Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs

Marlon Dumas(✉) and Luciano García-Bañuelos

University of Tartu, Tartu, Estonia
marlon.dumas@ut.ee

Abstract. Process mining is a family of methods to analyze event logs produced during the execution of business processes in order to extract insights regarding their performance and conformance with respect to normative or expected behavior. The landscape of process mining methods and use cases has expanded considerably in the past decade. However, the field has evolved in a rather ad hoc manner without a unifying foundational theory that would allow algorithms and theoretical results developed for one process mining problem to be reused when addressing other related problems. In this paper we advocate a foundational approach to process mining based on a well-known model of concurrency, namely event structures. We outline how event structures can serve as a unified representation of behavior captured in process models and behavior captured in event logs. We then sketch how process mining operations, specifically automated process discovery, conformance checking and deviance mining, can be recast as operations on event structures.

1 Introduction

Process mining [1,2] is a family of methods concerned with the analysis of event records produced during the execution of business processes. Process mining methods allow analysts to understand how a given process is executed on a day-to-day basis and to detect and analyze deviations with respect to performance objectives or normative pathways. Process mining has gained significant practical adoption in recent years, as evidenced by a growing number of case studies and commercial tools. An overview of methods, tools and case studies in this field is maintained by the IEEE Task Force on Process Mining.[1]

The main input of a process mining method is a *business process event log*, that is, a collection of event records relevant to a given business process. An event log is generally structured as a set of traces. Each *trace* consists of the sequence of events produced by one execution of the process (a.k.a. a *case*). An event in a trace denotes the start, end, abortion or other relevant state change of the process or an activity therein.

[1] http://www.win.tue.nl/ieeetfpm

© Springer International Publishing Switzerland 2015
R. Devillers and A. Valmari (Eds.): PETRI NETS 2015, LNCS 9115, pp. 33–48, 2015.
DOI: 10.1007/978-3-319-19488-2_2

Most typically, event logs used in the context of process mining consist of events that signal the start or the end of each activity of the process. As a minimum, an event record contains an identifier of the case of the process to which the event refers, a timestamp and an *event class*, that is, a reference to an activity in the process under observation. Each event in a trace may additionally carry a payload consisting of attributes such as the resource(s) involved in the execution of an activity or other data recorded alongside the event – for example if an event represents the creation of a loan application, possible attributes include the name of the applicant and the amount of the requested loan.

A simplified example of a log of a loan application process is sketched in Table 1. In this table, CID stands for "Customer Identifier" and constitutes the primary key that should be used to group the records in the log into traces.

Table 1. Extract of a loan application log

CID	Event Type	Timestamp	...
13219	Enter Loan Application	2007-11-09 11:20:10	...
13219	Retrieve Applicant Data	2007-11-09 11:22:15	...
13220	Enter Loan Application	2007-11-09 11:22:40	...
13219	Compute Installments	2007-11-09 11:22:45	...
13219	Notify Eligibility	2007-11-09 11:23:00	...
13219	Approve Simple Application	2007-11-09 11:24:30	...
13220	Compute Installments	2007-11-09 11:24:35	...
...	...	...	...

The output of process mining can be manifold, ranging from a model of the process, to a summarized view of the most frequent paths of the process or a description of the deviations of the process with respect to normative or expected behavior.

Process mining has been an active field of research for over a decade [1,2]. During this time, a number of process mining operations have been extensively studied. One widely studied operation is *automated process discovery*. This operation takes as input a log L and produces a process model M that is "likely" to have generated the traces in log L. For example, given the log in Table 1, the output of an automated process discovery method could be the process model shown in Figure 1, which uses the standard Business Process Model and Notation (BPMN) [3].

Another widely researched operation is conformance checking, which given a model and a log, produces an enumeration of their differences, that is, a description of the behavior observed in the log but not in the model, as well as behavior allowed by the model but not observed in the log. Related to the latter is *model repair*, where instead of simply enumerating the differences between a model M and a log L, the goal is to produce a process model M' that is "similar" to M and can parse every trace in the log.

