Process Discovery from Event Data: Relating Models and Logs Through Abstractions

Wil M.P. van der Aalst*

Article Type
Overview Article

Abstract

Event data are collected in logistics, manufacturing, finance, healthcare, customer relationship management, e-learning, e-government, and many other domains. The events found in these domains typically refer to activities executed by resources at particular times and for a particular case (i.e., process instances). Process mining techniques are able to exploit such data. In this article, we focus on *process discovery*. However, process mining also includes conformance checking, performance analysis, decision mining, organizational mining, predictions, recommendations, etc. These techniques help to diagnose problems and improve processes. All process mining techniques involve both event data and process models. Therefore, a typical first step is to automatically learn a control-flow model from the event data. This is very challenging, but in recent years many powerful discovery techniques have been developed. It is not easy to compare these techniques since they use different representations and make different assumptions. Users often need to resort to trying different algorithms in an ad-hoc manner. Developers of new techniques are often trying to solve specific instances of a more general problem. Therefore, we aim to unify existing approaches by focusing on *log and model abstractions*. These abstractions link observed and modeled behavior: Concrete behaviors recorded in event logs are related to possible behaviors represented by process models. Hence, such behavioral abstractions provide an "interface" between both. We discuss four discovery approaches involving three abstractions and different types of process models (Petri nets, block-structured models, and declarative models). The goal is to provide a comprehensive understanding of process discovery and show how to develop new techniques. Examples illustrate the different approaches and pointers to software are given. The discussion on abstractions and process representations is also used to reflect on the gap between process mining literature and commercial process mining tools. This facilitates users to select an appropriate process discovery technique. Moreover, structuring the role of internal abstractions and representations helps to broaden the view and facilitates the creation of new discovery approaches.

*Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany

INTRODUCTION

Mainstream data mining and knowledge discovery research and tool development has largely ignored "processes" as people working in logistics, manufacturing, finance, healthcare, customer relationship management, etc. understand them. When trying to improve processes, people in these domains immediately start drawing process models using "boxes and arrows", but these are rarely connected to the underlying data. *Process mining* aims to bridge the gap between process models and event data.[2] The confrontation between *observed behavior* in the form of events and *modeled behavior* in the form of process diagrams helps to diagnose performance and compliance problems.

Surprisingly, process mining did not emerge from data mining, machine learning, or knowledge discovery communities. The roots of process mining lie in the *Business Process Management* (BPM) discipline.[1,24,57] Here, process mining was introduced as a way to infer workflows and effectively use the audit trails present in modern information systems. The increased interest in process mining illustrates that BPM is rapidly becoming more data-driven. *Evidence-based BPM* powered by process mining helps to create a common ground for business process improvement and information systems development. The uptake of process mining is reflected by the growing number of commercial process mining tools available today. There are over 25 commercial products supporting process mining (Celonis, Disco, Minit, myInvenio, ProcessGold, QPR, etc.).[2] All support process discovery and can be used to improve compliance and performance issues. For example, without any modeling, it is possible to learn process models clearly showing the main bottlenecks and deviating behaviors.

This article focuses on *process discovery*, probably the most challenging process mining task. Input for process mining is an *event log*. As shown in Figure 1, an event log "views" a process from a particular angle (see the dashed line). Each event in the log refers to (1) a particular *process instance* (called *case*), (2) an *activity*, and (3) a *timestamp*. There may be additional event attributes referring to resources, people, costs, etc., but these are optional. Given a collection of events, several complementary event logs may be possible. One can change scope or change the case notion, resulting in another view (materialized in the form

of an alternative event log).

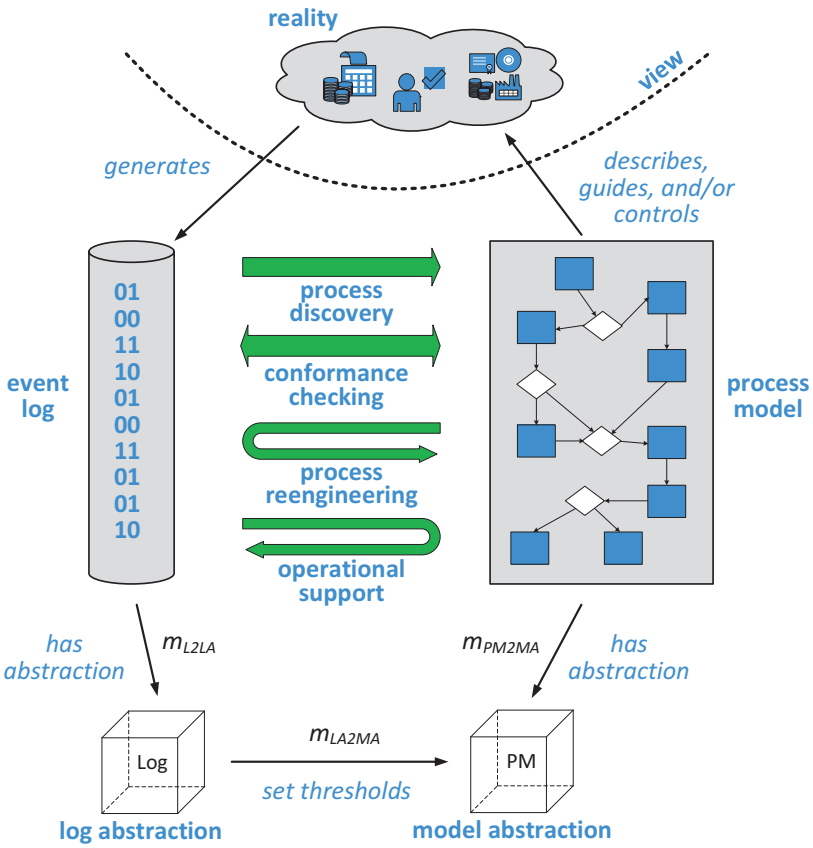


Figure 1: Overview positioning the different types of process mining and the role of log abstractions and model abstractions.

Once a process model has been discovered from an event log it can be used for different purposes, e.g., performance analysis, compliance checking, predictive analytics, etc. Figure 1 shows four main types of process mining: (1) process discovery, (2) conformance checking, (3) process reengineering, and (4) operational support. These will be elaborated later.

To provide an overview of process discovery techniques, we focus on the role of *abstractions*. As shown in Figure 1 log abstractions and model abstractions link observed and modeled behavior. A *log abstraction* characterizes the concrete behavior found in the event log. A *model abstraction* characterizes the possible behavior allowed by the process model. These two abstractions help to unify the different types of behavior. Very different types of abstractions are possible (e.g., directly-follows graphs, transition systems, and constraints),

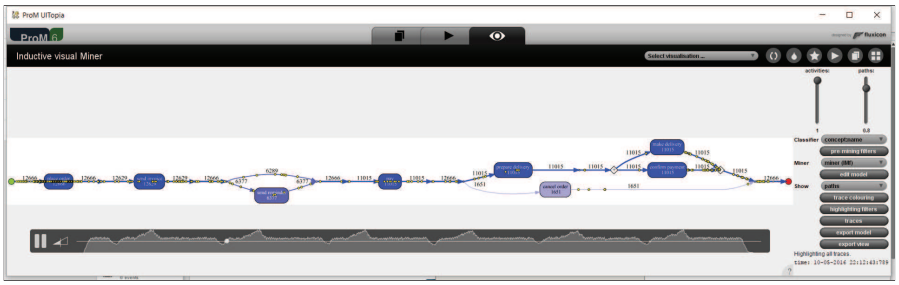
but they all characterize behavior.

This article first provides a short introduction to the different types of process mining, before limiting the scope to process discovery. Rather than presenting one specific discovery technique in detail or providing a survey, this article aims to unify several process discovery techniques. This will not cover all process discovery approaches, but is general enough to cover most. This approach allows us to discuss general principles without focusing on a particular algorithm. Towards the end of this article we will use these insights to discuss the gap between the process discovery algorithms used in commercial tools and the algorithms described in process mining literature. The former use informal models (often filtered directly-follows graphs) whereas the latter use formal models aimed at formally describing a set of possible traces.

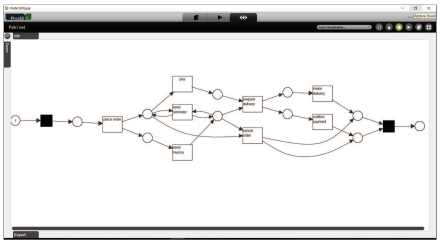
PROCESS MINING: AN OVERVIEW

In recent years, process mining emerged as a new scientific discipline linking process science to data science.[2] Process mining is both *data-driven* and *process-centric*. Progress in this young scientific discipline has been remarkable and this resulted in powerful techniques that are highly scalable. Moreover, process mining tools are increasingly used in practice. There are over 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), *Perceptive Processing Mining* (Lexmark).[2] See the website of the IEEE Task Force on Process Mining for examples of successful case studies.[49] 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.[18] The commercial tools and case studies illustrate the adoption and relevance of process mining.

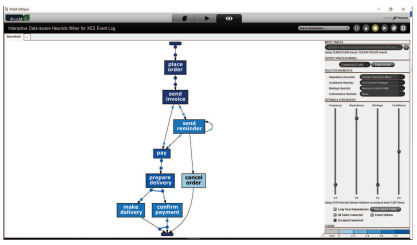
In the academic world, *ProM* is the de-facto standard (www.processmining.org) and research groups all over the world have contributed to the hundreds of *ProM* plug-ins available. The current version of *ProM* provides over 1500 plug-ins. A plug-in may provide a



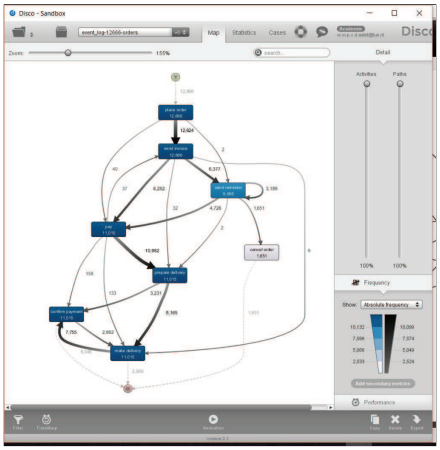
(a) Visual inductive miner (ProM) showing a process tree



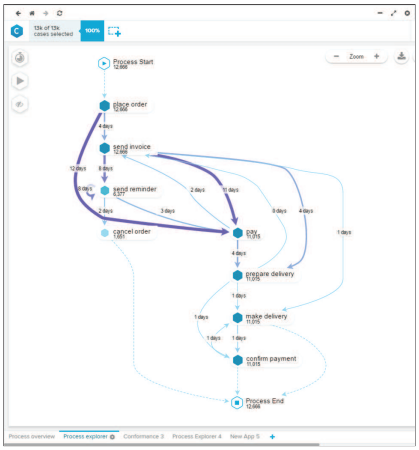
(b) ILP miner (ProM) showing a Petri net



(c) Heuristic miner (ProM) showing a C-net



(d) Disco (Fluxicon) showing a directly follows graph



(e) Celonis showing a directly follows graph

Figure 2: Screenshots of five different process mining tools.

concrete discovery approach or another piece of functionality to support a process mining task. The broad spectrum of *ProM* plug-ins shows that process mining is not limited to process discovery.

