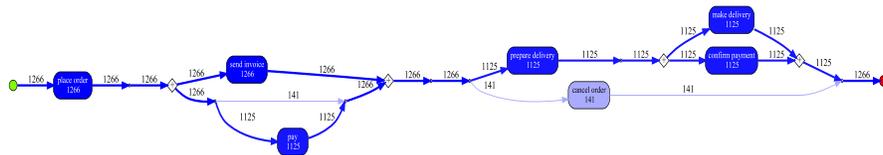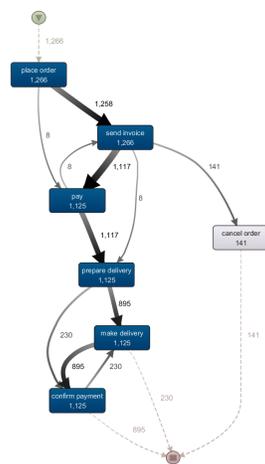
(a) Process model discovered using the inductive miner.
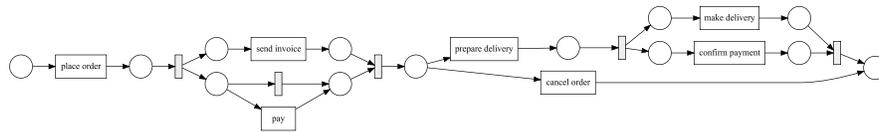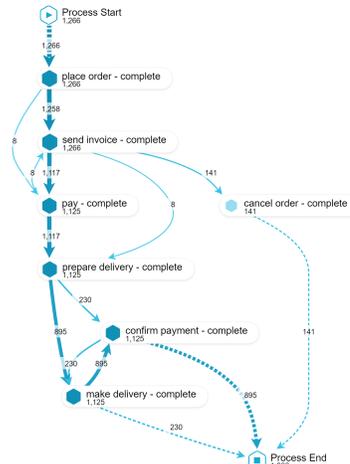

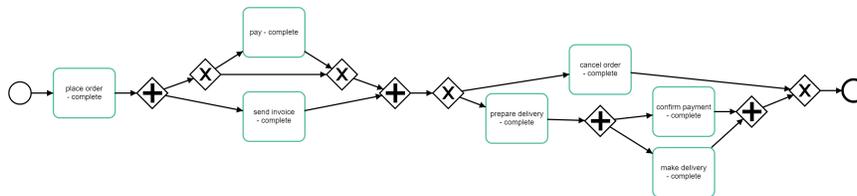(b) Petri net discovered using the inductive miner.
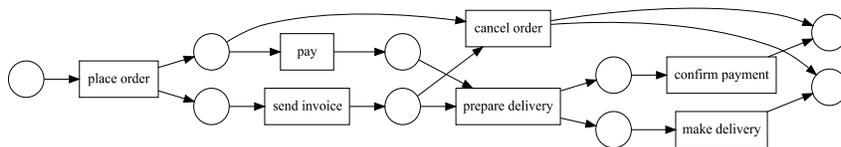
(c) Petri net discovered using state-based regions.

(d) Process model discovered using Disco.

(e) Process model discovered using Celonis.

(f) BPMN model discovered using Celonis.

Fig. 1. Different process models discovered using plug-ins of ProM (models (a), (b), and (c)), Disco (model (d)), and Celonis (models (e) and (f)). Process models (d) and (e) are so-called directly follows graphs. Process models (a), (b), and (f) all describe the same process, but use different notations. Process model (c) describes the process best, because *pay* and *cancel order* are mutually exclusive.

expect to see all combinations in our training data. Despite these limitations, the decision tree still provides valuable insights. When performing process discovery, we are facing similar challenges. The event log is just a sample and the fact that something happens in the logs once does not imply that the model should allow for it. Consider for example the 1266 cases in Table 3. Some traces are very frequent, e.g., $\langle po, si, py, pd, md, cp \rangle$ occurred 503 times. Other traces are very infrequent, e.g., $\langle po, py, si, pd, cp, md \rangle$ happened only twice. Traces in an event log typically follow a Pareto distribution, i.e., some are frequent, but many may be infrequent (for example, 80% of the log is described by 20% of the trace variants). When using a larger process model with many activities, one often witnesses that many traces in the event log are unique. Taking a different sample of the same process will lead to different unique variants. For example, if we take only the first 500 cases from the log shown in Table 3, we may not see $\langle po, py, si, pd, cp, md \rangle$. Moreover, if we observe the same process for a longer time, we will see cases that are not in Table 3 (e.g., new trace variants with three or more reminders). We may also be just interested in the dominant behavior. For example, to resolve a bottleneck, we may want to focus on the most frequent behavior. However, when investigating fraud we may be most interested in the least frequent behavior.

Researchers working on formal methods and concurrency theory often have problems dealing with uncertainty and incompleteness. Most synthesis approaches assume that the input provides a *full* and *unambiguous* description of *all* possible behavior [4]. When performing process mining in reality such assumptions are very unrealistic. To reason about the correctness of process mining results one needs to use a data science approach (e.g., cross-validation using test and training data) and apply notions such as precision and recall to assess the quality.

1.4 Industrial Uptake of Process Mining and Commercial Software

Around 2002, only simple stand-alone process mining tools were available (e.g., *MiMo*, *EMiT*, *Little Thumb*, and *InWolvE*) [1]. This triggered the development of the *ProM framework*. In 2004, the first fully functional version of the ProM framework was released. Since then, ProM has been the de facto standard for process mining research. ProM provides a "plug-able" open-source platform where developers can contribute new analysis techniques in the form of plug-ins. Currently, there are over 1500 plug-ins. ProM served as an example for a "wave" of commercial process mining tools.

Reports by Gartner [5] and Forrester [6] report on the uptake of process mining in industry. There are more than 25 commercial tools supporting process mining. Examples include: *Disco* (Fluxicon), *Celonis Process Mining* (Celonis), *ProcessGold Enterprise Platform* (ProcessGold), *QPR ProcessAnalyzer* (QPR), *SNP Business Process Analysis* (SNP AG), *minit* (Gradient ECM), *myInvenio* (Cognitive Technology), *Everflow* (Accelera Labs), *ProDiscovery* (Puzzeldata), *PAFnow* (Process Analytics Factory), *Stereologic* (Stereologic), *ARIS Process Mining* (Software AG), *Mehrwerk Process Mining* (Mehrwerk), *Logpickr Process*

Explorer (Logpickr), and *Lana Process Mining* (Lanalabs) [1]. See the website of the IEEE Task Force on Process Mining for examples of successful case studies [7]. For example, within Siemens there are currently over 2,500 active users of *Celonis Process Mining*. Siemens reported savings of "double-digit millions euros" as a result of the worldwide application of process mining [8]. Celonis is a process mining start-up founded in 2011 that is now valued over one billion dollar. The many tools available and the widespread adoption illustrate the relevance of process mining. However, at the same time, there are many challenges, as demonstrated in this paper.

As described in Gartner [5], there is an increasing need for process mining capabilities that go beyond process discovery. Initially, commercial tools focused on process discovery and often discovery capabilities were limited to the so-called *directly-follows graph*. In such a graph, nodes represent activities and arcs represent a simplistic view on causality. Activities a and b are connected if a is frequently followed by b. In the event log used before, $po = place\ order$ is directly followed by $si = send\ invoice$ 1258 times, and $po = place\ order$ is directly followed by $py = pay$ only 8 times. Depending on the threshold set, an arc connecting *place order* to *pay* is added to the directly-follows graph.

Figure 1 shows six process models discovered based on the event log described in tables 1 and 3 (without activity *send reminder*). Figure 1(a) shows a screenshot of ProM's visual inductive miner [9]. This model can be converted into a Petri net as shown in Figure 1(b). Note that the model allows for the trace where *pay* is followed by *cancel order*, i.e., the model is "underfitting" and allowing for unseen behavior (in the event log there are no orders for which both *pay* and *cancel order* occurred). Figure 1(c) shows a model generated by ProM based on state-based regions. This Petri net does not allow for the unseen behavior mentioned before (there is an exclusive choice between *pay* and *cancel order*). Figure 1(d) shows a directly-follows graph generated using Disco, the software from Fluxicon. The directly-follows graph is annotated with the frequencies. Using two sliders, it is possible to remove infrequent arcs and activities. Figure 1(e) shows a similar directly-follows graph generated using Celonis. Celonis also supports the basic inductive mining algorithm [1, 10]. The result is presented as a BPMN model as shown in Figure 1(f). Note that models (a), (b), and (f) are semantically equivalent, but use different notations. Each of the models in Figure 1 defines a language, i.e., the (possibly infinite) set of traces accepted by the model. Ideally, the model accepts most of the traces in the event log (but not necessarily all, e.g., outlier behavior may not be accepted) and does not accept behavior that is unlikely given the event log. Later, these notions will be defined more precisely.

Although the initial focus of commercial tools was on discovery, vendors started to add functionality related to conformance checking (i.e., monitoring deviations by comparing model and log), social network/organizational mining, root-cause analysis using machine learning, and case prediction. For example, it is possible to find possible causes for bottlenecks and compliance problems and generate statements like "The extreme waiting times are caused by unnec-

essary rework involving an external party and under-staffing of the backoffice on Fridays" and "Most of the compliance violations were caused by this new manager that approved requests without the necessary checks". Note that these more advanced questions have been researched for over more than a decade [1]. All of the analysis features that can be found in today's commercial tools, had been implemented in ProM years before. Note that, currently, ProM provides over 1500 plug-ins providing a wide range of analysis techniques.

In this paper, we focus on conformance checking and limit the scope to control-flow [1, 2]. Given a trace $\sigma_1 = \langle po, si, pd, py, md, cp \rangle$ and a process model like Figure 1(c), we would like to decide if the trace fits the model and if not, diagnose the difference(s). In σ_1, $pd = $ *prepare delivery* is performed before the $py = pay$ which is impossible according to the Petri net in Figure 1(c). In $\sigma_2 = \langle po, py, si, py, pd, md, cp \rangle$, the order is paid twice. In $\sigma_3 = \langle po, py, si, md, cp \rangle$, the order was delivered without the mandatory preparation step. Given an event log with possibly millions of events, conformance checking techniques need to provide aggregate diagnostics (e.g., the activity was skipped 1500 times).

1.5 Killer App for Petri Nets

The process mining discipline is highly relevant (see the uptake of commercial process mining tools and activities) and provide interesting scientific challenges. Progress in this young scientific discipline has been remarkable and this resulted in powerful techniques that are highly scalable. However, many of the problems identified (process discovery, conformance checking, etc.) are notoriously difficult and have not been solved satisfactorily. Process mining provides a great opportunity for Petri net researchers. When studying real-life processes, concurrency should be a starting point for analysis and not added as an afterthought (locality of actions). Many other modeling approaches start from a sequential view on the world and then add special operators to introduce concurrency. Petri nets are inherently concurrent. Although Petri nets are often seen as a procedural language, in essence, Petri nets are declarative. A Petri net without any places and a non-empty set of transitions allows for any behavior involving the activities represented by these transitions. Adding a place is like introducing a constraint. The idea that transitions (modeling activities or actions) are independent (i.e., concurrent) unless specified otherwise is foundational ([11]).