Until now, process mining methods have been developed on a case-by-case basis using disparate approaches and representations. Attempts have been made

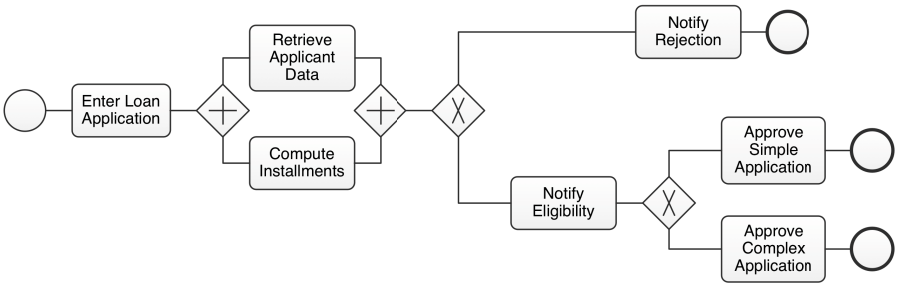


Fig. 1. Process model corresponding to the log extract in Table 1

at identifying a small number of primitive operations from which other operations can be defined [2] – but each of these primitives has been formally defined and developed independently, without an underpinning foundational theory.

In this paper we advocate a foundational approach to process mining based on a well-known model of concurrency, namely event structures [4]. We show that event structures can be used to represent both models and logs in a unified manner and that this unified representation can serve to define process mining operations that have until now been approached using different formalisms.

2 Overview of Process Mining Operations

Following a well-accepted classification in the broader field of data mining, process mining methods can be broadly classified into offline methods and online methods. Offline process mining methods aim at providing insights about the process as it is or as it has been observed in the logs. Online process mining methods on the other hand aim at providing insights about currently running cases of a process, like for example predicting the completion time of an ongoing case, or predicting whether an ongoing case will lead to a positive or a negative outcome [5]. For the sake of scoping the discussion, we hereby focus on offline process mining methods. Online process mining methods come with strong non-functional requirements, particularly with respect to performance and scalability, that deserve a separate treatment.

From the literature on process mining, one can distill the following broad classes of offline process mining operations: (i) automated process discovery and model enhancement; (ii) conformance checking and model repair; and (iii) deviance mining. Other classes of process mining operations include concept drift analysis [6] and variant identification [7], but we shall herein concentrate on the former three classes of operations for the sake of illustration.

Automated process discovery is a family of methods that given a log L generate a model M that "approximates" log L. A range of algorithms for automated process discovery have been proposed over the past decade, based on various representations of behavior. For example, the α-algorithm [8] starts by inferring

a matrix of behavioral relations between pairs of event classes in the log, specifically *direct follows*, *causality*, *conflict* and *concurrency* relations. Given a matrix capturing all such relations, the algorithm constructs a Petri net by applying a set of rules. Similarly, the heuristics miner [9] relies on behavioral relations between all pairs of event classes found in the log, but additionally takes into account the relative frequency of the *direct follows* relation between pairs of tasks. These data are used to construct a graph of events (called a Heuristics net), where edges are added based on a number of heuristics. The Heuristics net can then be converted into a Petri net or a BPMN model for example. Van der Werf et al. [10] propose a process model discovery method where behavioral relations observed in the logs are translated to an Integer Linear Programming (ILP) problem, while Carmona et al. [11] approach the problem of automated process discovery using theory of regions. Finally, the InductiveMiner [12] discovers a tree where each internal node represents a block-structured control-flow construct (e.g. block-structured parallelism or block-structured choice). Such trees can then be trivially transformed into a block-structured Petri net for example.

Automated process discovery methods can be evaluated along four dimensions: fitness, precision, generalization and complexity [1,13]. Fitness measures to what extent the traces in a log can be parsed by a model. Precision measures the additional behavior allowed by a discovered model not found in the log. A model with low precision is one that parses a proportionally large number of traces that are not in the log. Generalization captures how well the discovered model generalizes the behavior found in the log. For example, if a model discovered using 90% of traces in the log can parse all or most of the remaining 10% of traces in the log, it is said the model generalizes well the log. Finally, process model complexity is intended as a proxy for understandability. It can be measured in terms of size (number of nodes and/or edges) or using a number of structural complexity metrics such as cyclomatic complexity or density that have been empirically shown to be correlated with understandability and error-proneness [14].