Figure 2 shows the output of five different mining tools applied to the same event log containing 121,218 events related to 12,666 cases (customer orders). The first three screenshots show output generated by *ProM* plug-ins. Figure 2(a) shows process tree (visualized in a BPMN-like representation) learned using an inductive mining algorithm.[30–34] Figure 2(b) shows a Petri net learned using an approach based on language-based regions.[14,56] Figure 2(c) shows a Causal net (C-net) learned using a variant of the heuristic mining technique.[39,52]

Figure 2(d) shows a screenshot of Disco. The process model is a directly-follows graph and the numbers indicate frequencies. Figure 2(e) shows another directly-follows graph, but now generated by Celonis Process Mining. The annotations indicate delays. These examples show that different process mining tools may return very different process models using a variety of representations.

A Brief History of Process Mining

Process discovery has its roots in various established scientific disciplines such as concurrency theory, inductive inference, stochastics, data mining, machine learning and computational intelligence. It is impossible to give a complete overview here. Currently there are probably hundreds of process discovery techniques, some of which are discussed later. Here we put these approaches in a historical context.

In 1967 Mark Gold showed in his seminal paper "Language identification in limit" that even regular languages cannot be exactly identified from positive examples only.[26] Many inductive inference problems have been studied since Gold's paper.[12,15] Before the paper of Gold, there were already techniques to construct a finite state machine from a finite set of example traces. It is trivial to construct a finite state machine where the states are described by prefixes of observed traces and transitions move from one state to another by adding an activity to the prefix. Such a finite state machine can be made smaller by using the classical Myhill-Nerode theorem.[44] State are equivalent if their "sets of possible futures" coincide and can be merged. These classical approaches do not consider concurrency, noise, and incompleteness. Therefore, they cannot be used for process discovery in real-life settings based on variable example behavior.

Various data mining algorithms have been developed to find frequent patterns in large datasets.[28,40,48] However, none of these techniques aimed at discovering end-to-end processes. More related is the work on hidden Markov models[11]. Here end-to-end processes can be considered. However, these models are sequential and cannot be easily converted into readable business process models.

In the second half of the 1990s, Cook and Wolf developed process discovery techniques in

the context of software engineering processes.[21] In 1998 two papers appeared that, independently of one another, proposed to apply process discovery in the context of business process management.[10,23] All of these approaches produced sequential models and were based on classical approaches to learn finite state machines or hidden Markov models.

Joachim Herbst aimed at the discovery of more complicated process models at a very early stage.[29] However, it was the α-algorithm that first took concurrency as a starting point and not as an afterthought or post-optimization.[9] Similar ideas were used in the heuristic miner that was refined over time.[52] Many algorithms followed.

Over the last two decades hundreds of process discovery techniques have been proposed. Many of the initial techniques could not cope with infrequent behavior and made very strong assumptions about the completeness of the event log. For example, traditional region-based techniques assume that all behavior possible has been observed and that all behavior observed is equally important. State-based regions were introduced by Ehrenfeucht and Rozenberg in 1989 and generalized by Cortadella et al.[22,25] Various authors used state-based regions for process discovery.[8,47] Also language-based regions have been used for this purpose.[14,56]

Over time attention shifted to approaches able to deal with noise and infrequent behavior. Early approaches include heuristic mining, fuzzy mining, and various genetic process mining approaches.[27,52]

Since 2010 the speed at which new process discovery techniques are proposed is accelerating. As an example consider the family of inductive mining techniques.[31–33]. All of these discovery techniques (IM, IMf, IMc, IMd, IMfD, IMcD, etc.) produce sound models and are highly scalable. Moreover, these algorithms come with formal correctness guarantees. For example, the IM algorithm is fitness-preserving and for particular classes of models even rediscoverability is guaranteed. Note that these are just examples of techniques as the goal is not to provide a complete survey of all approaches.

The focus of this article is on process discovery. However, the process mining discipline is much broader and includes topics such as conformance checking, bottleneck analysis, predictive process analytics, etc.[2] From a historic point of view the PhD thesis of Anne Rozinat is important.[46]. She was the first of work on conformance checking and decision point analysis. Using her software, different perspectives could be discovered and merged

into a single process model. As a result process mining could be combined with simulation and techniques for operational support (e.g., prediction of remaining processing times) could be developed.[2]

Until 2010 there were only a few commercial process mining tools (Futura Reflect by Futura Process Intelligence, Disco by Fluxicon, and Interstage Automated Business Process Discovery by Fujitsu were notable exceptions). Since 2010 there has been a rapid increase in the number of tools and their maturity. For example, Celonis Process Mining (Celonis) was introduced in 2011, minit (Gradient ECM) was introduced in 2014, and ProcessGold Enterprise Platform (ProcessGold) was introduced in 2016. Currently, there are over 25 commercial tools available. These tools can easily deal with event logs having millions of events.

Four Types of Process Mining

As mentioned earlier, the process mining spectrum is quite broad and in this article we focus on process discovery. However, to describe the spectrum we group process mining techniques into the four types depicted in Figure 3. Process mining typically involves both an *event log* (i.e., observed behavior) and a *process model* (i.e., described or prescribed behavior). This naturally leads to the following four types of process mining.

- *Process discovery*: learning process models from event data. A discovery technique takes an event log and produces a process model without using additional information. An example is the α-algorithm which takes an event log and produces a Petri net explaining the behavior recorded in the log.[9]

- *Conformance checking*: detecting and diagnosing both differences and commonalities between an event log and a process model.[3] Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. The process model used as input may be descriptive or normative. Moreover, the process model may have been made by hand or learned using process discovery.

- *Process reengineering*: improving or extending the model based on event data. Like

for conformance checking, both an event log and a process model are used as input. However, now the goal is not to diagnose differences. The goal is to change the process model. For example, it is possible to "repair" the model to better reflect reality. It is also possible to enrich an existing process model with additional perspectives. For example, replay techniques can be used to show bottlenecks or resource usage.[2] Process reengineering yields updated models. These models can be used to improve the actual processes.

- *Operational support*: directly influencing the process by providing warnings, predictions, and/or recommendations.[2] Conformance checking can be done "on-the-fly" allowing people to act the moment things deviate. Based on the model and event data related to running process instances, one can predict the remaining flow time, the likelihood of meeting the legal deadline, the associated costs, the probability that a case will be rejected, etc. The process is not improved by changing the model, but by directly providing data-driven support in the form of warnings, predictions, and/or recommendations.

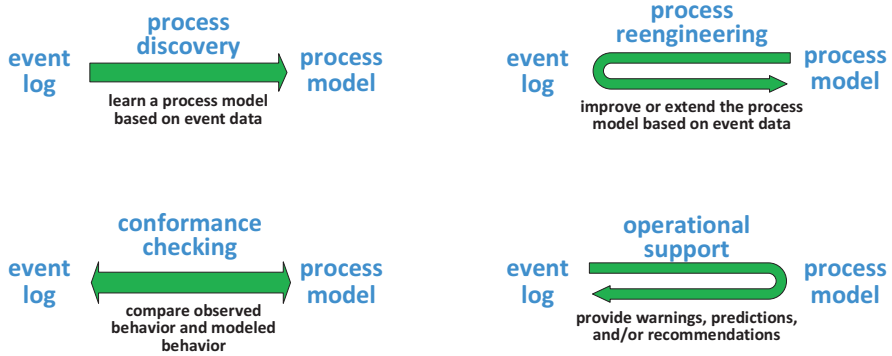


Figure 3: The four basic types of process mining: (1) process discovery, (2) conformance checking, (3) process reengineering (changing the process model), and (4) operational support (influencing the process without reengineering it).

Next to the four types of process mining there are two additional dimensions that can be used to structure the process mining spectrum (see Figure 4). Any process model makes statements about the *control-flow*. The control-flow perspective focuses on the ordering of

activities, i.e., a characterization of all possible sequences of activities. However, next to the control-flow perspective, one or more additional perspectives may be included.[2]

- The *time perspective* is concerned with the timing and frequency of events. When events bear timestamps it is possible to discover bottlenecks, measure service levels, monitor the utilization of resources, and predict the remaining processing time of running cases.

- The *data and decision perspective* focuses on the properties of cases and how these influence behavior. For example, if a case represents a replenishment order, it may be interesting to know the supplier or the number of products ordered. Attributes of events and cases may influence decisions. Therefore, one can learn decision rules and add these to the process model. For example, replenishment orders of less than 15,500 euro tend to skip the pre-approval step.

- The *resource and organization perspective* focuses on information about resources and organizational entities available in the event log. This can be used to learn which actors (e.g., people, systems, roles, and departments) are involved and how are they related. The process model can be extended with role information, work distribution rules, etc.

- Many other perspectives can be added. For example, cost information or information about risks, energy consumption, confidentiality, etc.

As Figure 4 shows that we can also distinguish between online and offline process mining. Offline process-mining techniques work on historic event data and tend to be used to analyze only events that belong to cases that have already completed. Such techniques are typically "backward looking". The goal is to improve the process and not to act immediately. Online process mining techniques may work on streaming event data or databases that are continuously updated. These techniques are typically "forward looking" and deal with cases that are still running. This enables online process mining to influence these running cases without changing the process. Clearly, operational support (the fourth type of process mining in

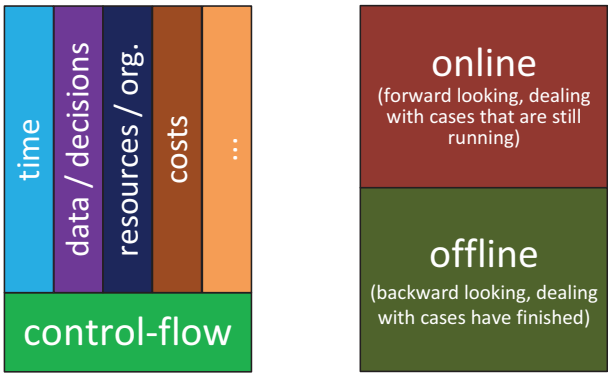


Figure 4: The control-flow perspective is the basis for the other process perspectives (left). Independent of the perspectives included, process mining techniques can be used in online and offline settings (right).