Most data science approaches do not consider behavioral aspects and when they do, they typically use sequential models (e.g., sequential patterns, Markov models, etc.). As the evolution of the process mining discipline shows, the combination of event data and Petri nets is very powerful. Therefore, we encourage Petri-net researchers to tackle the open problems identified in this paper. However, as Section 1.3 describes this is not "business as usual": One needs to approach the problem from a data-science perspective.

1.6 Outline

The remainder of this paper is organized as follows. Section 2 introduces preliminaries, including basic notations, event logs, and Petri nets. Section 3 discusses the challenges related to process discovery and presents inductive and region-based discovery techniques. Section 4 is devoted to conformance checking. The focus here is on alignment-based techniques. Alignments relate seen behavior to modeled behavior even when there are deviations. Such alignments are used for a range of analyses (bottleneck analysis, predictions, etc.). Section 5 focuses on the quality of a process model in the context of a given event log. Notions such as fitness, precision, generalization, and simplicity are discussed. Section 6 concludes the paper.

2 Preliminaries

2.1 Mathematical Notation

A multi-set is like a set in which each element may occur multiple times. For example, $[a, b^2, c^3, d^2, e]$ is the multi-set with nine elements: one a, two b's, three c's, two d's, and one e. The following three multi-sets are identical: $[a, b, b, c^3, d, d, e]$, $[e, d^2, c^3, b^2, a]$, and $[a, b^2, c^3, d^2, e]$. Formally, $\mathbb{B}(A) = A \to \mathbb{N}$ is the set of multi-sets (bags) over a finite domain A, i.e., $X \in \mathbb{B}(A)$ is a multi-set, where for each $a \in A$, $X(a)$ denotes the number of times a is included in the multi-set. For example, if $X = [a, b^2, c^3]$, then $X(b) = 2$ and $X(e) = 0$.

The sum of two multi-sets $(X \uplus Y)$, the difference $(X \setminus Y)$, the presence of an element in a multi-set $(x \in X)$, and the notion of subset $(X \leq Y)$ are defined as usual.

For a given set A, A^* is the set of all finite sequences over A. A finite sequence over A of length n is a mapping $\sigma \in \{1, \ldots, n\} \to A$. Such a sequence is represented by a string, i.e., $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. $|\sigma|$ denotes the length of the sequence, i.e. $|\sigma| = n$. $\sigma \oplus a' = \langle a_1, \ldots, a_n, a' \rangle$ is the sequence with element a' appended at the end. Similarly, $\sigma_1 \oplus \sigma_2$ appends sequence σ_2 to σ_1 resulting a sequence of length $|\sigma_1| + |\sigma_2|$.

$hd^k(\sigma) = \langle a_1, a_2, \ldots, a_{k \min n} \rangle$, i.e., the "head" of the sequence consisting of the first k elements (if possible). Note that $hd^0(\sigma)$ is the empty sequence and for $k \geq n$: $hd^k(\sigma) = \sigma$. $pref(\sigma) = \{hd^k(\sigma) \mid 0 \leq k \leq n\}$ is the set of prefixes of σ.

$tl^k(\sigma) = \langle a_{(n-k+1) \max 1}, a_{k+2}, \ldots, a_n \rangle$, i.e., the "tail" of the sequence composed of the last k elements (if possible). Note that $tl^0(\sigma)$ is the empty sequence and for $k \geq n$: $tl^k(\sigma) = \sigma$.

$\sigma \uparrow X$ is the projection of σ onto some subset $X \subseteq A$, e.g., $\langle a, b, c, a, b, c, d \rangle \uparrow \{a, b\} = \langle a, b, a, b \rangle$ and $\langle d, a, a, a, a, a, a, d \rangle \uparrow \{d\} = \langle d, d \rangle$.

For any sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ over A, $\{a \in \sigma\} = \{a_1, a_2, \ldots, a_n\}$ and $[a \in \sigma] = [a_1, a_2, \ldots, a_n]$, e.g., if $\sigma = \langle d, a, a, a, a, a, a, d \rangle$, then $\{a \in \sigma\} = \{a, d\}$ and $[a \in \sigma] = [a^6, d^2]$.

2.2 Process Models as Petri Nets

Definition 1 ((Labeled) Petri net). *A (labeled) Petri Net [12] is a tuple $N = \langle P, T, \mathcal{F}, M_0, M_f, \Sigma, \lambda \rangle$, where P is the set of places, T is the set of transitions (with $P \cap T = \emptyset$), $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation, $M_0 \in \mathbb{B}(P)$ is the initial marking, $M_f \in \mathbb{B}(P)$ is the final marking, Σ is an alphabet of actions and $\lambda : T \rightarrow \Sigma \cup \{\tau\}$ labels every transition by an action, or as a silent action denoted by the symbol τ.*

A marking $M \in \mathbb{B}(P)$ is an assignment of a non-negative integer to each place. If k is assigned to place p by marking M (denoted $M(p) = k$), we say that p is marked with k tokens. Given a node $x \in P \cup T$, its pre-set and post-set are denoted by $^\bullet x$ and $x^\bullet$ respectively.

A transition t is *enabled* in a marking M when all places in $^\bullet t$ are marked. When a transition t is enabled, it can *fire* by removing a token from each place in $^\bullet t$ and putting a token to each place in $t^\bullet$. A marking M' is *reachable* from M if there is a sequence of firings $t_1 \ldots t_n$ that transforms M into M', denoted by $M[t_1 \ldots t_n\rangle M'$. We define the *language* of N as the set of *full runs* defined by $\mathcal{L}(N) := \{\lambda(t_1) \ldots \lambda(t_n) \mid M_0[t_1 \ldots t_n\rangle M_f\}$. A Petri net is *k-bounded* if no reachable marking assigns more than k tokens to any place. A Petri net is bounded if there exist a k for which it is k-bounded. A Petri net is *safe* if it is 1-bounded. A bounded Petri net has an *executable loop* if it has a reachable marking M and sequences of transitions $u_1 \in T^*$, $u_2 \in T^+$, $u_3 \in T^*$ such that $M_0[u_1\rangle M[u_2\rangle M[u_3\rangle M_f$.

2.3 Process Event Data

Processes are crucial to manage the operations in organizations. Often, processes are complex, due to many reasons: they can comprise a large number of activities, can have many decision points, there can be many participants working jointly on a case, etcetera. Also, there can be many work streams running in parallel, possibly with the support of multiple information systems and also external suppliers. Regardless of their complexity, processes in organizations have a common element: They generate data. We call the event data generated in a process *event logs*.

The collection and analysis of event logs opens various opportunities for improving the underlying process. Table 4 shows a simple event log, which contains information about the underlying process, including *data* in the form of *event attributes*. The event log underlying traces are $\{\langle a, b, c, d \rangle, \langle c, d, a, b \rangle, \langle b, a, d, c \rangle, \langle d, c, b, a \rangle\}$ and they correspond to 'case IDs' 1, 2, 3, and 4, respectively. We assume that the set of attributes is fixed and the function *attr* maps pairs of events and attributes to the corresponding values. For each *event e* the log contains the case ID $case(e)$, the activity name $act(e)$, and the set of attributes defined for e, e.g., $attr(e, timestamp)$. For instance, for the event log in Table 4, $case(e_7) = 2$, $act(e_7) = a$, $attr(e_7, timestamp) = $ "10-04-2015 10:28pm", and $attr(e_7, cost) = 19$.

Event	Case ID	Activity	Timestamp	Temperature	Resource	Cost	Risk
1	1	a	10-04-2015 9:08am	25.0	Martin	17	Low
2	2	c	10-04-2015 10:03am	28.7	Mike	29	Low
3	2	d	10-04-2015 11:32am	29.8	Mylos	16	Medium
4	1	b	10-04-2015 2:01pm	25.5	Silvia	15	Low
5	1	c	10-04-2015 7:06pm	25.7	George	14	Low
6	1	d	10-04-2015 9:08pm	25.3	Peter	17	Medium
7	2	a	10-04-2015 10:28pm	30.0	George	19	Low
8	2	b	10-04-2015 10:40pm	29.5	Peter	22	Low
9	3	b	11-04-2015 9:08am	22.5	Mike	31	High
10	4	d	11-04-2015 10:03am	22.0	Mylos	33	High
11	4	c	11-04-2015 11:32am	23.2	Martin	35	High
12	3	a	11-04-2015 2:01pm	23.5	Silvia	40	Medium
13	3	d	11-04-2015 7:06pm	28.8	Mike	43	High
14	3	c	11-04-2015 9:08pm	22.9	Silvia	45	Medium
15	4	b	11-04-2015 10:28pm	23.0	Silvia	50	High
16	4	a	11-04-2015 10:40pm	23.1	Peter	35	Medium

Table 4. An example event log

For the sake of understandability, in this paper we will focus on the data corresponding to the temporal relation of activities, i.e., the *control-flow* perspective of a process. The reader can find in the literature references to also explore process mining techniques that go beyond this perspective.

Formally, an event log is a collection of traces, where a trace may appear more than once. Formally:

Definition 2 (Event Log). *An event log $L \in I\!B(\Sigma^*)$ (over an alphabet of actions Σ) is a multiset of traces. $\sigma \in \Sigma^*$ is a possible trace.*

The event log in Figure 2 can be compactly represented as $L = [\langle po, si, py, pd, md, cp \rangle^{503}, \langle po, si, sr, py, pd, md, cp \rangle^{247}, \langle po, si, sr, sr, co \rangle^{141}, \langle po, si, sr, sr, py, pd, md, cp \rangle^{139}, \langle po, si, py, pd, cp, md \rangle^{135}, \langle po, si, sr, py, pd, cp, md \rangle^{57}, \langle po, si, sr, sr, py, pd, cp, md \rangle^{36}, \langle po, py, si, pd, md, cp \rangle^{6}, \langle po, py, si, pd, cp, md \rangle^{2}]$.

3 Discovering Process Models

This section focuses on process discovery and introduces various techniques. The goal is not to be complete, but to provide valuable insights into the different solution approaches.

3.1 Definition of Discovery

The input for process discovery is an event log $L \in I\!B(\Sigma^*)$. As mentioned before, we focus on control flow and leave out other perspectives (time, costs, data, etc.).

Process discovery algorithms take an event log as input and aim to output a process model that satisfies certain properties. To judge the quality of the discovered model the following four quality dimensions of process mining [1] are used:

- *fitness* (also called *recall*): the discovered model should allow for the behavior seen in the event log (avoiding "non-fitting" behavior),
- *precision*: the discovered model should not allow for behavior completely unrelated to what was seen in the event log (avoiding "underfitting"),
- *generalization*: the discovered model should generalize the example behavior seen in the event log (avoiding "overfitting"), and
- *simplicity*: the discovered model should not be unnecessarily complex.