Related to automated process discovery is a family of methods known as *process model enhancement* [15], which given a model M and a log L (such that L is likely to have been produced by M) generate an annotated process model M'. A sub-family of model enhancement methods produce annotations referring to performance measures such as waiting times and processing times for each activity and branching probabilities for each branch of a decision point. Another family of model enhancement methods produce annotations related to resources: who performs which activity and who hands-over work to whom? In any case, from an algorithmic perspective, process model enhancement generally does not bring additional challenges with respect to automated process discovery.

Conformance checking is concerned with describing how and where actual process executions (recorded in an event log) deviate with respect to a given process model. This problem has been approached using replay techniques [15] and trace alignment techniques [16]. Replay takes as input one trace at a time and determines what maximal prefix of the trace (if any) can be parsed by the model.

When it is found that a prefix can no longer be parsed by the model, error-recovery techniques are used to correct the parsing error and to continue parsing as much as possible the remaining input trace. Alignment-based techniques on the other hand seek to find for each trace in the log, the closest corresponding trace(s) produced by the model and to determine where exactly in these traces the model and the log diverge.

Closely related to the problem of conformance checking is that of *model repair* [17], where instead of simply enumerating the differences between a model M and a log L, the goal is to generate a process model M' that is similar to M and can parse all the traces in the log. Model repair can be seen as a generative counter-part of conformance checking.

Deviance mining [18] is a family of process mining methods that aim at detecting and explaining differences between executions of a business process that lead to a positive outcome vs. those that lead to a negative outcome – with respect to a given labeling of cases into positive vs. negative ones. For example, one specific deviance mining problem is that of explaining the differences between executions of a process that fulfill a given service-level objective vs. those that do not.

Existing approaches to deviance mining can be classified into two categories [18]: *model delta analysis* [19,20] and *sequence classification*. The idea of model delta analysis is to apply automated process discovery methods to the traces of positive cases and to the traces of negative cases separately. The discovered process models are then visually compared in order to identify distinguishing patterns. This approach however does not scale up to large and complex logs. Sequence classification methods [20–22] construct a classifier (e.g. a decision tree) that can determine with sufficient accuracy whether a given trace belongs to the positive or the negative class. The crux of these methods is how sequences are encoded as feature vectors for classifier learning. Several sequence mining techniques have been explored for feature extraction in this setting. These techniques generally extract patterns of the form activity A occurs before activity B, which are frequent in (e.g.) positive cases but not in negative ones or vice-versa. An evaluation of these techniques on real-life logs has shown however that their explanatory power is rather limited [18], meaning that dozens or hundreds of rules are required to explain the differences between positive and negative cases.

Observations The above overview illustrates that various process mining problems have been approached from different angles and using disparate representations and approaches. Automated process discovery for example has been approached using representations based on binary relations between event classes as well as tree-based representations of (block-structured) process models. Meanwhile, conformance checking has been approached using replay (parsing) as well as trace alignment – techniques that are in essence disconnected from those used for automated process discovery. On the other hand, deviance mining has been approached using sequence mining techniques, which reason in terms of sequences and patterns on sequences. Again, these techniques are disconnected

from the previous ones. As a further case in point, the problem of concept drift analysis [6] – where the goal is to detect and explain how the behavior of a given process has evolved over time – has been approached using sequence patterns as well as abstractions of sets of traces based on polyhedra [23].

Underpinning these observations is the fact that the representations used to reason about the behavior captured in process models are different from those used to reason about the behavior captured in event logs. When reasoning on process models, representations based on behavioral relations or Petri nets tend to be favored. When reasoning from the perspective of logs, representations based on sequences are often preferred. Below we advocate for a unified representation of process models and event logs that provides a unified perspective into existing process mining problems and methods.

3 Event Structures as a Foundation for Process Mining

We contend that event structures [4] – a well-known model of concurrency – can serve as a common representation of process models and event logs for the purpose of defining and implementing process mining operations. Below we provide a brief overview of event structures and their relation with process models and event logs. We then sketch how conformance checking and deviance mining can be recast as problems of comparison of event structures, and we briefly discuss how automated process discovery could be tackled under this framework.