Figure 3) requires an online setting. However, models learned offline may serve as input for the online setting.

Figures 3 and 4 show three mostly orthogonal dimensions: (1) type of process mining, (2) the perspectives covered (next to control-flow), and (3) online versus offline. These illustrate the breadth of the process mining as a discipline. In the remainder, *we focus on process discovery while limiting ourselves to control-flow and the offline setting*. This capability is the starting point for any form of process mining. The control-flow representation is the backbone of any process model and techniques for online process mining are often based on offline techniques.

Process Discovery Through Abstractions

Rather than giving an overview of process discovery techniques or zooming in on a particular technique, we try to unify the lion's share of existing approaches by using the notion of log and model abstractions. Figure 1 already showed the way in which *abstractions* are used. A process discovery algorithm typically builds an abstraction based on the event log using some log abstraction function m_{L2LA}. An event log corresponds to multiset of traces when focusing on the control-flow perspective. There is one trace for each case, but different cases may follow the same path. A process model describes a set of traces. Hence, both event log and model refer to (multi)sets of traces. The log abstraction function m_{L2LA} used to find log

abstractions can be modified to provide model abstractions yielding the model abstraction function m_{PM2MA}. However, there are differences. The log abstraction function needs to deal with the fact that traces may have different multiplicities (e.g., trace $\langle a, b, c, b, d \rangle$ may occur twelve times in a log). The model abstraction function needs to deal with the fact there may be infinitely many traces (e.g., due to loops). The two abstractions need to bridge the gap. Note that the log contains only example behavior. On the one hand, the log may contain exceptional behaviors that do not need to be allowed by the model. On the other hand, in most cases the log contains only a small fraction of all possible behaviors. Hence, one cannot assume that all possible behaviors have been witnessed. For example, in case of loops there may be infinitely many possible traces. The abstractions help to bridge such differences. For example, the abstraction may be robust with respect to loops and concurrency and not all possibilities need to be witnessed. Function m_{LA2MA} may help to bridge further differences, e.g., using frequency based filtering.

Note that m_{L2LA} and m_{PM2MA} are functions, i.e., for each log or model there is precisely one abstraction. However, these functions do not need to be injective. Multiple logs and multiple models may be mapped onto exactly the same abstraction. Hence, if we take an abstraction and apply m_{PM2MA} in reverse direction there may be multiple candidate models. Depending on the class of process models considered, it could even be that exists no model that has such an abstraction. We will elaborate on this later. However, to sketch the overall approach assume that ${m_{PM2MA}}^{-1}$ (i.e., the inverse of m_{PM2MA}) yields a canonical process model. Given this assumption ${m_{PM2MA}}^{-1}(m_{LA2MA}(m_{L2LA}(L)))$ returns a process model for some event log L. This path is visualized at the bottom of Figure 1.

Many discovery approaches are of the form ${m_{PM2MA}}^{-1}(m_{LA2MA}(m_{L2LA}(L)))$, but may use very different abstractions. Later we will give several examples, but first we need to introduce basic concepts such as event logs and process models.

EVENT DATA

We distinguish between *event collections* and *event logs*. To conduct process mining we need to have an event log where each event refers to (1) a particular *process instance* (called case),

(2) an *activity*, and (3) a *timestamp*. The available event collection needs to be preprocessed to create an event log corresponding to the process (i.e., "viewpoint") to be studied. For example, the collection of events needs to be scoped and a suitable case notion has to be chosen. This corresponds to the *view* in Figure 1.

Preliminaries

Before defining event data and event logs, we first introduce some preliminaries used in the remainder.

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in A^*$ denotes a sequence over A. $\sigma(i) = a_i$ denotes the i-th element of the sequence, i.e., σ can be viewed as a function mapping positions onto element values. $|\sigma| = n$ is the length of σ and $dom(\sigma) = \{1, \ldots, |\sigma|\}$ is the domain of σ. $\langle\,\rangle$ is the empty sequence, i.e., $|\langle\,\rangle| = 0$ and $dom(\langle\,\rangle) = \emptyset$. $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences.

$\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $X \in \mathcal{B}(A)$, $X(a)$ denotes the number of times element $a \in A$ appears in X. Some examples: $X = [\,]$, $Y = [x, x, y]$, and $Z = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. X is the empty multiset, Y has three elements ($Y(x) = 2$, $Y(y) = 1$, and $Y(z) = 0$), and Z has six elements. Note that the ordering of elements is irrelevant.

$\mathcal{P}(A)$ is the powerset of A, i.e., all of its subsets. $X \in \mathcal{P}(A)$ if and only if $X \subseteq A$. For example, $\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$.

$f \in A \rightarrow B$ is a total function mapping elements of A onto elements of B. $f \in A \not\rightarrow B$ is a partial functions with domain $dom(f) \subseteq A$, i.e., $f(a)$ is defined if and only if $a \in dom(f)$. Partial functions can be applied to sequences. If $dom(f) = \{a, b\}$, $f(a) = x$, and $f(b) = y$, then $f(\langle a, b, c, a, c \rangle) = \langle x, y, x \rangle$.

Let A be a set and $X \subseteq A$ one of its subsets. $\restriction_X \in A^* \rightarrow X^*$ is a projection function and is defined recursively: $\langle\,\rangle\restriction_X = \langle\,\rangle$ and for $\sigma \in A^*$ and $a \in A$: $(\langle a \rangle \cdot \sigma)\restriction_X = \sigma\restriction_X$ if $a \notin X$ and $(\langle a \rangle \cdot \sigma)\restriction_X = \langle a \rangle \cdot \sigma\restriction_X$ if $a \in X$. For example, $\langle a, b, a \rangle\restriction_{\{a,c\}} = \langle a, a \rangle$. Projection can also be applied to multisets of sequences, e.g., $[\langle a, b, a \rangle^5, \langle a, d, a \rangle^5, \langle a, c, e \rangle^3]\restriction_{\{a,c\}} = [\langle a, a \rangle^{10}, \langle a, c \rangle^3]$.

Event Collections

An event log can be described as a particular view on an event collection. Unlike an event log, an event collection does not have a fixed case notion.

Definition 1 (Event Collection) *Let $\mathcal{E}$ be the universe of event identifiers, $\mathcal{U}_{Att}$ be the universe of attribute names, and $\mathcal{U}_{Val}$ the set of attribute names. $EC = (E, Att, \pi)$ with $E \subseteq \mathcal{E}$, $Att \subseteq \mathcal{U}_{Att}$, and $\pi \in E \rightarrow (Att \not\rightarrow \mathcal{U}_{Val})$ is an event collection. Event $e \in E$ has attributes $dom(\pi(e))$. For $a \in dom(\pi(e))$: $\pi(e)(a)$ is the value of attribute a for event e.*

We distinguish the following types of attributes:

- *Identifier*: the attribute value refers to a particular object (e.g., a customer, a patient, an order, a request, a machine, a room, or a software component). One can only test equality on such attribute values. If x and y are identifier values, then $x = y$ and $x \neq y$ are defined, but $x > y$, $x < y$, $x + y$, $x - y$, etc. have no meaning. Note that an identifier may be represented by a number, but is does not make any sense to add or multiply two identifiers.

- *Class*: the attribute value refers to a category or concept (e.g., an activity name, a status, or a type). One can only test equality on such attribute values and sometimes they are ordered (e.g., high, medium, or low).

- *Time*: the attribute value refers to a particular point in time. This includes absolute and relative timestamps and calendar dates. Examples of time attribute values are `2017-07-16T19:55:30+01:00`, `2017-07-16`, and `19:55:30`. Operations such as $x = y$, $x \neq y$, $x > y$, $x < y$, are defined. One can also add durations, e.g., add a delay of two days.

- *Numerical*: the attribute value represents some kind of measurement. $x = y$, $x \neq y$, $x > y$, $x < y$, $x+y$, $x-y$, etc. are all defined. The values may be discrete or continuous and may be bounded or not.

- *Other*: the attribute value does not fall in one of the above categories and may have more structure (lists, tables, etc.).

Extracting Event Logs

Starting point for process discovery is an event log where events are grouped into cases. Events logs can be extracted from event collections by choosing a particular view, i.e., selecting events and mapping attributes. Each case in an event log L is represented by a trace, e.g., $\langle \triangleright, a, b, c, d, \square \rangle \in L$.

Definition 2 (Event Log) *An* event log $L \in \mathcal{B}(A^*)$ *is a non-empty multiset of traces over some activity set* A*. A trace* $\sigma \in L$ *is a sequence of activities. There is a special start activity* $\triangleright$ *and a special end activity* $\square$*. We require that* $\{\triangleright, \square\} \subseteq A$ *and each trace* $\sigma \in L$ *has the structure* $\sigma = \langle \triangleright, a_1, a_2, \ldots, a_n, \square \rangle$ *and* $\{\triangleright, \square\} \cap \{a_1, a_2, \ldots, a_n\} = \emptyset$*.* $\mathcal{U}_L$ *is the set of all event logs satisfying these requirements.*

An event log captures the observed behavior that is used to learn a process model. An example log is $L_1 = [\langle \triangleright, a, b, c, d, \square \rangle^{45}, \langle \triangleright, a, c, b, d, \square \rangle^{35}, \langle \triangleright, a, e, d, \square \rangle^{20}]$ containing 100 traces and 580 events. $act(L) = \bigcup_{\sigma \in L} \{a \in \sigma\} \subseteq A$ is the set of activities appearing in event log L. For example, $act(L_1) = \{\triangleright, a, b, c, d, e, \square\}$.

The assumption that each trace in an event log starts with $\triangleright$ and ends with $\square$ is not limiting because we can always add such events artificially. However, for process discovery it is helpful to have a unique start and a unique end.

In reality, each event has a timestamp and may have any number of additional attributes. For example, an event may refer to a customer, a product, the person executing the event, associated costs, etc. Here, we abstract from these notions and simply represent an event by its activity name.

How to get from the event collection $EC = (E, Att, \pi)$ (Definition 1) to an event log $L \in \mathcal{B}(A^*)$? This requires the selection of subsets of events in $E_1, E_2, \ldots, E_n \subseteq E$. This way it is possible to scope the process. Events in $E \setminus (E_1 \cup E_2 \cup \ldots \cup E_n)$ are discarded. Per subset E_i there needs to be a mapping relating the set of attribute names in A to process mining concepts such as case, activity, timestamp, etc. We allow for multiple subsets of events in $E_1, E_2, \ldots, E_n$ each having a separate mapping, because not all events need to have the same set of attributes. The event sets may be overlapping (e.g., when events in the event collection have multiple timestamp attributes).