The simplicity dimension refers to Occam's Razor: "one should not increase, beyond what is necessary, the number of entities required to explain anything". In the context of process mining, this is often operationalized by quantifying the complexity of the model (number of nodes, number of arcs, understandability, etc.). The other three dimensions typically abstract from the representation. This means that an event log $L \in \mathbb{B}(\Sigma^*)$ is a multiset of traces and a model $\mathcal{M} \subseteq \Sigma^*$ is simply seen as a set of traces (a language). Is this paper $\mathcal{M}$ corresponds to the language of a labeled accepting Petri net $N = \langle P, T, \mathcal{F}, M_0, M_f, \Sigma, \lambda \rangle$, i.e., $\mathcal{M} = \mathcal{L}(N)$.

Many conformance measures have been proposed throughout the years. There seems to be consensus on the four quality dimensions, but these can be operationalized in many different ways. In [13], 21 so-called conformance propositions were defined to discuss desirable properties of existing measures for recall, precision, and generalization. It has been shown that seemingly obvious conformance propositions are violated by existing approaches. Furthermore, [13] also shows the importance of probabilistic conformance measures that also take into account trace likelihoods in process models. However, to date, very few conformance measures exist that can actually support probabilistic process models.

Another approach to get handle on the problem of evaluating a process model in the context of an event log is to assume the existence of an underlying system $\mathcal{S} \subseteq \Sigma^*$ that generated the event log L. (Note that we assume $\mathcal{S}$ to be a language just like $\mathcal{M}$.). Process mining techniques aim at extracting a process model $\mathcal{L}(N) = \mathcal{M}$ from a log L with the goal to elicit the process underlying system $\mathcal{S}$. By relating the behaviors of L, $\mathcal{M}$ and $\mathcal{S}$, particular concepts can be defined [14]. A log is *incomplete* if $\mathcal{S} \setminus \{\sigma \in L\} \neq \emptyset$. A model $\mathcal{M}$ *fits* log L perfectly if $\{\sigma \in L\} \subseteq \mathcal{M}$. A model is *precise* in describing a log L if $\mathcal{M} \setminus \{\sigma \in L\}$ is small. A model $\mathcal{M} = \mathcal{L}(N)$ represents a *generalization* of log L with respect to system $\mathcal{S}$ if some behavior in $\mathcal{S} \setminus \{\sigma \in L\}$ exists in $\mathcal{M}$.

As shown in [1, 13], the above line of thinking is rather naive when it comes to real-life problems. The following properties make process discovery and the evaluation of process discovery results particulary challenging.

- In reality one will never know the underlying system $\mathcal{S}$. This is only possible in a simulation setting.
- One cannot witness negative examples, i.e., the event log does not show what could not happen.
- The event log only contains a tiny fraction of the set of all possible traces.
- If the model has loops, then the set $\mathcal{M} \setminus \{\sigma \in L\}$ is infinitely large and expressions such as $\mathcal{M} \setminus \{\sigma \in L\}$ do not make any sense.

– If one observes a system for a longer time, one can see new behaviors. This may be due to the large number of possible variants, the low probability of some variants, or concept drift (the underlying system changes over time). Hence, Murphy's law for process mining states that "anything is possible if one waits long enough". Therefore, one cannot assume the existence of a system oracle that acts as a binary classifier, and therefore it is vital to take frequencies, incompleteness, and probabilities into account to get the full picture [13].

The above challenges explain why a range of discovery approaches have been developed over time. In the remainder of this section we aim to provide insights into a few representative examples.

3.2 Process Discovery Using Inductive Mining

Process models discovered from event logs describe possible life-cycles of cases (i.e., process instances). Such models have a start and end. In terms of Petri nets, we are interested in models that have an initial marking $M_0 \in \mathbb{B}(P)$ and a final marking $M_f \in \mathbb{B}(P)$. All accepted traces correspond to paths from M_0 to M_f. However, Petri nets may be deadlocking or livelocking. It may even be the case that there is no accepting trace (i.e., M_f is not reachable from M_0). The notion of soundness was first introduced in the context of workflow nets [15], but can be relaxed and generalized to accepting Petri nets. Soundness cannot be decided locally, yet process discovery techniques may need to make local decisions. Therefore, most process discovery techniques may generate unsound models. One way to ensure soundness is to only consider block-structured process models. Inductive process discovery, as presented in [9, 10, 16, 17], uses *process trees* to ensure soundness. Moreover, the hierarchical nature of a process tree also enables a divide-and-conquer approach. Rather than using a single pass through the event log, it is also possible to try and break the problem into smaller problems. Inductive process discovery approaches split the event log recursively into smaller sublogs. For example, if one group of activities is preceded by another group of activities, but never the other way around, then we may deduce that these two groups are in a sequence relation. Subsequently, the event log is decomposed based on the two groups of activities. Next to the sequence relation, it is also possible to detect choices, concurrency, and loops, and split the log accordingly. This divide-and-conquer approach is repeated until each sublog refers to single activity. Leemans et al. developed a family of inductive mining techniques [9, 10, 16, 17]. Some of these techniques are tailored to dealing with huge event logs and process models, other techniques address challenges such as infrequent behavior and incompleteness of logs. In this section, we only consider the basic algorithm also described in [1].

Figure 2 shows a process tree. This process tree was discovered using the event log introduced in Section 1. Next to the visual tree representation, process trees also have a textual description $\rightarrow(po, \wedge(\times(\tau, py), si), \times(co, \rightarrow(pd, \wedge(md, cp))))$. The root node is of type $\rightarrow$ meaning that the three subtrees po, $\wedge(\times(\tau, py), si)$,
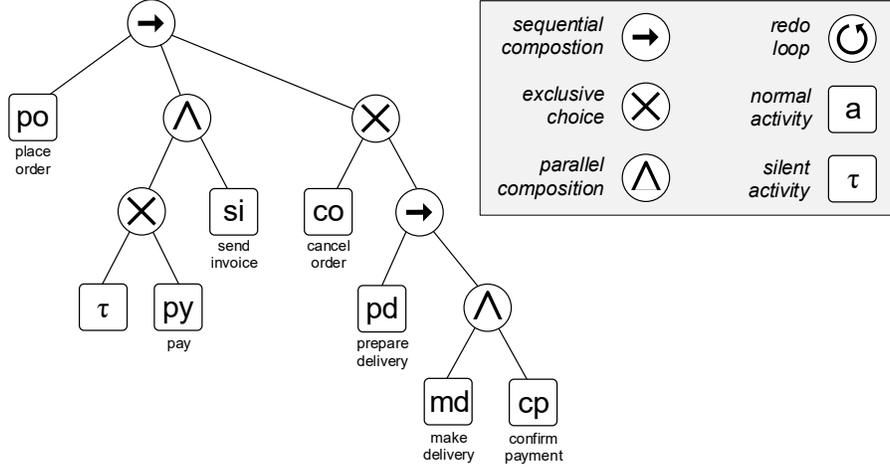
Fig. 2. Process tree $\rightarrow(po, \wedge(\times(\tau, py), si), \times(co, \rightarrow(pd, \wedge(md, cp))))$ discovered for the event log described in tables 1 and 3 (without activity $sr = send\ reminder$). The process tree corresponds to models (a), (b), and (f) in Figure 1.

and $\times(co, \rightarrow(pd, \wedge(md, cp)))$ are executed in sequence. The subtree in the middle $\wedge(\times(\tau, py), si)$ is of type $\wedge$ meaning that its two parts are executed concurrently. The subtree on the left $\times(co, \rightarrow(pd, \wedge(md, cp)))$ is of type $\times$ meaning that there is an exclusive choice between its two children.

Next to $\rightarrow$ (sequential composition), $\times$ (exclusive choice), and $\wedge$ (parallel composition), there is also the $\circlearrowleft$ (redo loop) operator. The redo loop operator $\circlearrowleft$ has at least two children. The first child is the "do" part and the other children are alternative "redo" parts. Process tree $\circlearrowleft(a, b, c)$ allows for traces $\{\langle a \rangle, \langle a, b, a \rangle, \langle a, c, a \rangle, \langle a, b, a, b, a \rangle, \langle a, c, a, c, a \rangle, \langle a, c, a, b, a \rangle, \langle a, b, a, c, a \rangle, \ldots\}$. Activity a is executed at least once and the process always starts and ends with a. The "do" part alternates with the "redo" parts b or c. When looping back either b or c is executed. The redo loop operator $\circlearrowleft$ is often used in conjunction with silent activity τ. For example, $\circlearrowleft(\tau, a, b, c, \ldots, z)$ allows for any trace (including the empty one) involving activities $a, b, c, \ldots, z$.

The same activity may appear multiple times in the same process tree. For example, process tree $\rightarrow(a, a, a)$ models a sequence of three a activities. From a behavioral point of view $\rightarrow(a, a, a)$ and $\wedge(a, a, a)$ are indistinguishable. Both allow for one possible trace: $\langle a, a, a \rangle$.

Definition 3 (Process tree). *Let $A \subseteq \Sigma$ be a finite set of activities with $\tau \notin \Sigma$. $\bigoplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ is the set of process tree operators. Process trees are defined inductively:*

- *if $a \in A \cup \{\tau\}$, then $Q = a$ is a process tree,*
- *if $n \geq 1$, $Q_1, Q_2, \ldots, Q_n$ are process trees, and $\oplus \in \{\rightarrow, \times, \wedge\}$, then $Q = \oplus(Q_1, Q_2, \ldots Q_n)$ is a process tree, and*

– *if $n \geq 2$ and $Q_1, Q_2, \ldots, Q_n$ are process trees,*
   then $Q = \circlearrowleft(Q_1, Q_2, \ldots Q_n)$ is a process tree.
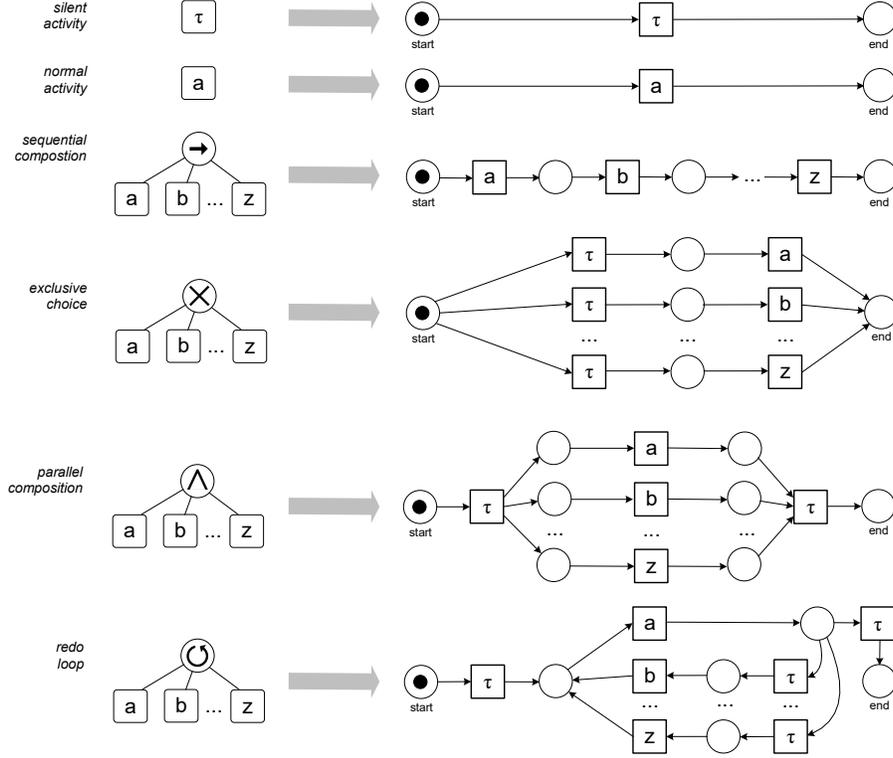

Fig. 3. Mapping process trees onto Petri to illustrate their semantics.