3.1 Event Structures

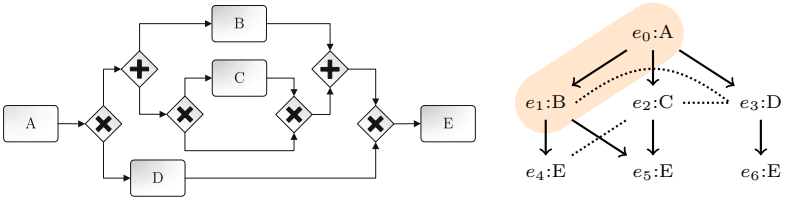
A Prime Event Structure (PES) [4] is a graph of events, where an event e represents the occurrence of an action (e.g. a task) in the modeled system (e.g. a business process). If a task occurs multiple times in a run, each occurrence is represented by a different event. The order of occurrence of events is defined via binary relations: i) *Causality* ($e < e'$) indicates that event e is a prerequisite for e'; ii) *Conflict* ($e \# e'$) implies that e and e' cannot occur in the same run; iii) *Concurrency* ($e \parallel e'$) indicates that no order can be established between e and e'.

Definition 1 (Labeled Prime Event Structure [4]). *A* Labeled Prime Event Structure *over the set of event labels $\mathcal{L}$ is the tuple $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$ where*

- *E is a set of events (e.g. tasks occurrences),*
- *$\leq \subseteq E \times E$ is a partial order, referred to as* causality*,*
- *$\# \subseteq E \times E$ is an irreflexive, symmetric* conflict *relation,*
- *$\lambda : E \rightarrow \mathcal{L}$ is a labeling function.*

We use $<$ to denote the irreflexive causality relation. The concurrency relation *of $\mathcal{E}$ is defined as $\parallel = E^2 \setminus (< \cup <^{-1} \cup \#)$. Moreover, the* conflict relation *satisfies the principle of conflict heredity, i.e. $e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$ for $e, e', e'' \in E$.*

For illustration, Fig. 2 presents side-by-side a BPMN process model and a corresponding PES $\mathcal{E}^1$. Nodes are labelled by an event identifier followed by the



(a) BPMN (b) Prime event structure $\mathcal{E}^1$

Fig. 2. Sample BPMN process model and corresponding PES

label of the represented task, e.g. "e_2:C" tells us that event e_2 represents an occurrence of task "C". The causality relation is depicted by solid arcs whereas the conflict relation is depicted by dotted edges. For the sake of simplicity, transitive causal and hereditary conflict relations are not depicted. Every pair of events that are neither directly nor transitively connected are in a concurrency relation. Note that three different events refer to the task with label "E". This duplication is required to distinguish the different states where task "E" occurs.

A state on an event structure (hereby called a *configuration*) is characterized by the set of events that have occurred so far. For instance, set $\{e_0{:}\mathrm{A}, e_1{:}\mathrm{B}\}$ – highlighted in Fig. 2(b) – is the configuration where tasks "A" and "B" have occurred. In this configuration, event $\{e_3{:}\mathrm{D}\}$ can no longer occur because it is in conflict with $\{e_1{:}\mathrm{B}\}$. On the other hand, events $\{e_2{:}\mathrm{C}\}$ and $\{e_4{:}\mathrm{E}\}$ can occur, but the occurrence of one precludes that of the other. Formally:

Definition 2 (Configuration). *Let $\mathcal{E} = \langle E, \leq, \#, \lambda \rangle$ be a prime event structure. A* configuration *of $\mathcal{E}$ is the set of events $C \subseteq E$ such that*

- *C is causally closed, i.e. $\forall e' \in E, e \in C : e' \leq e \Rightarrow e' \in C$, and*
- *C is conflict-free, i.e. $\forall e, e' \in C \Rightarrow \neg(e \# e')$.*

The local configuration *of an event $e \in E$ is the set $\lfloor e \rfloor = \{e' \mid e' \leq e\}$. Similarly, the* (set of) strict causes *of an event $e \in E$ is defined as $\lfloor e) = \lfloor e \rfloor \setminus \{e\}$.*

We denote by $Conf(\mathcal{E})$ the set of all possible configurations of $\mathcal{E}$ and by $MaxConf(\mathcal{E})$ the subset of maximal configurations with respect to set inclusion. In the running example, $MaxConf(\mathcal{E}^1) = \{\{e_0, e_1, e_2, e_5\}, \{e_0, e_1, e_4\}, \{e_0, e_3, e_6\}\}$.