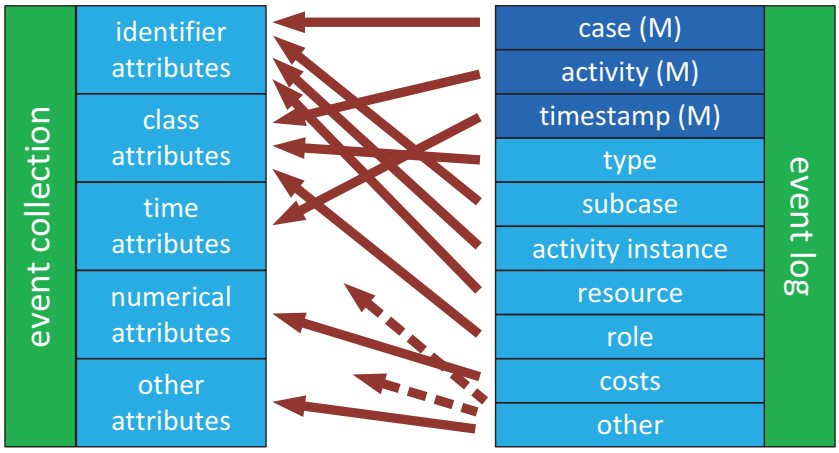


Figure 5: Extracting an event log (right) from a collection of events (left). Each event in an event log has a case, activity, and timestamp.

Figure 5 illustrates such mappings. Technically, there are mappings $m_1, m_2, \ldots m_k \in \{case, activity, timestamp, \ldots\} \not\to Att$ such that $\{case, activity, timestamp\} \subseteq dom(m_i)$. For example, the case of an event $e \in E_i$ is determined by the $\pi(e)(m_i(case))$ value, the timestamp of e is $\pi(e)(m_i(timestamp))$, etc. Events in E_i are only included if the mandatory attributes ($dom(m_i)$) are defined. Each pair (E_i, m_i) yields a set of events having a case, activity, and timestamp. These together form the event log.

In $L \in \mathcal{B}(A^*)$ we abstract from the actual timestamp and case of an event. Events corresponding to the same case form a sequence of activities. The case attribute is used to group cases and the timestamp attribute is used to order events. Here we assume a total order within each case. However, there are also process mining techniques that use partial orders as input.

Next to the three mandatory events attributed marked with an "M" in Figure 5, there are several other (optional) attributes that have a meaning that is specific for process mining.

- *Type*: to capture transactional information (e.g., start, complete, schedule, abort, suspended, etc.).
- *Subcase*: to group events within and between cases (e.g., identifying different order lines within an order). This can be used as a secondary case notion. It can be used to separate causally unrelated events within a case (e.g., events related to different order

lines within an order should not be related) or to establish links between cases (e.g., different orders grouped into a single payment or delivery).

- *Activity instance*: to relate events corresponding to the same activity execution (e.g., link events of type *start* to events of type *complete*).
- *Resource*: to relate events to a specific resource (worker, machine, etc.).
- *Role*: to relate events to a collection of resources (e.g., an organizational unit or specific qualifications).
- *Costs*: to add cost information to events.
- *Other*: any other attribute that may be relevant for the execution of cases. For example, attributes influencing decisions or durations.

In the remainder we focus on the core discovery problem and assume an event log abstracts from these additional attributes. However, it is good to understand that $L \in \mathcal{B}(A^*)$ provides a simplified viewpoint (on a much more complex reality that extends far beyond this article).

PROCESS MODELS

A *formal process model* is able to make firm statements about the inclusion or exclusion of traces, e.g., trace $\langle \triangleright, a, b, c, d, \square \rangle$ fits the model or not. *Informal process models* are unable to make such precise statements about traces. Here, we focus on formal process models. Using this assumption we can view process models as classifiers for traces: a trace fits or not. This leads to the following generic notion.

Definition 3 (Process Model) *A* process model $PM \in \mathcal{P}(A^*)$ *corresponds to a set of traces over some activity set* A*.*

Of course one does not model a process by simply enumerating all possible traces. Modeling languages aim to describe sets of traces in a compact and intuitive manner. In this

article we consider three notations: Petri nets, a particular form of structured process models (process trees), and a particular form of declarative process models (similar to Declare). These are described next.

Petri Nets

Petri nets are probably the oldest and best investigated process modeling language allowing for the modeling of concurrency. Figure 6 shows an example model allowing for traces like $\langle a, b, c, d\rangle$, $\langle a, c, b, d\rangle$, $\langle a, b, c, e, f, c, b, d\rangle$, etc.

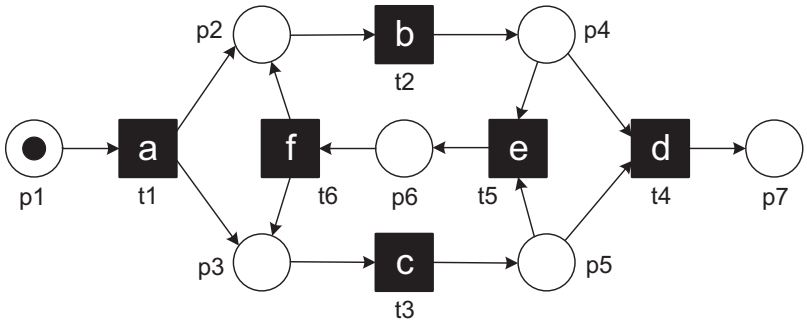


Figure 6: A Petri net with six transitions $\{t1, t2, \ldots, t6\}$ and seven places $\{p1, p2, \ldots, p7\}$. The initial marking $M_{init} = [p1]$ is shown. $M_{final} = [p7]$ is the final marking.

A Petri net is a bipartite graph composed of places (represented by circles) and transitions (represented by squares).

Definition 4 (Petri Net) *A Petri net is a tuple $N = (P, T, F)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation. A labeled Petri nets $N = (P, T, F, l, A)$ extends the Petri nets with a labeling function $l \in T \not\rightarrow A$.*

Transitions may represent activities. Transitions without a label are called "silent". If $l(t) = a$, then a is the observed activity when transition t is executed. For example, $l(t1) = a$, $l(t2) = b$, etc. in Figure 6. Note that the labeling function in the example is total (no silent transitions) and injective (no two transitions with the same label). Places are added to model causal relations. $\bullet x = \{y \mid (y, x) \in F\}$ and $x\bullet = \{y \mid (x, y) \in F\}$ define input and output sets of places and transitions.

The state of a Petri net, called *marking*, indicates how many *tokens* each place contains. Formally: marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$. Tokens are shown as block dots inside places. Figure 6 shows the initial marking $[p1]$.

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places ($p \in {\bullet}t$) contains at least one token. An enabled transition t may *fire*, i.e., one token is removed from each of the input places ($p \in {\bullet}t$) and one token is produced for each of the output places ($p \in t{\bullet}$).

$(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'. Let $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \leq i < n$.

A system net has an initial and a final marking. The *behavior* of a system net corresponds to the visible behaviors of the set of all traces starting in the initial marking M_{init} and ending in the final marking M_{final}. Note that deadlocking or livelocking traces that do not reach M_{final} are not part of the behavior.

Definition 5 (System Net Behavior) *A system net is a triplet $SN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l, A)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. $PM_{SN} = \{l(\sigma) \mid \sigma \in T^* \ \wedge \ (N, M_{init})[\sigma\rangle(N, M_{final})\}$ is the set of traces possible according to the model.*

In the preliminaries we explained that partial functions can be applied to sequences. Hence, $l(\sigma)$ yields a sequence of activities. Transitions that are silent are skipped. Visible transitions are replaced by the corresponding activities.

For the system net in Figure 6: $PM_{SN} = \{\langle a, b, c, d\rangle, \langle a, c, b, d\rangle, \langle a, b, c, e, f, b, c, d\rangle, \langle a, c, b, e, f, b, c, d\rangle, \ldots \langle a, c, b, e, f, b, c, e, f, b, c, e, f, c, b, d\rangle, \ldots\}$. Note that a system net classifies traces σ into *fitting* ($\sigma \in PM_{SN}$) and *non-fitting* ($\sigma \notin PM_{SN}$). Hence, system nets provide a concrete language for the more abstract notion introduced in Definition 3.

Structured Process Models

Petri nets, WF-nets, BPMN models, EPCs, and UML activity diagrams may suffer from deadlocks, livelocks, and other anomalies. Models having undesirable properties *independent* of the event log, are called *unsound*. In principle no mining technique should return an unsound model. One can use relaxed soundness notions. For example, one can consider $PM \in \mathcal{P}(A^*)$ to be only the behavior that does not result in deadlocks. However, in some cases discovered models may have no sound executions, i.e., $PM = \emptyset$. Consequently, one cannot interpret the model or assess its quality. Moreover, users will have difficulties to discard deadlocking and livelocking behavior from a model.

Structured process models, also referred to as *block-structured models*, are sound by construction. In this article we use *process trees* as a notation to represent such block-structured models. Process trees are tailored towards process discovery. A process tree is a hierarchical process model where the (inner) nodes are operators such as sequence and choice and the leaves are activities. A range of *inductive process discovery* techniques exists for process trees.[30–33] These techniques benefit from the fact that the representation ensures soundness.

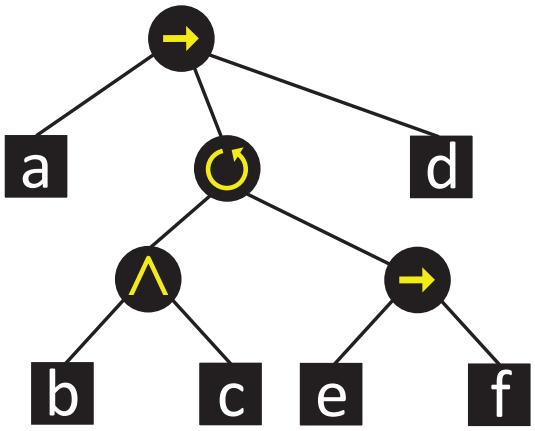


Figure 7: Process tree $\rightarrow(a, \circlearrowleft(\wedge(b, c), \rightarrow(e, f)), d)$.

Figure 7 shows a process tree that is behaviorally equivalent to the system net in Figure 6. The inner nodes of the process tree represent operators. The leaves represent activities. There is one root node. There are four types of operators that can be used in a process tree: $\rightarrow$ (sequential composition), $\times$ (exclusive choice), $\wedge$ (parallel composition), and $\circlearrowleft$ (redo

loop). The process tree in Figure 7 can also be represented textually:

$$\rightarrow(a, \circlearrowleft(\wedge(b, c), \rightarrow(e, f)), d)$$

The root node is a sequence operator stating that the process starts with activity a and ends with activity d. In-between these two activities there is the subprocess $\circlearrowleft(\wedge(b, c), \rightarrow(e, f))$. The redo loop starts with its leftmost child and may loop back through any of its other children. In the subtree $\circlearrowleft(\wedge(b, c), \rightarrow(e, f))$, the leftmost child ("do part") is $\wedge(b, c)$, i.e., the parallel execution of b and c, and the rightmost child ("redo part") is $\rightarrow(e, f)$ modeling the sequence of e followed by f.