$\mathcal{L}(Q)$ is the *language* of a process tree Q. Formal definitions are provided in $[1, 9, 10, 16, 17]$. Here we only provide a mapping to Petri nets and some examples. Figure 3 shows the semantics of each operator in terms of Petri nets. The following examples further illustrate the process tree operators and their semantics: $\mathcal{L}(\tau) = \{\langle \, \rangle\}$, $\mathcal{L}(a) = \{\langle a \rangle\}$, $\mathcal{L}(\rightarrow(a, b, c)) = \{\langle a, b, c \rangle\}$, $\mathcal{L}(\times(a, b, c)) = \{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$, $\mathcal{L}(\wedge(a, b, c)) = \{\langle a, b, c \rangle, \langle a, c, b \rangle, \langle b, a, c \rangle, \langle b, c, a \rangle, \langle c, a, b \rangle, \langle c, b, a \rangle\}$, $\mathcal{L}(\circlearrowleft(a, b)) = \{\langle a \rangle, \langle a, b, a \rangle, \langle a, b, a, b, a \rangle, \ldots\}$, $\mathcal{L}(\rightarrow(a, \times(b, c), \wedge(a, a))) = \{\langle a, b, a, a \rangle, \langle a, c, a, a \rangle\}$, $\mathcal{L}(\times(\tau, a, \tau, \rightarrow(\tau, b), \wedge(c, \tau))) = \{\langle \, \rangle, \langle a \rangle, \langle b \rangle, \langle c \rangle\}$, and $\mathcal{L}(\circlearrowleft(a, \tau, c)) = \{\langle a \rangle, \langle a, a \rangle, \langle a, a, a \rangle, \langle a, c, a \rangle, \langle a, a, c, a \rangle, \langle a, c, a, c, a \rangle, \ldots\}$.

Consider process tree $Q = \rightarrow(po, \wedge(\times(\tau, py), si), \times(co, \rightarrow(pd, \wedge(md, cp))))$, i.e., the process tree shown in Figure 2. $\mathcal{L}(Q) = \{\langle po, si, py, co \rangle, \langle po, si, py, pd, md, cp \rangle, \langle po, si, py, pd, cp, md \rangle, \langle po, py, si, co \rangle, \langle po, py, si, pd, md, cp \rangle, \langle po, py, si, pd, cp, md \rangle, \langle po, si, co \rangle, \langle po, si, pd, md, cp \rangle, \langle po, si, pd, cp, md \rangle\}$.

Given an event log $L \in \mathbb{B}(\Sigma^*)$ we would like to discover a process tree Q_L. For example, for event log $L = [\langle a,b,c,e \rangle^{85}, \langle a,c,b,e \rangle^{56}, \langle a,d,e \rangle^{34}]$ we would like to discover $Q = \to(a, \times(\wedge(b,c),d), e)$. For event log $L = [\langle a,b,d \rangle^{33}, \langle a,b,c,b,d \rangle^{25},$ $\langle a,b,c,b,c,b,d \rangle^{12}, \langle a,b,c,b,c,b,c,b,d \rangle^{6}, \langle a,b,c,b,c,b,c,b,c,b,d \rangle^{2}]$ we would like to discover $Q = \to(a, \circlearrowleft(b,c), d)$. The general idea of the inductive mining approach is to build a *directly-follows graph* and decompose the event log based on a particular *cut* of the directly-follows graph. The decomposition partitions the set of activities, i.e., sublogs are created in such a way that each event appears in precisely one of the sublogs. This is repeated until each sublog refers to only one activity. Note that each of the four operators, i.e., $\to$ (sequential composition), $\times$ (exclusive choice), $\wedge$ (parallel composition), and $\circlearrowleft$ (redo loops), corresponds to a particular type of cut and decomposes the event log accordingly.

To be able to define cuts, we first formalize the notion of a directly-follows graph.

Definition 4 (Directly-Follows Graph). *Let L be an event log, i.e., $L \in \mathbb{B}(\Sigma^*)$. The* directly-follows graph *of L is $G(L) = (A_L, \mapsto_L, A_L^{start}, A_L^{end})$ with:*

- $A_L = \{a \in \sigma \mid \sigma \in L\}$ *is the set of activities in L,*
- $\mapsto_L = \{(a,b) \in A \times A \mid a >_L b\}$ *is the directly-follows relation,*[5]
- $A_L^{start} = \{a \in A \mid \exists_{\sigma \in L} a = first(\sigma)\}$ *is the set of start activities, and*
- $A_L^{end} = \{a \in A \mid \exists_{\sigma \in L} a = last(\sigma)\}$ *is the set of end activities.*

The *Inductive Mining* (*IM*) algorithm iteratively splits the initial event log into smaller *sublogs*. For any sublog L we can create a directly-follows graph $G(L)$. $a \mapsto_L b$ if a was directly followed by b somewhere in L. $a \not\mapsto_L b$ if a was never directly followed by b. $\mapsto_L^+$ is the transitive closure of $\mapsto_L$. $a \mapsto_L^+ b$ if there is a non-empty *path* from a to b in $G(L)$, i.e., there exists a sequence of activities $a_1, a_2, \ldots, a_k$ such that $k \geq 2$, $a_1 = a$ and $a_k = b$ and $a_i \mapsto_L a_{i+1}$ for $i \in \{1, \ldots, k-1\}$. $a \not\mapsto_L^+ b$ if there is no path from a to b in the directly-follows graph.

Note that $a \mapsto_L b$ if a was directly followed by b only once in L. It is also possible to set thresholds to filter out infrequent behavior [16].

Definition 5 (Cut). *Let L be an event log with corresponding directly-follows graph $G(L) = (A_L, \mapsto_L, A_L^{start}, A_L^{end})$. Let $n \geq 1$. An n-ary cut of $G(L)$ is a partition of A_L into pairwise disjoint sets $A_1, A_2, \ldots A_n$: $A_L = \bigcup_{i \in \{1,\ldots,n\}} A_i$ and $A_i \cap A_j = \emptyset$ for $i \neq j$. Notation: $(\oplus, A_1, A_2, \ldots A_n)$ with $\oplus \in \{\to, \times, \wedge, \circlearrowleft\}$. For each type of operator ($\to$, $\times$, $\wedge$, and $\circlearrowleft$) specific conditions apply:*

- *An* exclusive-choice cut *of $G(L)$ is a cut $(\times, A_1, A_2, \ldots A_n)$ such that*
  - $\forall_{i,j \in \{1,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j} \ i \neq j \implies a \not\mapsto_L b$.
- *A* sequence cut *of $G(L)$ is a cut $(\to, A_1, A_2, \ldots A_n)$ such that*
  - $\forall_{i,j \in \{1,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j} \ i < j \implies (a \mapsto_L^+ b \ \wedge \ b \not\mapsto_L^+ a)$.
- *A* parallel cut *of $G(L)$ is a cut $(\wedge, A_1, A_2, \ldots A_n)$ such that*

---

[5] $a >_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \ldots t_n \rangle$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$.

- $\forall_{i\in\{1,...n\}} \; A_i \cap A_L^{start} \neq \emptyset \;\wedge\; A_i \cap A_L^{end} \neq \emptyset$ and
- $\forall_{i,j\in\{1,...n\}}\forall_{a\in A_i}\forall_{b\in A_j} \; i \neq j \implies a \mapsto_L b$.

– A redo-loop cut *of $G(L)$ is a cut $(\circlearrowleft, A_1, A_2, \dots A_n)$ such that*
- $n \geq 2$,
- $A_L^{start} \cup A_L^{end} \subseteq A_1$,
- $\{a \in A_1 \mid \exists_{i\in\{2,...n\}}\exists_{b\in A_i} \; a \mapsto_L b\} \subseteq A_L^{end}$,
- $\{a \in A_1 \mid \exists_{i\in\{2,...n\}}\exists_{b\in A_i} \; b \mapsto_L a\} \subseteq A_L^{start}$,
- $\forall_{i,j\in\{2,...n\}}\forall_{a\in A_i}\forall_{b\in A_j} \; i \neq j \implies a \not\mapsto_L b$,
- $\forall_{i\in\{2,...n\}}\forall_{b\in A_i}\exists_{a\in A_L^{end}} \; a \mapsto_L b \implies \forall_{a'\in A_L^{end}} \; a' \mapsto_L b$, and
- $\forall_{i\in\{2,...n\}}\forall_{b\in A_i}\exists_{a\in A_L^{start}} \; b \mapsto_L a \implies \forall_{a'\in A_L^{start}} \; b \mapsto_L a'$.

A cut $(\oplus, A_1, A_2, \dots A_n)$ with $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ of directly-follows graph $G(L)$ is maximal *if there is no cut $(\oplus, A_1', A_2', \dots A_m')$ with $m > n$. Cut $(\oplus, A_1, A_2, \dots A_n)$ is called* trivial *if $n = 1$.*

Definition 6 (Projection). *Let L be an event log and $(\oplus, A_1, A_2, \dots A_n)$ a cut with $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ based on the directly-follows graph $G(L)$ ($n \geq 2$). L is split into sublogs $L_1, L_2, \dots, L_n$ such that each event ends up in precisely one log and $\cup_{\sigma\in L_i}\{a \in \sigma\} = A_i$ for any $1 \leq i \leq n$.*

The precise way in which the event log is split depends on the operator [9,10,16]. Consider cut $(\rightarrow, \{a\}, \{b, c, d\}, \{e\})$ in the context of $L = [\langle a, b, c, e\rangle^{85}, \langle a, c, b, e\rangle^{56}, \langle a, d, e\rangle^{34}]$. L will be split into $L_1 = [\langle a\rangle^{175}]$, $L_2 = [\langle b, c\rangle^{85}, \langle c, b\rangle^{56}, \langle d\rangle^{34}]$, and $L_3 = [\langle e\rangle^{175}]$. Consider cut $(\times, \{a, b\}, \{c, d\})$ in the context of $L = [\langle a, b\rangle^{10}, \langle b, a\rangle^{10}, \langle c, d\rangle^{20}]$. L will be split into $L_1 = [\langle a, b\rangle^{10}, \langle b, a\rangle^{10}]$ and $L_2 = [\langle c, d\rangle^{20}]$. Consider cut $(\wedge, \{a, b\}, \{c\})$ in the context of $L = [\langle a, b, c\rangle^{10}, \langle b, a, c\rangle^{10}, \langle a, c, b\rangle^{10}, \langle b, c, a\rangle^{10}, \langle c, a, b\rangle^{10}, \langle c, b, a\rangle^{10}]$. L will be split into $L_1 = [\langle a, b\rangle^{30}, \langle b, a\rangle^{30}]$ and $L_2 = [\langle c\rangle^{60}]$. Consider cut $(\circlearrowleft, \{a, b\}, \{c, d\})$ in the context of $L = [\langle a, b\rangle^{10}, \langle a, b, c, d, a, b\rangle^{4}, \langle a, b, c, d, a, b, c, d, a, b\rangle^{2}]$. L will be split into $L_1 = [\langle a, b\rangle^{24}]$ and $L_2 = [\langle c, d\rangle^{8}]$. Note that each iteration creates a new case, e.g., case $\langle a, b, c, d, a, b, c, d, a, b\rangle$ is split into three $\langle a, b\rangle$ cases and two $\langle c, d\rangle$ cases.