Prime event structures can be extracted from Petri nets using well-known unfolding techniques. In the case of acyclic nets, a full unfolding can be computed and a PES can be trivially derived therefrom. In the case of bounded Petri nets with cycles, it is possible to calculate a finite *prefix unfolding* that captures all the behavior in the original net. A PES can then be derived from such prefix unfolding. Several prefix unfoldings have been defined in the literature, such as the *complete prefix unfolding* [24]. In [25] we defined a type of unfolding that additionally captures all the causes of every event – including events inside a cycle – thus allowing us to pinpoint which events are repeated and which are not.

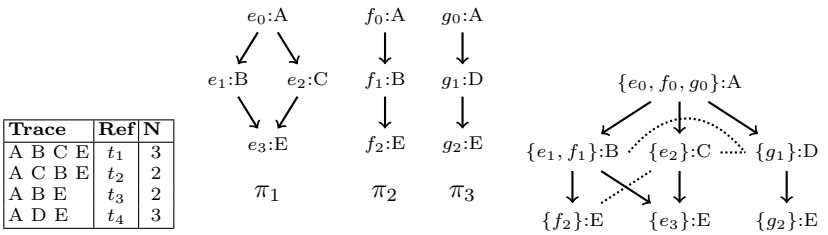
This information allows us to do more fine-grained reasoning on the repetitive behavior of a process compared to a complete prefix unfolding.

3.2 From Logs to Event Structures

In previous work [26], we presented a method to generate a PES from an event log. The method consists of two steps. First the event log, seen as a set of traces, is transformed into a set of runs by invoking a *concurrency oracle*. In essence, each trace is turned into a run by relaxing the total order induced by the trace into a partial order such that two events are not causally related if the concurrency oracle has determined that they occur concurrently. The concurrency oracle is left open. Existing concurrency oracles such as those proposed in the α process mining algorithm [8] or in [27] can be used for this purpose.

Second, the set of runs are merged into an event structure in a lossless manner, meaning that the set of maximal configurations of the resulting event structure is exactly equal to the set of runs. In this way and modulo the accuracy of the concurrency oracle, we ensure that the resulting event structure is a lossless representation of the input log.

For example, consider the log given in Figure 3(a). This event log consists of 10 traces, including 3 instances of distinct trace t_1 (as specified in column "N"), 2 instances of t_2, so on and so forth. Using the concurrency oracle of the α algorithm we conclude that event classes B and C are in a concurrency relation, thus we construct the set of runs in Figure 3(b). In this latter figure, the notation e:A indicates that event e represents an occurrence of event class A in the original log. By merging together events with the same label and the same history (i.e. same prefix), we obtain the PES in Figure 3(c). In this figure, the notation $\{e_1, e_2 \dots e_i\}$:A indicates that events $\{e_1, e_2 \dots e_i\}$ represent occurrences of event class A in different runs.



(a) Event log (b) Runs (c) Induced PES

Fig. 3. Example of construction of a PES from a set of traces

3.3 Comparison of Event Structures

In previous work [25], we presented a technique for comparing pairs of event structures. This technique operates by performing a *Partially Synchronized Product* (PSP) of the event structures, which is in essence a synchronized simulation starting from the empty configurations. At each step, the events that can occur given the current configuration in each of the two event structures (i.e. the *enabled* events) are compared. If they match, the simulation adds those events to the current configurations and continues. If on the other hand an enabled event in the current configuration of one event structure does not match with an enabled event in the current configuration in the other event structure, a mismatch is declared and this mismatch will be reflected in a *difference statement* that tells us that there is a pair of matching configurations where an event can occur or a behavioral relation holds in one event structure, but not in the other. Having diagnosed the difference, the unmatched event is "hidden" and the simulation jumps to the next matching configurations.