In this article, we do not provide formal definitions for process trees and their semantics, but use some examples to explain the behaviors they represent. Let $PT_1 = \wedge(\times(a, b), c)$, i.e., a choice between a and b running in parallel with c. The corresponding behavior is $PM_{PT_1} = \{\langle a, c\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle\}$. If $PT_2 = \circlearrowleft(\rightarrow(a, b), c)$, then $PM_{PT_2} = \{\langle a, b\rangle, \langle a, b, c, a, b\rangle, \langle a, b, c, a, b, c, a, b\rangle, \ldots\}$.

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 have precisely one possible trace: $\langle a, a, a\rangle$. A silent activity is denoted by τ and cannot be observed. Process tree $\times(a, \tau)$ can be used to model an activity a that can be skipped. Process tree $\circlearrowleft(a, \tau)$ can be used to model the process that executes a at least once. The "redo" part is silent, so the process can loop back without executing any activity. Process tree $\circlearrowleft(\tau, a)$ models a process that executes a any number of times. The "do" part is now silent and activity a is in the "redo" part.

The mapping from process trees to behaviorally equivalent systems nets is easy. Note that this is possible because we use labeled Petri nets. Different transitions can have the same label and we can add transitions that are not observable (these correspond to τ and can be used to model choices in a block-structured manner). Similar translations are possible to BPMN models, EPCs, and UML activity diagrams.

Declarative Process Models

Declarative, constraint-based languages like *Declare*[7], start from an open-world assumption: anything is possible unless there is a constraint explicitly forbidding it. *Declare* has been formalized in terms of Linear Temporal Logic (LTL) over finite traces[6], using abductive logic programming[42], Event Calculus[43], regular expressions[19], and colored automata[38]. Here we use a simpler approach based on the cardinality constraints shown in Figure 8.

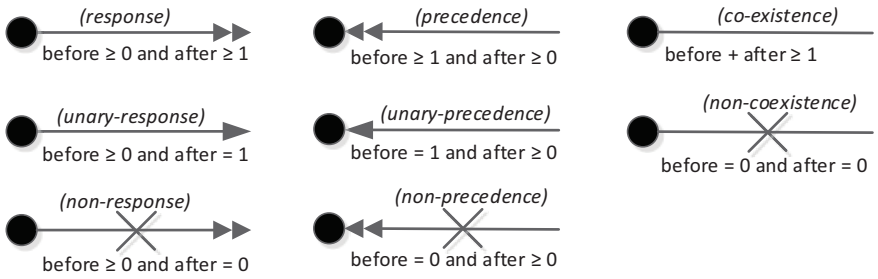


Figure 8: Declarative notations: Eight example constraints.

Given some *reference event* e we can reason about the events *before* e and the events *after* e. We may require that the cardinality of the set of events corresponding to a particular activity before or after the reference event lies within a particular range. The back dots in Figure 8 refer to reference events.

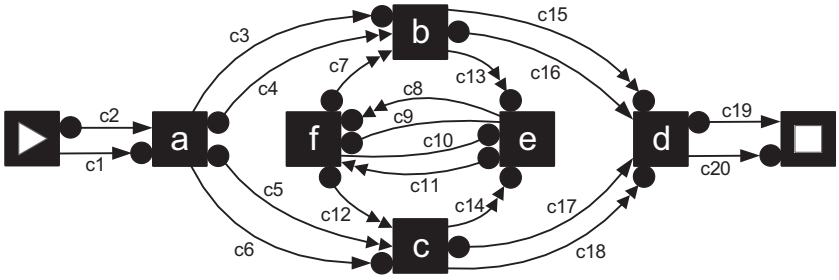


Figure 9: A declarative model.

Rather than providing formal semantics, we explain the notation using an example. Constraint $c1$ in Figure 9 models that every a event should be preceded by precisely one start event ($\triangleright$). Traces like $\langle \triangleright, a, b, c \rangle$, $\langle \triangleright, b, a, c \rangle$, and $\langle \triangleright, a, a, c \rangle$ satisfy this constraint. However, traces like $\langle a, b, c \rangle$, $\langle a, \triangleright, b, c \rangle$, and $\langle \triangleright, a, \triangleright, a, c \rangle$ do not. How to check this? Each a appearing

anywhere in the trace serves as a reference event. For each reference event we count the number of b's before and after. This yields the numbers *before* and *after* mentioned in Figure 8. Consider for example trace $\langle \triangleright, a, \triangleright, a, c \rangle$ and constraint $c1$. There are two reference events (both a's). For the first a: $\mathit{before} = 1$ and $\mathit{after} = 1$. This satisfies the constraint $\mathit{before} = 1$ and $\mathit{after} \geq 0$, so the first a meets the requirement. For the second a: $\mathit{before} = 2$ and $\mathit{after} = 0$. This violates the constraint, so the second reference event a does not meet the requirement.

Constraint $c2$ in Figure 9 states that every start event ($\triangleright$) event should be followed by precisely one a event. Constraint $c3$ states that every b event should be preceded by precisely one a event. Constraint $c4$ states that every a event should be followed by at least one b event. Etc.

Given a declarative model DM, PM_{DM} is the set of all traces that meet all constraints, i.e., for each constraint and any of the corresponding reference events, the values *before* and *after* need to be in the specified range. Obviously, Figure 9 allows for much more behavior than the system net in Figure 6 and process tree in Figure 7. For example, $\langle \triangleright, a, b, c, b, c, d, \square \rangle \in PM_{DM}$. Declarative models are particular useful when allowing for lots of flexibility. However, such models become convoluted when trying to restrict the behavior to a structured set of traces.

USING ABSTRACTIONS

Given an event log which provides a particular view on some collection of events, we would like to discover a process model. As input we assume $L \in \mathcal{B}(A^*)$, i.e., a non-empty multiset of traces over some activity set A. The output is a process model, e.g., a system net or a process tree, defining a set of traces $PM \in \mathcal{P}(A^*)$. Discovery approaches typically operate on an abstraction of the event log as a means to relate observed behavior to modeled behavior. Figure 10 details the lower part of Figure 1.

A log abstraction function m_{L2LA} maps an event log L onto a log abstraction $m_{L2LA}(L)$ (e.g., a weighted directly-follows graph). The log abstraction is converted into a model abstraction using m_{LA2MA}. In this step one may use thresholds to remove exceptional be-

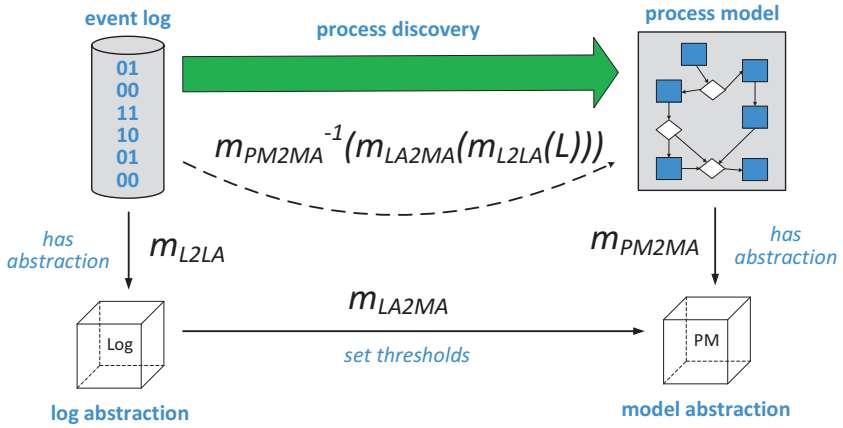


Figure 10: Approach based on abstractions.

havior or "massage" the log abstraction in some other way. Given a model abstraction $m_{LA2MA}(m_{L2LA}(L))$, we are interested in all models that are mapped onto this abstraction using model abstraction function m_{PM2MA}. Obviously, here may be multiple models having the same abstraction. However, depending on the abstraction used, it could also be that there are no process models having the model abstraction $m_{LA2MA}(m_{L2LA}(L))$. This is ultimately determined by the representational bias of the modeling language used. We would like to use the inverse of m_{PM2MA}. However, ${m_{PM2MA}}^{-1}$ corresponds to a possibly non-empty set of process models. This can be resolved in the following way. One can assume a canonical process model given some abstraction. If there is no corresponding model, one can always use a fall-through resulting in a very general model, e.g., the model which allows for any behavior over some activity set A. If we assume that ${m_{PM2MA}}^{-1}$ returns the canonical process model for the abstraction used as input, then ${m_{PM2MA}}^{-1}(m_{LA2MA}(m_{L2LA}(L)))$ turns an event log into a process model. This is shown in Figure 10.

In principle any abstraction can be used. However, in the remainder we show three characteristic abstractions: directly-follows graphs, transition systems, and cardinality constraint models.

Directly-Follows Graphs

Many discovery approaches use the directly-follows graph as an abstraction. This graphs simply shows whether one activity is (often) followed by another activity. A notable example is α-algorithm[9], which was the first discovery algorithm taking concurrency as a starting point. Other approaches that use the directly-follows graph (sometimes in conjunction with other abstractions) include variants of the α-algorithm[41,54,55], fuzzy mining[27], heuristic mining[52,53], and inductive mining[30–33].

Definition 6 ((Weighted) Directly-Follows Graph) *$DFG = (A, R)$ is a directly-follows graph if A is a set of activities with $\{\triangleright, \square\} \subseteq A$ and $R \subseteq A \times A$ is a relation such that $\{(a, \triangleright) \in R \mid a \in A\} = \{(\square, a) \in R \mid a \in A\} = \emptyset$.*

$WDFG = (A, R_w)$ is the weighted directly-follows graph of event log L if $A = act(L)$ and $R_w = \bigcup_{\sigma \in L} [(a, b) \in A \times A \mid \exists_{1 \leq i < |\sigma|}\ \sigma(i) = a\ \wedge\ \sigma(i+1) = b]$.

Computing the weighted directly-follows graph of event log L corresponds to $m_{L2LA}(L)$ in Figure 10. $WDFG = (A, R_w)$ captures how often some activity a is followed by some activity b: $R_w(a, b)$ denotes this frequency. Using some function m_{LA2MA}, the weighted directly-follows graph is transformed into a non-weighted directly-follows graph. In this step activities may be removed and relations below some threshold may be removed. ${m_{PM2MA}}^{-1}$ is applied to the graph resulting in a process model (e.g., a Petri net or process tree).