The IM *algorithm works as follows* [1]. *IM* is a function that converts an event log into a process tree. Given a log or sublog L, $Q = IM(L)$ is the corresponding (sub)tree. Given an event log, the directly-follows graph is constructed. If there is a non-trivial exclusive-choice cut, then a maximal exclusive-choice cut is applied, splitting the event log into smaller event logs. If there is no non-trivial exclusive-choice cut, but there is a non-trivial sequence cut, then a maximal sequence cut is applied splitting the event log into smaller event logs. If there are no non-trivial exclusive-choice and sequence cuts, but there is a non-trivial parallel cut, then a maximal parallel cut is applied splitting the event log into smaller event logs. If there are no non-trivial exclusive-choice, sequence and parallel cuts, but there is a redo-loop cut, then a maximal redo-loop cut is applied splitting the event log into smaller event logs. After splitting the event log into sublogs the procedure is repeated until a *base case* (sublog with only one activity) is reached.

How the event log is split into sublogs, depends on the operator (see before). Empty traces are handled in a dedicated manner (based on the operator) and may result in the insertion of τ activities. If there are no non-trivial cuts meeting

the requirements in Definition 6, a *fall-through* is selected. The part that cannot be split is presented by a so-called *flower model* ("anything can happen"). Note that such a model can be easily represented as process tree $\circlearrowleft(\tau, a, b, \ldots)$ allowing for any trace involving the activities in the different redo parts. The fall-through serves as a last resort ensuring fitness, but possibly resulting in lower precision.

In the base case, the sublog contains only events corresponding to a particular activity, say a. If the sublog is of the form $L = [\langle a \rangle^k]$ with $k \geq 1$ (i.e., a occurs once in each trace), then the subtree a is returned. If the sublog is of the form $L = [\langle \ \rangle^k, \langle a \rangle^l]$ with $k, l \geq 1$, then the subtree $\times(a, \tau)$ is returned because a is sometimes skipped. If a is executed at least once in each trace in the sublog and sometimes multiple times (e.g., $L = [\langle a \rangle^9, \langle a, a \rangle^2, \langle a, a, a \rangle]$), then the subtree $\circlearrowleft(a, \tau)$ is returned. In all other cases (e.g., $L = [\langle \ \rangle^3, \langle a \rangle^4, \langle a, a, a \rangle]$), the subtree $\circlearrowleft(\tau, a)$ is returned because a is executed zero or more times in the traces of sublog L.

First, we show a larger worked out example showing the iterative process of splitting the event logs into sublogs based on cuts.

- Let $L_{abcdef} = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle]$ be an event log. Based on the directly-follows graph we identify a maximal sequence cut $(\rightarrow, \{a\}, \{b, c, e, f\}, \{d\})$ splitting the event log into L_a, L_{bcef}, L_d.
  - $L_a = [\langle a \rangle^{13}]$ is a base case. Hence, $IM(L_a) = a$.
  - $L_{bcef} = [\langle b, c \rangle^3, \langle c, b \rangle^4, \langle b, c, e, f, b, c \rangle^2, \langle c, b, e, f, b, c \rangle^2, \langle b, c, e, f, c, b \rangle, \langle c, b, e, f, b, c, e, f, c, b \rangle]$. There are no non-trivial exclusive-choice, sequence or parallel cuts. Therefore, we apply the maximal redo-loop cut $(\circlearrowleft, \{b, c\}, \{e, f\})$ splitting the event log into L_{bc} and L_{ef}.
    * $L_{bc} = [\langle b, c \rangle^{11}, \langle c, b \rangle^9]$. Based on the maximal parallel cut $(\wedge, \{b\}, \{c\})$, we obtain L_b and L_c.
      · $L_b = [\langle b \rangle^{20}]$ is a base case. Hence, $IM(L_b) = b$.
      · $L_c = [\langle c \rangle^{20}]$ is a base case. Hence, $IM(L_c) = c$.
    * $L_{ef} = [\langle e, f \rangle^7]$. Based on the maximal sequence cut $(\rightarrow, \{e\}, \{f\})$, we obtain L_e and L_f.
      · $L_e = [\langle e \rangle^7]$ is a base case. Hence, $IM(L_e) = e$.
      · $L_f = [\langle f \rangle^7]$ is a base case. Hence, $IM(L_f) = f$.
  - $L_d = [\langle d \rangle^{13}]$ is a base case. Hence, $IM(L_d) = d$.
- After splitting the event log to reach the base cases, we can construct the overall tree:
  - $IM(L_{b,c}) = \wedge(IM(L_b), IM(L_c)) = \wedge(b, c)$.
  - $IM(L_{e,f}) = \rightarrow(IM(L_e), IM(L_f)) = \rightarrow(e, f)$.
  - $IM(L_{bcef}) = \circlearrowleft(IM(L_{b,c}), IM(L_{e,f})) = \circlearrowleft(\wedge(b, c), \rightarrow(e, f))$.
  - $IM(L_{abcdef}) = \rightarrow(IM(L_a), IM(L_{bcef}), IM(L_d)) = \rightarrow(a, \circlearrowleft(\wedge(b, c), \rightarrow(e, f)), d)$ represents the overall process tree for the whole event log.

To illustrate the handling of base cases we show a few smaller examples:

- If $L = [\langle a, a \rangle^{12}, \langle a, a, a \rangle^6]$, then $IM(L) = \circlearrowleft(a, \tau)$.

– If $L = [\langle a, b, c \rangle^{20}, \langle a, c \rangle^{30}]$, then $IM(L) = \rightarrow(a, \times(b, \tau), c)$.
– If $L = [\langle b \rangle^{12}, \langle a, b \rangle^6, \langle b, c \rangle^5, \langle a, b, c \rangle^4]$, then $IM(L) = \rightarrow(\times(a, \tau), b, \times(c, \tau))$.
– If $L = [\langle a, c \rangle^2, \langle a, b, c \rangle^3, \langle a, b, b, c \rangle^2, \langle a, b, b, b, c \rangle^2, \langle a, b, b, b, b, b, b, c \rangle]$, then $IM(L) = \rightarrow(a, \circlearrowleft(\tau, b), c)$.

In this section, we only described the basic *IM* algorithm [1, 10]. This algorithm provides many guarantees. *The discovered models are always sound and are guaranteed to be able to replay the whole event log.* Moreover, for large subclasses of process trees, rediscoverability is guaranteed (i.e., a sufficiently large event log obtained by simulating a model is sufficient to reconstruct a behaviorally equivalent model). However, the basic algorithm presented here cannot abstract from infrequent behavior and does not handle incompleteness well. The log is assumed to be directly-follows complete and frequencies are not taken into account. Fortunately, the inductive mining framework is quite flexible. Using the basic ideas presented in this section, a family of inductive mining techniques has been developed [1, 9, 10, 16, 17]. All use a *divide-and-conquer* approach in combination with process trees that are *sound by construction*. As demonstrated in [9, 17] the approach can be made highly scalable and can be applied to huge event logs.

3.3 Process Discovery Using Region-Based Approaches

In contrast to inductive mining, which is able to guarantee a sound workflow model, the existing approaches that rely on the notion of *region theory* [18] search for a process model that is both fitting and precise [19]. This section shows two branches of region-based approaches for process discovery: state and language-based approaches.

State-based Region Approach for Process Discovery State-based region approaches for process discovery need to convert the event log into a state-based representation, that will be used to discover the Petri net. The techniques described in [20] present many variants for solving this first step. The basic idea to incorporate state information is to look at the pre/post history of a subtrace in the event log. Figures 4(a)-(b) show an example, where states are decided by looking at the set of common prefixes.

A *transition system* (TS) is a tuple (S, Σ, A, s_{in}), where S is a set of *states*, Σ is an alphabet of *activities*, $A \subseteq S \times \Sigma \times S$ is a set of *(labeled) arcs*, and $s_{in} \in S$ is the *initial state*. We will use $s \xrightarrow{e} s'$ as a shortcut for $(s, e, s') \in A$, and the transitive closure of this relation will be denoted by $\xrightarrow{*}$. Figure 4(b) presents an example of a transition system.

A *region*[6] in a transition system is a set of states that satisfy an homogeneous relation with respect to the set of arcs. In the simplest case, this relation can

---

[6] In this paper we will use region to denote a 1-bounded region. However, when needed we will use k-bounded region to extend the notion, necessary to account for k-bounded Petri nets.
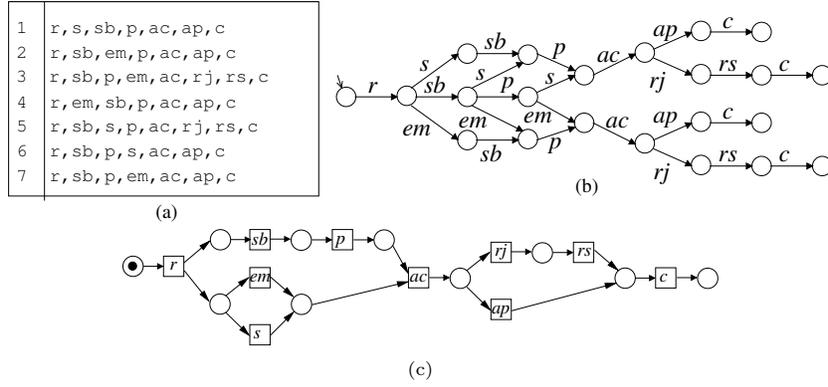
1	`r,s,sb,p,ac,ap,c`
2	`r,sb,em,p,ac,ap,c`
3	`r,sb,p,em,ac,rj,rs,c`
4	`r,em,sb,p,ac,ap,c`
5	`r,sb,s,p,ac,rj,rs,c`
6	`r,sb,p,s,ac,ap,c`
7	`r,sb,p,em,ac,ap,c`

(a)

(b)

(c)

Fig. 4. State-based region discovery: (a) log L, (b) a transition system corresponding to L, (c) derived Petri net.
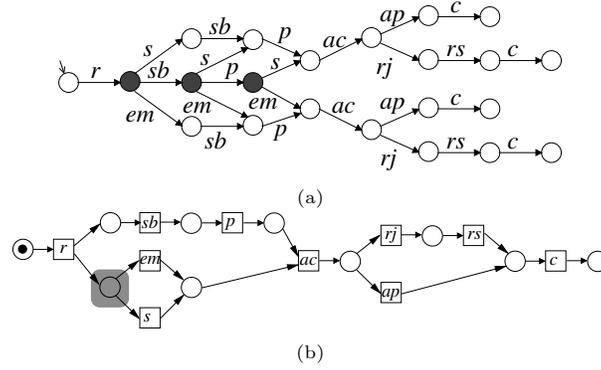

(a)

(b)

Fig. 5. (a) Example of region (three shadowed states). The predicates are r enters, s and em exits, and the rest of events do not cross, (b) Corresponding place shadowed in the Petri net.

be described by a predicate on the set of states considered. Formally, let S' be a subset of the states of a TS, $S' \subseteq S$. If $s \notin S'$ and $s' \in S'$, then we say that transition $s \xrightarrow{a} s'$ *enters* S'. If $s \in S'$ and $s' \notin S'$, then transition $s \xrightarrow{a} s'$ *exits* S'. Otherwise, transition $s \xrightarrow{a} s'$ *does not cross* S': it is completely inside ($s \in S'$ and $s' \in S'$) or completely outside ($s \notin S'$ and $s' \notin S'$). A set of states $r \subseteq S$ is a region if for each event $e \in E$, exactly one of the three predicates (*enters*, *exits* or *does not cross*) holds for each of its arcs. An example of region is presented in Figure 5 on the TS of our running example. In the highlighted region, r enters the region, s and em exit the region, and the rest of labels do not cross the region.