Transition Systems

There are various approaches to convert an event log to a transition system which is then converted to a process model.[8] Positions in the traces in the event log are translated to states in the transition system.

Definition 7 ((Weighted) Transition System) *$TS = (A, S, S_{init}, S_{final}, R)$ is a transition system if A is a set of activities with $\{\triangleright, \square\} \subseteq A$, S is a set of states, $S_{init} \subseteq S$ is the set of initial states, $S_{final} \subseteq S$ is the set of final states, and $R \subseteq S \times A \times S$ is a relation such that $\{(s_1, \triangleright, s_2) \in R \mid s_1 \notin S_{init}\} = \{(s_1, \square, s_2) \in R \mid s_2 \notin S_{final}\} = \emptyset$.*

$WTS = (A, S, S_{init}, S_{final}, R_w)$ is the weighted transition system of L for given a state function $state \in A^* \times A^* \to S$ if $A = act(L)$, $S = \{state(\sigma_1, \sigma_2) \mid \exists_{\sigma \in L}\ \sigma = \sigma_1 \cdot \sigma_2\}$, $S_{init} = \{state(\langle\ \rangle, \sigma) \mid \sigma \in L\}$, $S_{final} = \{state(\sigma, \langle\ \rangle) \mid \sigma \in L\}$, $R_w = \bigcup_{\sigma \in L} [(state(\sigma_1, \langle a \rangle \cdot \sigma_2), a, state(\sigma_1 \cdot \langle a \rangle, \sigma_2)) | \sigma = \sigma_1 \cdot \langle a \rangle \cdot \sigma_2]$.

The state of a case is determined using state function $state \in A^* \times A^* \to S$. Consider a case following trace $\sigma = \sigma_1 \cdot \sigma_2$ after the occurrence of the activities in σ_1, i.e., σ_1 has a happened and σ_2 still has to happen. The case is then in state $state(\sigma_1, \sigma_2)$.

Examples of state functions include:

- $state(\sigma_1, \sigma_2) = \sigma_1$: the whole history σ_1 matters (the future σ_2 has no influence on the state),
- $state(\sigma_1, \sigma_2) = \{a \in \sigma_1\}$: only the set of activities that occurred in the past matters, and
- $state(\sigma_1, \sigma_2) = [a \in \sigma_1]$: the multiset of activities that occurred in the past matters.

It is also possible to let the state depend on all future activities, the next activity, the last two activities, etc.

Computing the weighted transition system corresponds to applying a log abstraction function $m_{L2LA}(L)$. Relation R_w captures how often one state is followed by another. Let $\sigma = \sigma_1 \cdot \langle a \rangle \cdot \sigma_2$ be the trace of some case involving activity a at some position. $state(\sigma_1, \langle a \rangle \cdot \sigma_2)$ is the state before the occurrence of a and $state(\sigma_1 \cdot \langle a \rangle, \sigma_2)$ is the state after the occurrence of a. R_w is computed by considering all cases and all positions in the corresponding traces.

Using some function m_{LA2MA}, the weighted transition system is transformed into an ordinary transition system that can be converted onto a process model. ${m_{PM2MA}}^{-1}$ is applied to resulting transition system to get a Petri net or some other model having the behavior captured in the transition system.

For example, there are various so-called *state-based region approaches* able to translate a transition system into a behaviorally equivalent Petri net.[17,22] There may be several system nets having the same transition system (modulo renaming or bisimulation), e.g., adding a redundant place does not change the behavior. This shows that m_{PM2MA} is not injective.

However, it is possible to choose one of the behaviorally equivalent Petri nets, thus realizing ${m_{PM2MA}}^{-1}$. Note that the abstraction provided by $m_{L2LA}(L)$ is essential to be able to synthesize a Petri net that is not severely overfitting the event log.

Cardinality Constraint Model

The directly-follows graph only looks at the next activity. This can be generalized to capture more general constraints among activities. Given an event corresponding to activity a we may look at the b events before and after it. For example, this particular a event may be preceded by 3 b events, but is never followed by a b event. Such observations can be used to formulate cardinality constraints. An example constraint could be "Any a event may be preceded by an arbitrary number of b events, but can never be followed by a b event."

Declarative languages like *Declare* start from the assumption that anything is possible as long as it is not explicitly forbidden by a constraint.[7] Several discovery techniques have been proposed for *Declare*.[19,36,37] A recently proposed declarative process modeling language is the class of Object-Centric Behavioral Constraint (OCBC) models. OCBC models allow for multiple intertwined instance notions and can be discovered by monitoring database updates.[35] Most of these approaches simply evaluate many, if not all, possible constraints. Such approaches are rather time-consuming (depending on the constraint types allowed). However, for some constraints it is also possible to compute an intermediate structure and use this to generate a model. We illustrate this by formalizing a particular abstraction: (weighted) cardinality constraint models. These are employed as an abstraction to mediate between event log and process model.

Definition 8 ((Weighted) Cardinality Constraint Model) *$CCM = (A, C, R)$ is a cardinality constraint model if A is a set of activities with $\{\triangleright, \square\} \subseteq A$, C is a set of disjoint cardinalities, and $R \subseteq A \times A \times C$ is a relation.*

$WCCM = (A, C, R_w)$ is the weighted cardinality constraint model of L for given a cardinality function $card \in (A^* \times A \times A^*) \times A \to C$ if $A = act(L)$, $C = \{card((\sigma_1, a, \sigma_2), b) \mid \exists_{\sigma \in L}\ \sigma = \sigma_1 \cdot \langle a \rangle \cdot \sigma_2 \wedge b \in A\}$, $R_w = \bigcup_{\sigma \in L} [(a, b, card((\sigma_1, a, \sigma_2), b)) | \sigma = \sigma_1 \cdot \langle a \rangle \cdot \sigma_2 \wedge b \in A]$.

A cardinality constraint model $CCM = (A, C, R)$ consists of activities A, possible car-

dinalities C, and a relation $R \subseteq A \times A \times C$. $(a, b, c) \in R$ means that the cardinality c is possible for a and b. Note that the roles of a and b are different. The cardinality always considers one a event in relation to all b events before and after it (within a case). Such an a event is called a *reference event*. The b events before and after this reference event are called *target events*. The above definition is generic and does not fix a particular cardinality notion. However, for a particular reference a event and b activity there should be precisely one cardinality. If $(a, b, c) \notin R$, then cardinality c is not possible for any reference a event and b activity. Although cardinalities are mutual exclusive at the event level, there may be different reference events having different cardinalities.

The weighted cardinality constraint model counts the number of times (a, b, c) is observed. Each event serves as a reference event for each of the activities. Hence, the number of observations is equal to the number of events times the number of activities.

Cardinality function $card \in (A^* \times A \times A^*) \times A \to C$ determines the semantics of the relations in R. $card((\sigma_1, a, \sigma_2), b)$ is the cardinality of an a event preceded by σ_1 and followed by σ_2 with respect to activity b. Some example functions:

- $card_1((\sigma_1, a, \sigma_2), b) = (bin(\left|\sigma_1{\restriction}_{\{b\}}\right|), bin(\left|\sigma_2{\restriction}_{\{b\}}\right|))$ with $bin(0) = 0$, $bin(1) = 1$, $bin(k) = *$ if $k \geq 2$ yields observations that indicate both whether there were no, one, or more b events before the reference a event and whether there were no, one, or more b events after the same reference a event.

- $card_2((\sigma_1, a, \sigma_2), b) = (\left|\sigma_1{\restriction}_{\{b\}}\right|, \left|\sigma_2{\restriction}_{\{b\}}\right|)$ yields observations that indicate the number of b events before and after the reference a event.

- $card_3((\sigma_1, a, \sigma_2), b) = (hd(\sigma_2) = b)$ where $hd(\langle a_1, a_2, \ldots, a_n \rangle) = a_1$ and $hd(\langle\ \rangle) = \bot$ yields the directly-follows relation: when the reference a events was followed immediately by a b event *true* is returned, otherwise *false*.

- $card_4((\sigma_1, a, \sigma_2), b) = \left|(\sigma_1 \cdot \sigma_2){\restriction}_{\{b\}}\right|$ yields observations that indicate the total number of b events before and after the reference a event.

For each reference event (a in some trace $\sigma_1 \cdot \langle a \rangle \cdot \sigma_2 \in L$) and activity b we get an observation. By counting the number of observations we can get a "footprint" of the relation

between a and b. Consider for example $card_1$ which allows for nine different observations, i.e., $C = \{(0,0),(1,0),(*,0),(0,1),(1,1),(*,1),(0,*),(1,*),(*,*)\}$. For a given a and b we can count the observations for each element of C. The ones that are above a threshold remain and are used to build a model. It is easy to see that this way it is relatively easy to build the declarative models described before (cf. Figure 8). For example, if $\{(0,1),(1,1),(*,1)\} \subseteq C$ are all observed frequently and the rest not, then it makes sense to add a unary response. If $\{(0,0),(1,0),(*,0)\} \subseteq C$ are all observed frequently and the rest not, then a non-response constraint can be added. If only $(0,0) \in C$ is observed frequently and the rest not, then a non-coexistence constraint can be added. Note that these are merely examples and in reality things are often not so clear-cut. However, the examples illustrate that the intermediate abstraction allows for the discovery of declarative process models.

In this section we introduced three types of abstractions illustrating the general principle described in Figure 10. For each abstraction we introduced a weighted variant that can be obtained by applying the log abstraction function m_{L2LA} to some event log L. These weighted variants count some relation. By setting thresholds or using some other cut-off criterion, we can get a non-weighted variant of the abstraction (apply m_{LA2MA}). Based on this we return a model that has the same abstraction (apply ${m_{PM2MA}}^{-1}$). We deliberately did not describe any of the discovery approaches in detail. The goal was to conceptualize process discovery.

BRIDGING THE GAP

The focus thus far has been on formal process models. Therefore, we could state that a process model $PM \in \mathcal{P}(A^*)$ corresponds to a set of traces over some activity set A. Petri nets, structured process models (process trees), declarative process models (Declare) were used as examples. Such formal models are able to classify traces (i.e., sequences of events) as fitting or non-fitting, and provide explicit constructs to model sequences, concurrency, choices, and loops (or their declarative counterparts). Commercial process mining tools remain *deliberately vague* for reasons of *scalability* (dealing with millions of events) and *simplicity* (results need to be understandable by end-users). The 25 commercial products

mentioned before (e.g., Celonis, Disco, Minit, myInvenio, ProcessGold, and QPR) mostly rely on discovery techniques that produce *filtered directly-follows graphs*. Earlier we defined $WDFG = (A, R_w)$ to be the weighted directly-follows graph of event log L with $A = act(L)$ and $R_w = \bigcup_{\sigma \in L} [(a, b) \in A \times A \mid \exists_{1 \leq i < |\sigma|} \ \sigma(i) = a \ \wedge \ \sigma(i+1) = b]$. $R_W(a, b)$ denotes how many times a was followed directly by b. We can set a threshold on the inclusion of arcs, i.e., a and b are connected if $R_W(a, b)$ is above some threshold. This threshold can be absolute or relative to the other weights. If we set the threshold to 1 (i.e., a and b are connected if and only if $R_W(a, b) \geq 1$), then the resulting graph can be viewed as a transition system or Markov chain (since only the last activity matters). Such a model is often underfitting (introducing loops that may not exist) and complex (many connections). However, when the threshold is set higher (e.g., $R_W(a, b) \geq 100$), then it becomes impossible to view the filtered directly-follows graph as a classifier for traces. Which combinations of output arcs can be combined? Showing frequencies on nodes (activities) and arcs may further add to the confusion because "numbers do not add up".

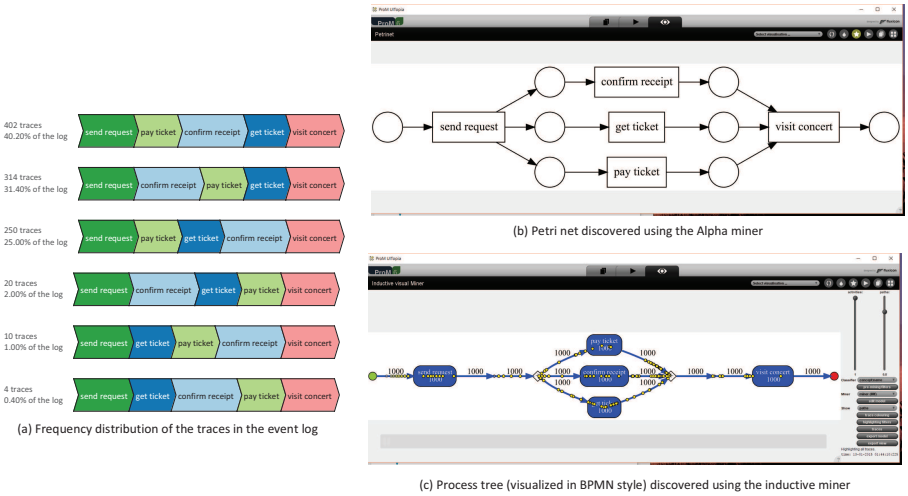


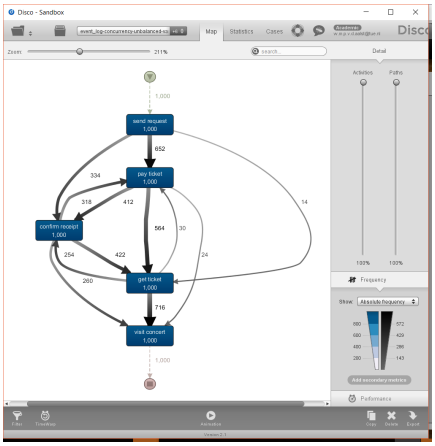
Figure 11: A simple synthetic event log and two formal models derived from it.

To illustrate the problem let us look at a very simple process for which we created an artificial event log $L = \langle \triangleright, sr, pt, cr, gt, vc, \square \rangle^{402}, \langle \triangleright, sr, cr, pt, gt, vc, \square \rangle^{314}, \langle \triangleright, sr, pt, gt, cr, vc, \square \rangle^{250}, \langle \triangleright, sr, cr, gt, pt, vc, \square \rangle^{20}, \langle \triangleright, sr, gt, pt, cr, vc, \square \rangle^{10}, \langle \triangleright, sr, gt, cr, pt, vc, \square \rangle^{4}$ where sr refers to activity "send request", pt refers to "pay ticket", cr refers to "confirm receipt", gt refers to "get ticket", and vc refers to "visit concert". Figure 11 visualizes the event log and two

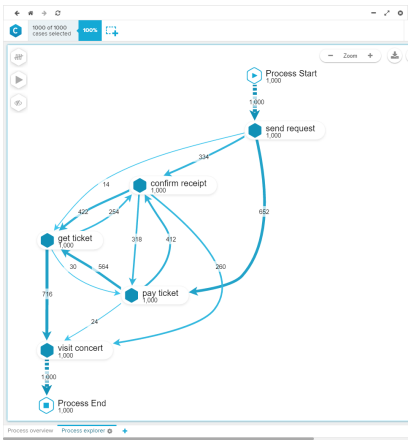
formal models. One model (b) is discovered by the α-algorithm and the other one (c) by the inductive miner. Both models allow for the traces observed (and nothing more) and are quite simple. Therefore, we consider these models to be an adequate description of the behavior observed. Actually the event log was generated from a simulation model in CPN Tools having the structure shown in Figure 11(b). If we feed the same event log to a commercial process mining tool, we are unable to capture this behavior. Figure 12 illustrates the problem using Disco and Celonis as examples. The unfiltered directly-follows graph is complex and also underfitting, Figure 12(a) and (b) show multiple loops suggesting behaviors that are not possible. Each activity is executed precisely once for each case. The models discovered by commercial tools do not show the split and join behavior (AND, XOR, or OR) in the process model. Some of these tools are able to discover concurrency under particular circumstances, but this is not shown. Sometimes concurrency is only detected when activities overlap (Disco) or additional information is given (Celonis). However, in both cases, this is not exposed to the user.

The filtered directly-follows graphs in Figure 12(c) and (d) show that the model can be simplified, but this leads to inconsistent models. First of all, the activity frequencies and the arc frequencies do not match. See for example activity "visit concert" that occurred 1000 times, but the only input arc has a frequency of only 716. Second, the diagrams seem to suggest that it is possible to bypass "confirm receipt", but this never happens. Finally, the computed delays (not shown in Figure 12) are very misleading. The times returned by Disco and Celonis are "conditional delays". If "send request" is followed directly by "pay ticket", then the average delay is 44 hours. However, the actual delay between "send request" and "pay ticket" is 51 hours.

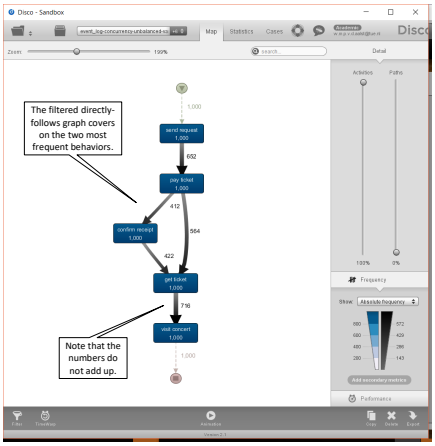
This small example shows that filtered directly-follows graphs cannot be viewed as formal models and may lead to invalid conclusions. Formal models are able to classify traces into fitting and non-fitting. Looking at the filtered directly-follows graphs in Figure 12(c) and (d) does not really provide insights into which traces are fitting the model. The model generated by both Disco and Celonis suggests that $\langle \triangleright, sr, pt, cr, gt, vc, \square \rangle$ and $\langle \triangleright, sr, pt, gt, vc, \square \rangle$ are the two traces possible. However, the second trace never happened (there is no "confirm receipt") and the first trace represents only one of six variants.



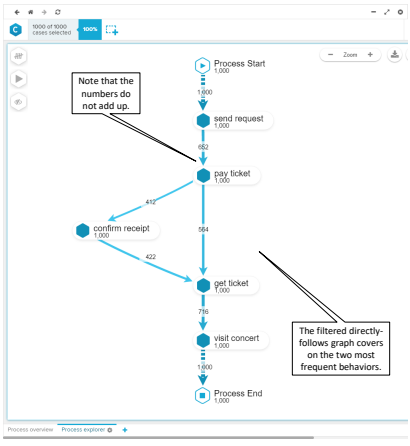
(a) Disco showing the full directly-follows graph



(b) Celonis showing the full directly-follows graph



(c) Disco showing the full directly-follows graph



(d) Celonis showing a filtered directly-follows graph

Figure 12: Screenshots from Disco and Celonis to show that one should be careful to interpret results from informal models like a filtered directly-follows graph correctly.

However, formal models may be hard to understand by end-users when things get complex. Formal models may also be incorrect, have soundness problems (e.g., deadlocks), or take a long time to compute. Consider for example the α-algorithm that is guaranteed to discover a particular class of process model.[9] However, when the event log is incomplete or the underlying process is outside the class of models, the α-algorithm often returns a process model that is not sound (e.g., having deadlocks and livelocks) or that does not allow for behavior seen (e.g., skips). In such cases formality is simply misleading. Therefore, it is recommended to always check a formal model using conformance checking. There are several algorithms that guarantee certain properties (e.g., soundness or ability to replay the event

log). However, such models tend to be extremely complex due to overfitting the event log or simple but severely underfitting the event logs (unconstrained behavior). For example, region-based discovery techniques provide models that are guaranteed to be able replay the event log and that may construct very "smart" places constraining behavior to what was seen in the log.[8,14,47,56] However, Petri nets with smartly constructed places and BPMN models with many gateways are quickly perceived as too complex. Also the difference between fitting and non-fitting traces may be less clear cut. These challenges explain why commercial vendors resorted to informal models. Commercial process mining tools also need to be able to handle logs with millions of events and still be used in an interactive manner. Therefore, the challenge is to bridge the gap between the formal approaches described in literature and the more pragmatic approaches used by commercial vendors.

Additional research is needed to better combine the best of two worlds. One approach is to use relaxed versions of the inductive mining techniques.[33,34] The inductive approach ensures a sound model that is able to replay a substantial part of the event log. There are also highly scalable variants of the approach. If needed, the algorithm can be forced to guarantee perfect fitness. However, the resulting process model may be severely underfitting. For unstructured processes, the discovered process trees tend to have many loops combined with activities that can be skipped. As a result almost any behavior is possible.

Another approach is to use *hybrid process models*.[5] These models aim to combine the best of both worlds. Constructs with formal semantics are only used for the parts that are clear-cut. For these definite parts of the process we distinguish between concurrency, choice, and other exactly specified behaviors. For the other parts we resort to arcs showing dependencies that should not be interpreted formally. This way hybrid process models are able to express informal dependencies (like in commercial tools) that are deliberately vague and at the same time provide formal semantics for the parts that, supported by the data, can be expressed in an unambiguous manner. Whenever dependencies are a bit weaker or too complex, then they are not left out, but depicted in an informal manner. Install the *HybridMiner* package from `http://www.promtools.org` to see an example implementation of this idea.[5]

The need for new approaches that better balance between informal models (e.g., just

showing strong directly-follows relations) and formal models (e.g., Petri net places and BPMN gateways) is nicely illustrated by the way that commercial vendors started to support conformance checking. For example, Celonis uses formal BPMN models when checking compliance and filtered directly-follows relations when discovering model from event data. Using different process models for different process mining tasks is far from optimal: A more seamless integration is needed.