A region corresponds to a place in the Petri net, and the role of the arcs determine the Petri net flow relation: when an event e enters the region, there is an arc from the corresponding transition for e to the place, and when e exits the

region, there is an arc from the region to the transition for e. Events satisfying the do not cross relation are not connected to the corresponding place. For instance, the region shown in Figure 5(a) corresponds to the shadowed place in Figure 5(b), where event r belongs to the set of input transitions of the place whereas events em and s belong to the set of output transitions. Hence, the algorithm for Petri net derivation from a transition system consists in finding regions and constructing the Petri net as illustrated with the previous example. In [21] it was shown that only a minimal set of regions was necessary, whereas further relaxations to this restriction can be found in [19]. The Petri net obtained by this method is guaranteed to accept the language of the transition system, and satisfy the *minimal language containment property*, which implies that if all the minimal regions are used, the Petri net derived is the one whose language difference with respect to the log is minimal, hence being the most precise Petri net for the set of transitions considered.

In any case, the algorithm that searches for regions in a transition system must explore the lattice of sets (or multisets, in the case for *k-bounded* regions), thus having a high complexity: for a transition system with n states, the lattice for k-bounded regions is of size $\mathcal{O}(k^n)$. For instance, the lattice of sets of states for the toy TS used in this article (which has 22 states) has 2^{22} possibles sets to check for the region conditions. Although many simplification properties, efficient data structures and algorithms, and heuristics are used to prune this search space [19], they only help to alleviate the problem. Decomposition alternatives, which for instance use partitions of the state space to guide the search for regions, significantly alleviate the complexity of the state-based region algorithm, at the expense of not guaranteeing the derivation of precise models [22]. Other state-based region approaches for discovery have been proposed, which complement the approach described in this section [23–25].

Language-based Region Approach for Process Discovery In Language-based region theory [26–31] the goal is to construct the smallest Petri net such that the beharviour of the net is equal to the given input language (or minimally larger). [32] provides an overview for language-based region theory for different classes of languages: step languages, regular languages, and (infinite) partial languages.

More formally, let $L \in \mathbb{B}(\Sigma^*)$ be an event log, then language based region theory constructs a Petri net with the set of transitions equals to Σ and in which all traces of L are a firing sequence. The Petri net should have only minimal firing sequences not in the language L (and all prefixes in L). This is achieved by adding places to the Petri net that restrict unobserved behavior, while allowing for observed behavior. The theory of regions provides a method to identify these places, using *language regions*.

Definition 7 (Prefix Closure). *Let $L \in \mathbb{B}(\Sigma^*)$ be an event log. The prefix closed language $\mathcal{L} \subseteq \Sigma^*$ of L is defined as: $\mathcal{L} = \{\sigma \in \Sigma^* \mid \exists_{\sigma' \in \Sigma^*} \sigma \circ \sigma' \in L\}$.*

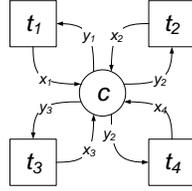The prefix closure of a log is simply the set of all prefixes in the log (including the empty prefix).

Fig. 6. Region for a language over four activities [33].

Definition 8 (Language Region). *Let Σ be a set of activities. A region of a prefix-closed language $\mathcal{L} \in \Sigma^*$ is a triple $(\boldsymbol{x}, \boldsymbol{y}, c)$ with $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^{\Sigma}$ and $c \in \{0,1\}$, such that for each non-empty sequence $w = w' \circ a \in \mathcal{L}$, $w' \in \mathcal{L}$, $a \in \Sigma$:*

$$c + \sum_{t \in \Sigma} \left(\boldsymbol{w}'(t) \cdot \boldsymbol{x}(t) - \boldsymbol{w}(t) \cdot \boldsymbol{y}(t) \right) \geq 0$$

This can be rewritten into the inequation system:

$$c \cdot \mathbf{1} + M' \cdot \boldsymbol{x} - M \cdot \boldsymbol{y} \geq \mathbf{0}$$

where M and M' are two $|\mathcal{L}| \times |\Sigma|$ matrices with $M(w,t) = \boldsymbol{w}(t)$, and $M'(w,t) = \boldsymbol{w}'(t)$, with $w = w' \circ a$. The set of all regions of a language is denoted by $\Re(\mathcal{L})$ and the region $(\mathbf{0}, \mathbf{0}, 0)$ is called the trivial region.

Intuitively, vectors $\boldsymbol{x}, \boldsymbol{y}$ denote the set of incoming and outgoing arcs of the place corresponding to the region, respectively, and c sets if it is initially marked. Figure 6 shows a region for a language over four activities, i.e. each solution $(\boldsymbol{x}, \boldsymbol{y}, c)$ of the inequation system can be regarded in the context of a Petri net, where the region corresponds to a feasible place with preset $\{t | t \in T, \boldsymbol{x}(t) = 1\}$ and postset $\{t | t \in T, \boldsymbol{y}(t) = 1\}$, and initially marked with c tokens. Note that we do not assume arc-weights here, while the authors of [26–28, 34] do.

Since the place represented by a region is a place which can be added to a Petri net, without disturbing the fact that the net can reproduce the language under consideration, such a place is called a *feasible* place.

Definition 9 (Feasible place). *Let $\mathcal{L}$ be a prefix-closed language over Σ and let $N = ((P, \Sigma, F), m)$ be a marked Petri net. A place $p \in P$ is called* feasible *if and only if there exists a corresponding* region $(\boldsymbol{x}, \boldsymbol{y}, c) \in \Re(\mathcal{L})$ *such that $m(p) = c$, and $\boldsymbol{x}(t) = 1$ if and only if $t \in {}^{\bullet}p$, and $\boldsymbol{y}(t) = 1$ if and only if $t \in p^{\bullet}$.*

In general, there are many feasible places for any given event log (when considering arc-weights in the discovered Petri net, there are even infinitely many). Several methods exist for selecting an appropriate subset of these places. The authors of [28,34] present two ways of finitely representing these places, namely a *basis representation* and a *separating representation*. Both representations maximize precision, i.e. they select a set of places such that the behavior of the model outside of the log is minimal.

In contrast, the authors of [33, 35–37] focus on those feasible places that express some causal dependency observed in the event log, and/or ensure that the entire model is a connected workflow net. They do so by introducing various cost functions favouring one solution of the equation system over another and then selecting the top candidates.

Process Discovery vs. Region Theory The goal of region theory is to find a Petri net that perfectly describes the observed behavior (where this behavior is specified in terms of a language or a statespace). As a result the Petri nets are perfectly fitting and maximally precise.

As a consequence, the assumption on the input is that it provides a *full behavioral specification*, i.e. that the input is *complete* and *noise free*. Furthermore, the assumption on the output is that it is a *compact, exact representation* of the input behavior.

When applying region theory in the context of process mining, it is therefore very important to perform any generalization *before* calling region theory algorithms. For state-based regions, the challenges are in the construction of the statespace from the event log and in language based regions in the selection of the appropriate prefixes to include in the final prefix-closed language in order to ensure some level of generalization.

4 Conformance Checking and the Challenge of Alignments

Conformance checking is a crucial dimension in process mining: by relating modelled and observed behavior, process models that have either been discovered or manually created, can be confronted with event data [2]. On its core, conformance checking relies on the fundamental problem of identifying, among the set of runs of a process model (which can be infinite), the run that mostly resembles an observed trace. In this section we overview the problem of computing alignments, and provide applications to be build on top of alignments.

4.1 Formal Definition of Alignments

An *alignment* of an observed trace and a process model relates events of the observed trace to elements of the model and vice versa. Such an alignment reveals how the given trace can be replayed on the process model. The classical notion of aligning an event log and process model was introduced by [38]. To achieve an alignment, we need to relate *moves* in the observed trace to *moves* in the model. It may be the case that some of the moves in the observed trace can not be mimicked by the model and vice versa. For instance, consider the model N_1 in Figure 7, with the following labels, $\lambda(t_1) = a_1, \lambda(t_2) = a_2, \lambda(t_3) = a_3$ and $\lambda(t_4) = a_4$, and the trace $\sigma = \langle a_1, a_1, a_4, a_2 \rangle$; four possible alignments are:

$$\alpha_1 = \begin{array}{|c|c|c|c|c|} \hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline t_1 & \bot & t_3 & t_4 & \bot \\ \hline \end{array} \quad \alpha_2 = \begin{array}{|c|c|c|c|c|} \hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline \bot & t_1 & t_2 & t_4 & \bot \\ \hline \end{array}$$
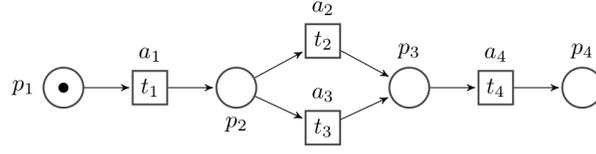
Fig. 7. Process model N_1.

$$\alpha_3 = \begin{array}{|c|c|c|c|c|} a_1 & a_1 & a_4 & a_2 & \bot \\ \hline t_1 & \bot & \bot & t_2 & t_4 \end{array} \quad \alpha_4 = \begin{array}{|c|c|c|c|c|} a_1 & a_1 & a_4 & a_2 & \bot \\ \hline \bot & t_1 & \bot & t_2 & t_4 \end{array}$$

The moves are represented in tabular form, where moves by the trace are at the top, and moves by the model are at the bottom of the table. For example the first move in α_2 is $(a_1, \bot)$ and it means that the observed trace moves a_1, while the model does not make any move. Formally, an alignment is defined as follows:

Definition 10 (Alignment). *Given a labeled Petri net N and an alphabet of events Σ, Let A_M and A_L be the alphabet of transitions in the model and events in the log, respectively, and $\bot$ denote the empty set, then:*

- *(X, Y) is a synchronous move if $X \in A_L$, $Y \in A_M$ and $X = \lambda(Y)$*
- *(X, Y) is a move in log if $X \in A_L$ and $Y = \bot$.*
- *(X, Y) is a move in model if $X = \bot$ and $Y \in A_M$.*
- *(X, Y) is an illegal move, otherwise.*

The set of all legal moves is denoted as A_{LM} and given an alignment $\alpha \in A_{LM}^*$, the projection of the first element (ignoring $\bot$), $\alpha \uparrow_{A_L}$, results in the observed trace σ, and projecting the second element (ignoring $\bot$), $\alpha \uparrow_{A_M}$, results in the model trace.