A described in this section, there exists a gap between the commercial systems often generating informal models (i.e., models that cannot be used to classify trace into fitting and non-fitting) and the more formal approaches described in literature. The framework depicted in Figure 10 may help to bridge this gap and create awareness for the problems identified in this section.

RELATED WORK

For a more elaborate introduction to the topic of process mining we refer to the book "Process Mining: Data Science in Action" [2] In the current article, unlike the book, we focused on process discovery. The first comprehensive survey on process discovery appeared in 2003. [4] In 2008 another survey by Tiwari, Turner, and Majeed was published. [50] Both surveys do not reflect the current state-of-the-art in process discovery anymore. Many new algorithms appeared in recent years.

There have also been several papers comparing the performance of different algorithms in terms of model quality. [13,51] These papers focus on measuring the performance of algorithms on selected sets of event logs and not on unifying the inner working of different algorithms. To support the selection of process discovery techniques an automated service was proposed by Ribeiro et al. [45] However, the quality of this recommender system depends on portfolio of models and logs used as input. Claes focused on providing an overview of the use of ProM plug-ins. [20] Also more specific surveys appeared such as the recent study on educational process mining. [16]

This article complements existing literature by introducing the topic of process discovery in an original manner. By unifying existing approaches and discussing the abstractions used,

the reader gets a compact, yet insightful, overview of the field. Moreover, Figure 10 and the expression $m_{PM2MA}{}^{-1}(m_{LA2MA}(m_{L2LA}(L)))$ elegantly shows how to get from an event log to a process model through log and model abstractions. Often these notions are not visible and used in an implicit manner.

CONCLUSIONS

Event data are everywhere and process mining techniques help to unlock the value in such data. The four main types of process mining described in this article are (1) process discovery, (2) conformance checking, (3) process reengineering, and (4) operational support. Process mining is not only about control-flow, other perspectives like the time perspective, the data and decision perspective, the resource and organization perspective, etc. can be added. Independent of the perspectives included, process mining techniques can be used in online and offline settings. In this article, we first provided an overview of the process mining spectrum, and then focused on control-flow discovery.

The starting point is an event log. We showed that an event log represents a viewpoint on some event collection. Each event should have a case identifier, an activity name and a timestamp as a bare minimum. We also described different target languages: system nets (i.e., Petri nets with in initial and final state), structured process models (process trees in particular), and declarative models composed of cardinality constraints.

There are many discovery techniques to learn process models from event logs. These techniques often use log and model abstractions. In this article we discussed three types of abstractions: (1) the (weighted) directly-follows graph, (2) the (weighted) transition system, and (3) the (weighted) cardinality constraint model. Although these abstractions are very different, we showed that they can be captured in a common framework (Figure 10).

Despite the many successful applications of process mining and the availability of over 25 commercial products supporting process mining (Celonis, Disco, Minit, myInvenio, Process-Gold, QPR, etc.), there are still many open challenges. Discovery techniques need to balance different goals (e.g., fitness, simplicity, precision, and generalization) and at the same time provide guarantees (e.g., able to rediscover a behaviorally equivalent process model).

We expect that the framework presented in this article will trigger novel ideas for new discovery techniques. Existing approaches tend to have tight coupling between the abstractions used and the target modeling language. *Decoupling the log-based and model-based abstractions from the process modeling notation stimulates a cross-fertilization among existing approaches.* For example, cardinality constraint models could be used to learn process trees or Petri nets. Another direction for future research is to include formal and informal elements in process models resulting in so-called hybrid process models.[5] The log-based abstraction may hold information that cannot be mapped onto modeling constructs as these make binary decisions on included and excluded behavior.

References

1. W.M.P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, pages 1–37, 2013. doi:10.1155/2013/507984.

2. W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, Berlin, 2016.

3. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

4. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

5. W.M.P. van der Aalst, R. De Masellis, C. Di Francescomarino, and C. Ghidini. Learning Hybrid Process Models From Events: Process Discovery Without Faking Confidence. In J. Carmona, G. Engels, and A. Kumar, editors, *International Conference on Business Process Management (BPM 2017)*, volume 10445 of *Lecture Notes in Computer Science*, pages 59–76. Springer-Verlag, Berlin, 2017.

6. W.M.P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Nunez, and G. Zavattaro, editors, *International Conference on Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag, Berlin, 2006.

7. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

8. W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

9. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

10. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998.

11. E. Alpaydin. *Introduction to Machine Learning*. MIT press, Cambridge, MA, 2010.

12. D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.

13. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, and A. Soo. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *CoRR*, abs/1705.02288, 2017.

14. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.

15. A.W. Biermann and J.A. Feldman. On the Synthesis of Finite-State Machines from Samples of their Behavior. *IEEE Transaction on Computers*, 21:592–597, 1972.

16. A. Bogarin, R. Cerezo, and C. Romero. A survey on educational process mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2017.

17. J. Carmona, J. Cortadella, and M. Kishinevsky. New Region-Based Algorithms for Deriving Bounded Petri Nets. *IEEE Transactions on Computers*, 59(3):371–384, 2010.

18. 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.

19. C. Di Ciccio and M. Mecella. A Two-Step Fast Algorithm for the Automated Discovery of Declarative Workflows. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013)*, pages 135–142. IEEE Computer Society, 2013.

20. J. Claes and G. Poels. *Process Mining and the ProM Framework: An Exploratory Survey*, pages 187–198. Springer-Verlag, Berlin, Berlin, Heidelberg, 2013.

21. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

22. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.

23. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.

24. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer-Verlag, Berlin, 2013.

25. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.

26. E.M. Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.

27. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.

28. D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT press, Cambridge, MA, 2001.

29. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.

30. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In J.M. Colom and J. Desel, editors, *Applications and Theory of Petri Nets 2013*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer-Verlag, Berlin, 2013.

31. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In N. Lohmann, M. Song, and P. Wohed, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013)*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78. Springer-Verlag, Berlin, 2014.

32. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-structured Process Models from Incomplete Event Logs. In G. Ciardo and E. Kindler, editors, *Applications and Theory of Petri Nets 2014*, volume 8489 of *Lecture Notes in Computer Science*, pages 91–110. Springer-Verlag, Berlin, 2014.

33. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Scalable Process Discovery with Guarantees. In K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, editors, *En-*

terprise, Business-Process and Information Systems Modeling (BPMDS 2015), volume 214 of *Lecture Notes in Business Information Processing*, pages 85–101. Springer-Verlag, Berlin, 2015.

34. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Scalable Process Discovery and Conformance Checking. *Software and Systems Modeling (Online First)*, pages 1–33, 2016.

35. G. Li, R. Medeiros de Carvalho, and W.M.P. van der Aalst. Automatic Discovery of Object-Centric Behavioral Constraint Models. In W. Abramowicz, editor, *Business Information Systems (BIS 2017)*, volume 288 of *Lecture Notes in Business Information Processing*, pages 43–58. Springer-Verlag, Berlin, 2017.

36. F.M. Maggi, R.P. Jagadeesh Chandra Bose, and W.M.P. van der Aalst. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In J. Ralyte, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *International Conference on Advanced Information Systems Engineering (Caise 2012)*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer-Verlag, Berlin, 2012.

37. F.M. Maggi, R.P. Jagadeesh Chandra Bose, and W.M.P. van der Aalst. A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps. In C. Salinesi, M.C. Norrie, and O. Pastor, editors, *International Conference on Advanced Information Systems Engineering (Caise 2013)*, volume 7908 of *Lecture Notes in Computer Science*, pages 433–448. Springer-Verlag, Berlin, 2013.

38. F.M. Maggi, M. Montali, M. Westergaard, and W.M.P. van der Aalst. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer-Verlag, Berlin, 2011.

39. F. Mannhardt, M. de Leoni, and H.A. Reijers. Heuristic Mining Revamped: An Interactive, Data-aware, and Conformance-aware Miner. In *Proceedings of the BPM 2017 Demo Track*, pages 1–5, 2017.

40. H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

41. A.K. Alves de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow Mining: Current Status and Future Directions. In R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer-Verlag, Berlin, 2003.

42. M. Montali. *Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2010.

43. M. Montali, F.M. Maggi, F. Chesani, P. Mello, and W.M.P. van der Aalst. Monitoring Business Constraints with the Event Calculus. *ACM Transactions on Intelligent Systems and Technology (AMC TIST)*, 5(1):17:1–17:30, 2013.

44. A. Nerode. Linear Automaton Transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

45. J. Ribeiro, J. Carmona, M. Mısır, and M. Sebag. A Recommender System for Process Discovery. In S. Sadiq, P. Soffer, and H. Voelzer, editors, *International Conference on Business Process Management (BPM 2014)*, volume 8659 of *Lecture Notes in Computer Science*, pages 67–83. Springer-Verlag, Berlin, 2014.

46. A. Rozinat. *Process Mining: Conformance and Extension*. Phd thesis, Eindhoven University of Technology, November 2010.

47. M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.

48. R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalization and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT 96)*, pages 3–17, 1996.

49. TFPM. Process Mining Case Studies, http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies, 2017.

50. A. Tiwari, C.J. Turner, and B. Majeed. A Review of Business Process Mining: State-of-the-Art and Future Trends. *Business Process Management Journal*, 14(1):5–22, 2008.

51. J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A Multi-Dimensional Quality Assessment of State-of-the-Art Process Discovery Algorithms Using Real-Life Event Logs. *Information Systems*, 37(7):654–676, 2012.

52. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

53. A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). BETA Working Paper Series, WP 334, Eindhoven University of Technology, Eindhoven, 2010.

54. L. Wen, W.M.P. van der Aalst, J. Wang, and J. Sun. Mining Process Models with Non-Free-Choice Constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

55. L. Wen, J. Wang, W.M.P. van der Aalst, B. Huang, and J. Sun. A Novel Approach for Process Mining Based on Event Types. *Journal of Intelligent Information Systems*, 32(2):163–190, 2009.

56. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.

57. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.

FURTHER READING

For more information on process mining the reader should read the book "Process Mining: Data Science in Action" [2] or follow the free Massive Open Online Course (MOOC) with the same title: www.coursera.org/course/procmin. Also the website www.processmining.org provides further information and pointers.