For the previous example, $\alpha_1 \uparrow_{A_M} = t_1 t_3 t_4$ and $\alpha_1 \uparrow_{A_L} = a_1 a_1 a_4 a_2$.

Costs can be associated to the different types of moves in Def. 10. Traditionally, the approaches in the literature use a cost function that assigns higher costs to asynchronous moves (move in model/log) than to synchronous moves, and the model trace that minimizes the cost (hence, minimizing the number of asynchronous moves) is computed. When a cost function is in place, then one can consider optimality: an *optimal alignment* is an alignment with minimal cost. The most simple cost function that satisfies this requirement is the *standard cost function*, which assigns cost 1 to asynchronous moves, and cost 0 to synchronous moves. In this paper we will assume the standard cost function, which will for instance assign cost 3 to the alignments $\alpha_1 - \alpha_4$ shown before. According to the standard cost function, all four alignments are optimal.

4.2 Techniques for the Computation of Alignments

In this section we report some of the existing alternatives to compute alignments. First we describe the reference technique nowadays for alignment computation.
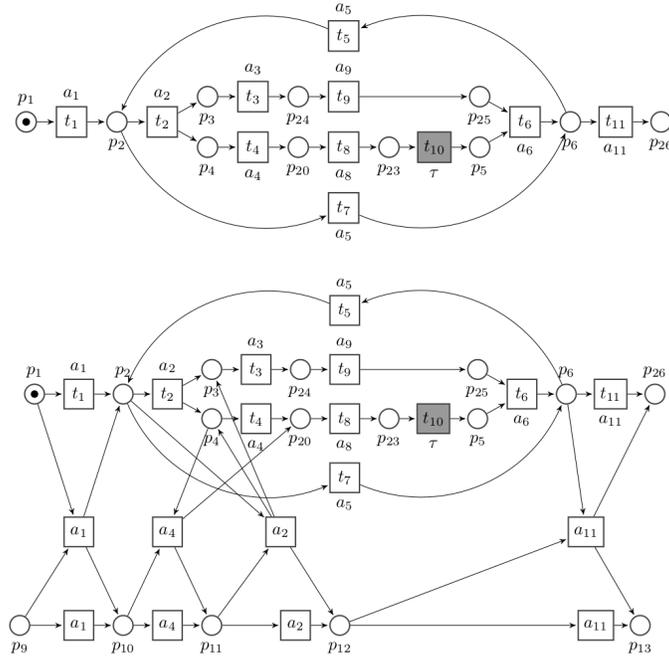
Fig. 8. (Top) Process model, (Bottom) Synchronous product net over the trace
$\langle a_1, a_4, a_2, a_{11} \rangle$.

Then, we provide pointers to other techniques so that the reader gets an overall impression on the vivid field of alignment computation.

A* Technique over the Synchronous Product Net The reference technique for alignment computation was presented in the context of Arya Adriansyah's PhD thesis [38]. It is based on the notion of *synchronous product net* (SPN, for short), which we define informally now with the help of an example.

Figure 8(Top) describes a process model. Now let us assume that it should be aligned with the trace $\sigma = \langle a_1, a_4, a_2, a_{11} \rangle$. The idea underlying the method from [38] is to first create an SPN, that encompasses the joint behavior between the process model and σ. This SPN is described in Figure 8(Bottom). Transitions in the SPN can be partitioned into three sets

- Transitions at the top (i.e., $t_1, \ldots t_{11}$) correspond to original model transitions.
- Transitions at the bottom correspond to the Petri net representation of σ.
- Transitions in the middle represent the joint synchronization of the model and the trace states.

Accordingly, transitions in the SPN will be assigned a cost[7]: transitions in the top of the SPN, and in the bottom of the SPN, will receive cost one. Transitions in the middle, will receive cost 0. Informally, this cost assignment penalizes the process model or the log independent executions, and in contrast favours synchronous executions, that are executed without cost. Then, a search for the cost-minimal path between initial and final marking of the state space of the created SPN is computed.

The cost-minimal path search can be done either once the full state-space is computed, or more intelligently by applying a A^* strategy to avoid, whenever possible, the full exploration of the state-space. This can be achieved by using heuristics that at each state reached, estimate the minimal cost to reach the final marking, and prune the exploration for successor states that are not promising. Several heuristics can be applied, that include the use of the *marking equation of Petri nets* [12] to estimate the remaining distance by solving an (integer) linear program that provides a lower bound to the real distance.

Other Techniques Alternatives to A^* have appeared very recently: in the approach presented in [39], the alignment problem is mapped as an *automated planning* instance. Unlike the A^*, the aforementioned work is only able to produce one optimal alignment (not all optimal), but it is expected to consume considerably less memory. Automata-based techniques have also appeared [40, 41]. In particular, the technique in [40] can compute all optimal alignments. The technique in [40] relies on state space exploration and determinization of automata, whilst the technique in [41] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [42] presented the notion of *approximate* alignment to alleviate the computational demands of the current challenge by proposing a recursive paradigm on the basis of structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. A follow-up work of [42] is presented in [43], which proposes a trade-off between complexity and optimality of solutions, and guarantees executable properties of results. The technique in [44] presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of alignments. The obtained alignment, called *macro-alignment* since some of the positions are high-level elements, is expanded based on the information gathered during the initial reduction. Decompositional techniques have been presented [45, 46] that, instead of computing optimal alignments, they focus on the *decisional problem* of whereas a given trace fits or not a process model.

Recently, two different approaches by the same authors have appeared: the work in [47] proposes using binary decision diagrams to alleviate the computation of alignments. The work in [48], which has the goal of maximizing the synchronous moves of the computed alignments, uses a pre-processing step on the model.

---

[7] Remember that we are assuming the standard cost function that assigns cost 1 to synchronous moves and cost 0 to asynchronous moves.

4.3 Alignments Applications

In this section we overview different use cases of alignments. The reader can find a complete and detailed presentation in [2].

Model Enhancement Alignments open the door to incorporate information from the event log to the process model. More concretely, from an alignment α, one can transfer the information from the log trace $\alpha \uparrow_{A_L}$ to the model activities in $\alpha \uparrow_{A_M}$ corresponding to the synchronous moves. This information is contained in the events attributes. For instance, after aligning the complete log, one can realize that in reality a given model transition is only performed by a limited set of persons or roles, or the cost of executing it is within certain margins.

Alignments can also be used to *animate* the process model. This has already illustrated early in the paper (see for instance the annotations in Figure 1 (a)). This helps to understand better the visualization of real traces through the process model, and is one of the most interesting features of several existing process mining tools.

Furthermore, more elaborated information can be projected on top of the process model, with the help of a tailored analysis. One possibility is *performance analysis*, in order to display performance information like *activity durations*, *waiting times* and *routing probabilities*. For instance, in the left-most decision of Figure 1 (a), one can see that 11.1% (141 out of 1266) of the cases the activity *pay* is skipped.

Alternatively, the event log attributes can be used to explain decision in process models, once alignments are obtained. This is known as *decision point analysis*. Decision point analysis is based on building prediction models next to each decision point, using the data available. Often, the data used to feed these models can be obtained from the events corresponding to the prefixes of traces that lead to the decision point. Examples of data attributes that can be used for the previous example are product, resource, prod-price, quantity and address in Table 1. Continuing with the previous example, one can for instance infer that the explanation on why the activity *pay* is sometimes skipped is due to products orders placed which both contain more than one item and the item price is less than 500.

Model and Log Repair Another interesting application of using alignments is the possibility to repair the model or the log, so that they are more aligned. Model repair can be performed by selecting and resolving particular asynchronous moves in an alignment [49]. Intuitively, to resolve model moves one need to insert routing logic that allows to skip certain activities to be executed. Symmetrically, to resolve log moves one should extend the process model so that new behavior is possible in particular situations.

On the other hand, log repair considers that models contain the truth and whenever deviations are found, they should be corrected at the log level. Correcting deviations at log level is straightforward: model moves amounts to insert events in the observed trace, whilst log moves imply to remove events in

the observed trace. For instance, the log repair of the alignment α_1 for trace $\sigma = \langle a_1, a_1, a_4, a_2 \rangle$, and the model shown in Figure 7, derives the new trace $\sigma' = \langle a_1, a_3, a_4 \rangle$.

5 Evidence-based Quality Metrics for Process Models

With the aim of quantifying the relation between observed and modeled behavior, conformance checking techniques consider four quality dimensions: fitness, precision, generalization and simplicity [50]. In section 3.1 we already described the intuition behind these dimensions. In this section we do a step further, and present ways to measure them.

When alignments are available, most of the quality dimensions can be defined on top [2]. In a way, alignments are optimistic: although observed behavior may deviate significantly from modeled behavior, it is always assumed that the least deviations are the best explanation (from the model's perspective) for the observed behavior. For the first three dimensions, the alignment between a process model and an event log is of paramount importance, since it allows relating modelled and observed behavior.

5.1 Fitness

Fitness evaluates to which extent the observed behavior is possible according to the modelled behavior. Intuitively (and abusing a bit the notation), if L is also considered the language of the log, then the fitness of L with respect to a process model N can be computed by the following formula:

$$fitness = \frac{|L \cap \mathcal{L}(N)|}{|L|}$$

Alternatively, a refined metric for fitness can be measured through alignments. The way alignments are constructed, i.e. by looking for a shortest path through the state space of the SPN, is not necessarily deterministic. There may be more than one shortest path. However, the final cost of the alignment is minimal and therefore deterministic. On the basis of this cost, alignment-based fitness is defined as:

$$fitness_a = 1 - \frac{\text{cost of the optimal alignment}}{\text{cost of worst-case alignment}}$$

Again, for any log L we have:

$$fitness_a^l = 1 - \frac{\sum_{\sigma \in L} \text{cost of the optimal alignment for } \sigma}{\sum_{\sigma \in L} \text{cost of worst-case alignment for } \sigma}$$

For alignment-based fitness, two costs are of interest, namely the cost of the optimal alignment and the cost of the worst-case alignment. The former is obtained by the alignment algorithm defined in section 4.2. The latter is simply defined as the cost of aligning the empty trace in the model plus the cost of treating all events as log moves.

5.2 Precision

One important metric in conformance checking is to assess the precision of the model with respect to the observed executions, i.e., characterize the ability of the model to produce behavior unrelated to the one observed. Therefore, precision can be measured by the following formula:

$$precision = \frac{|L \cap \mathcal{L}(N)|}{|\mathcal{L}(N)|}$$

The formula above poses a problem for measuring precision in practice: since in general a model can have an infinite language, the formula on its limit tends to 0. Below we provide a couple of alternatives to fight this problem, thus coining metrics for estimating the precision dimension.

Escaping Arcs Precision can be approximated by exploring the behavior of the process model using as a reference the traces of the log, and stopping the exploration each time modelled behavior deviates from recorded behavior [51]. The following formula would then be used to estimate precision:

$$precision_{ea}(L, N) = \frac{\sum_{\sigma \in L, e \in \sigma} enabled_L(e)}{\sum_{\sigma \in L, e \in \sigma} enabled_N(e)}$$

where $enabled_L(e)$ provides the activities that are possible in L after the same prefix that contains the event e has been observed, and $enabled_N(e)$ denotes the number of tasks that can be executed in the state right before executing the task corresponding to e. In the formula above, we assume fitting models, i.e., $enabled_L(e) \subseteq enabled_N(e)$. This assumption can be lifted with the help of the alignments [52].

Anti-Alignments. The idea of *anti-alignments* [53] is to seek in the language of a model N, what are the runs which differ considerably with all the observed traces. Hence, this is the opposite of the notion of alignment. Anti-alignments can be used to measure precision [53, 54]. The intuition behind the metric based on anti-alignments is as follows. A very precise process model allows for exactly the traces to be executed and not more. Hence, if one trace σ is removed from the log, this trace becomes the anti-alignment for the remaining log, as it is the only execution of the model that is not in the log. This idea would lead to a *trace-based* precision metric grounded in anti-alignments, that penalizes the model precision for traces σ in the log that are very deviating from the anti-alignment obtained when removing σ in the log. Alternatively, a *log-based* precision metric can also be defined, if not per trace but instead a single anti-alignment with respect to the whole log is used.

5.3 Generalization

In process mining, the challenge is not to discover *the correct* process model for a given log, but to discover the process model that provides the most insights into the process from which the log originated. This is comparable to the area of process modelling, where a model is made of a process in order to describe that process in a meaningful way [55].

In process modelling, there are essentially two fundamental concepts that describe a lot of behavior using only a limited number of modelling elements, namely parallelism and loops. Both constructs have in common that they allow for many traces to be executed, while at the same time the number of states a process model can be in remains limited. In order to generalize from a set of observed sequences, process mining techniques make use of the inference of concurrency and loops. All process discovery techniques presented in Section 3 do so, however the way they decide is different.

Another fundamental property of correctly modelling processes is *abstraction*, i.e. one should only include parts of a model if they are relevant enough. By making models too detailed, an end-user can no longer see the important parts of the process.

Generalization is a quality dimension that tries to measure whether the inference of parallelism and loops and abstraction are done properly. For generalization, only few metrics exist [14, 53, 56]. The approach in [14] considers frequency of use, where models are assumed to generalize if all parts of the model are used equally frequently when reproducing the event log, i.e. this metric focusses entirely on proper abstraction of the observed data. The metric in [56] uses artificial negative events, i.e. events that were not observed at a particular point in the trace and uses a confidence in these events to measure generalization. Finally, in [53] a metric based on anti-alignments is presented which focusses entirely on the relation between the number of states and the number of traces in the model.

5.4 Simplicity

When a process model is discovered, one crucial metric for its evaluation is simplicity: is the derived process model the most simple explanation of the underlying process? This metric refers to the *Occam's Razor* principle. One method for measuring simplicity is by analysing the complexity of the underlying graph. In [57] some examples of such complexity metrics (e.g., *size*, *diameter*, *connectivity*) can be found. Alternatively, process model related metrics can also be defined, such as *sequentiality*, *structuredness*, among others.

6 Concluding Remarks

Process mining is a discipline which already impacts organizations in the present, but promises to impact them even more in the near future. In this paper we

have overviewed the area, focusing on the fundamental algorihtmic challenges that need to be confronted, to make these promises to become realities soon. Behind these challenges lie traditional theory that has been there several decades already: Petri net theory, data science methods, optimization, business process management, to cite the most important ones. We believe research in these areas would contribute to the development of the process mining field as well, a phenomena that we have already observed in the last decade.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
3. Pearl, J.: Reasoning under uncertainty. Annual Review of Computer Science **4**(1) (1990) 37–72
4. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series. Springer (2015)
5. Kerremans, M.: Gartner Market Guide for Process Mining, Research Note G00353970. `www.gartner.com` (2018)
6. Koplowitz, R., Mines, C., Vizgaitis, A., Reese, A.: Process Mining: Your Compass For Digital Transformation: The Customer Journey Is The Destination. `www.forrester.com` (2019)
7. TFPM: Process Mining Case Studies, `http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies` (2017)
8. Celonis: Process Mining Success Story: Innovation is an Alliance with the Future, `http://www.win.tue.nl/ieeetfpm/lib/exe/fetch.php?media=:casestudies:siemens_celonis_story_english.pdf` (2017)
9. Leemans, S., Fahland, D., van der Aalst, W.: Scalable process discovery and conformance checking. Software and Systems Modeling (2016) 1–33
10. Leemans, S., Fahland, D., Aalst, W.: Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In Colom, J., Desel, J., eds.: Applications and Theory of Petri Nets 2013. Volume 7927. (2013) 311–329
11. Aalst, W.: Discovering the "Glue" Connecting Activities - Exploiting Monotonicity to Learn Places Faster. In Boer, F., Bonsangue, M., Rutten, J., eds.: It's All About Coordination. (2018) 1—-20
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (April 1989) 541–574
13. Aalst, W.: Relating Process Models and Event Logs: 21 Conformance Propositions. In: Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data (ATAED 2018). Volume 2115 of CEUR Workshop Proceedings., CEUR-WS.org (2018) 56–74
14. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. Int. J. Cooperative Inf. Syst. **23**(1) (2014)

15. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Asp. Comput. **23**(3) (2011) 333–363

16. Leemans, S., Fahland, D., Aalst, W.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In Lohmann, N., Song, M., Wohed, P., eds.: Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013). Volume 171. (2014) 66–78

17. Leemans, S.: Robust Process Mining With Guarantees. Phd thesis, Eindhoven University of Technology (2017)

18. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica **27** (1990) 315–368

19. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Transactions on Computers **59**(3) (2009) 371–384

20. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W.E., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (2009)

21. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4) (1996) 297–315

22. Carmona, J.: Projection approaches to process mining using region-based techniques. Data Min. Knowl. Discov. **24**(1) (2012) 218–246

23. Solé, M., Carmona, J.: Light region-based techniques for process discovery. Fundam. Inform. **113**(3-4) (2011) 343–376

24. Solé, M., Carmona, J.: Incremental process discovery. Trans. Petri Nets and Other Models of Concurrency **5** (2012) 221–242

25. Solé, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1) (2013) 192–205

26. Darondeau, P.: Deriving Unbounded Petri Nets from Formal Languages. In: CONCUR 1998, London, UK, Springer-Verlag (1998) 533–548

27. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial Algorithms for the Synthesis of Bounded Nets. In: TAPSOFT. (1995) 364–378

28. Lorenz, R., Juhás, R.: How to Synthesize Nets from Languages - a Survey. In: Proceedings of the Wintersimulation Conference (WSC) 2007. (2007)

29. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of petri nets from infinite partial languages. In: ACSD. (2008) 170–179

30. Lorenz, R.: Towards synthesis of petri nets from general partial languages. In: AWPN. (2008) 55–62

31. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of petri nets from term based representations of infinite partial languages. Fundam. Inform. **95**(1) (2009) 187–217

32. Mauser, S., Lorenz, R.: Variants of the language based synthesis problem for petri nets. In: ACSD. (2009) 89–98

33. van der Aalst, W.M.P., van Dongen, B.F.: Discovering petri nets from event logs. Trans. Petri Nets and Other Models of Concurrency **7** (2013) 372–422

34. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: BPM 2007. Volume 4714 of LNCS., Springer (2007) 375–383

35. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inform. **94**(3-4) (2009) 387–412

36. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5) (2018) 529–556

37. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Ilp-based process discovery using hybrid regions. In van der Aalst, W.M.P., Bergenthum, R., Carmona, J., eds.: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015. Volume 1371 of CEUR Workshop Proceedings., CEUR-WS.org (2015) 47–61

38. Adriansyah, A.: Aligning observed and modeled behavior. PhD thesis, Technische Universiteit Eindhoven (2014)

39. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Expert Syst. Appl. **82** (2017) 162–183

40. Reißner, D., Conforti, R., Dumas, M., Rosa, M.L., Armas-Cervantes, A.: Scalable conformance checking of business processes. In: OTM CoopIS, , Rhodes, Greece. (2017) 607–627

41. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Software and System Modeling **17**(2) (2018) 599–631

42. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: 14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, September 18 - 22. (2016)

43. van Dongen, B., Carmona, J., Chatain, Th., Taymouri, F.: Aligning modeled and observed behavior: A compromise between complexity and quality. In Dubois, E., Pohl, K., eds.: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE'17). Volume 10253 of Lecture Notes in Computer Science., Essen, Germany, Springer (June 2017) To appear.

44. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016), Graz, Austria, December 15-16, 2016. (2016) 50–62

45. Munoz-Gama, J., Carmona, J., Van Der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46** (December 2014) 102–122

46. van der Aalst, W.M.P.: Decomposing petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4) (2013) 471–507

47. Bloemen, V., van de Pol, J., van der Aalst, W.M.P.: Symbolically aligning observed and modelled behaviour. In: 18th International Conference on Application of Concurrency to System Design, ACSD, Bratislava, Slovakia, June 25-29. (2018) 50–59

48. Bloemen, V., van Zelst, S.J., van der Aalst, W.M.P., van Dongen, B.F., van de Pol, J.: Maximizing synchronization for aligning observed and modelled behaviour. In: Business Process Management - 16th International Conference, BPM, Sydney, NSW, Australia. (2018) 233–249

49. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47** (2015) 220–243

50. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1) (2008) 64–95

51. Munoz-Gama, J.: Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. Volume 270 of Lecture Notes in Business Information Processing. Springer (2016)

52. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E-Business Management **13**(1) (2015) 37–67

53. Chatain, T., Carmona, J.: Anti-alignments in conformance checking - the dark side of process models. In: Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings. (2016) 240–258

54. van Dongen, B.F., Carmona, J., Chatain, T.: A unified approach for measuring precision and generalization based on anti-alignments. In: Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings. (2016) 39–56

55. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)

56. vanden Broucke, S.K.L.M., Weerdt, J.D., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. IEEE Trans. Knowl. Data Eng. **26**(8) (2014) 1877–1889

57. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the occurrence of errors in process models based on metrics. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25–30, 2007, Proceedings, Part I. (2007) 113–130