Figure 5 presents an excerpt of the PSP for $\mathcal{E}^1$ and $\mathcal{E}^2$, shown in Figure 2 and Figure 4 respectively. Note that $MaxConf(\mathcal{E}^2) = \{\{f_0, f_1, f_2, f_4\}, \{f_0, f_3, f_5\}\}$. Clearly, all maximal configurations of $\mathcal{E}^2$ can be matched to configurations of $\mathcal{E}^1$. The right-hand leaf node in the PSP illustrates the matching of configuration $\{e_0, e_1, e_2, e_5\}$ from $\mathcal{E}^1$ and $\{f_0, f_1, f_2, f_4\}$ from $\mathcal{E}^2$. There, the set m records the fact that all the events in both configuration have been matched, lh records that none of the events from $\mathcal{E}^1$ (the one to the left of the "product") has been hidden, and rh records that no event from $\mathcal{E}^2$ has been hidden. Similarly, the leaf node at the left-hand side corresponds to the best matching of configurations $\{e_0, e_1, e_4\}$ and $\{f_0, f_1, f_2, f_4\}$, respectively from $\mathcal{E}^1$ and $\mathcal{E}^2$. The cloud at the top indicates that some states precedes to the matching of a pair of events sharing the label "B". The label on the edge from the cloud to the node just below records such matching. The configuration $\{e_0, e_1\}$ enables the occurrence of e_4:E but that occurrence precludes the occurrence of e_2:C. This gives rise to a behavioral mismatch, that is resolved by hiding f_2:C. The red arrow in the PSP captures this hiding: the event f_2:C from $\mathcal{E}^2$ (right-hand side model in the product) is hidden. Note that in the target box, m remains the same, i.e. no additional matching, whereas rh records now the hiding of f_2:C. By aggregating the information in the states and edges associated to the moves "rhide C" and "match C" on the PSP, it is possible to diagnose that "Task 'C' in model 1 can be skipped, whereas the same task is always executed in model 2".

Further details on the event structures comparison method are given in [25]. A tool implementing this method, namely BP-Diff, is described in [28]. BPDiff takes as input two process models captured in standard BPMN notation and outputs a number of statements describing their behavioral differences. Each difference is verbalized in natural language and can also be visually represented on top of the process model. The tool performs the comparison at the level of event structures. Prior to the comparison step, BP-Diff converts the input BPMN process model into a Petri net and unfolds the latter into an event structure.

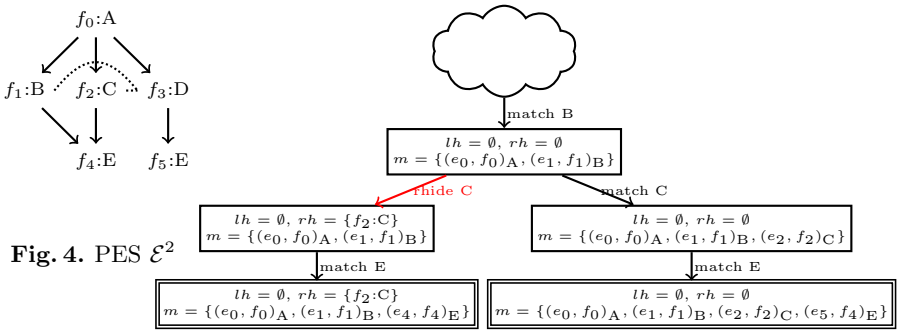


Fig. 4. PES $\mathcal{E}^2$

Fig. 5. Fragment of PSP of the PESs in Figs. 2(b) and 3

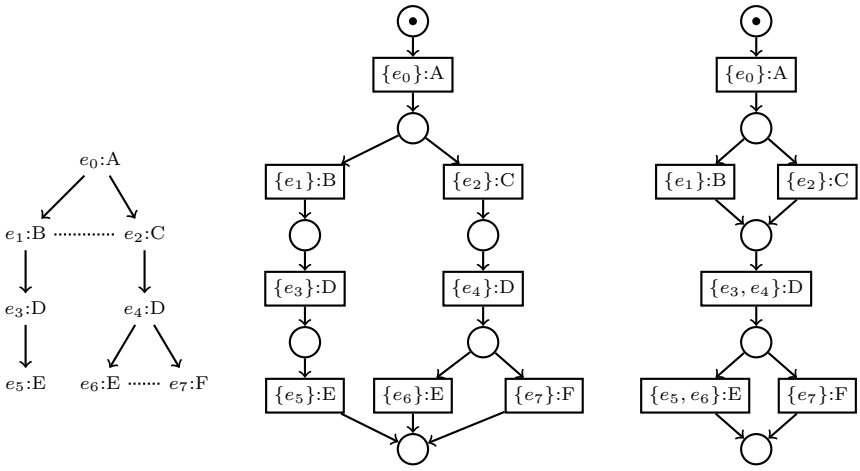
3.4 Folding of Event Structures

The PES derived from a given Petri net is generally not a space-efficient representation of behavior as it may contain significant amount of duplication (several events referring to the same task). A more compact representation can be obtained by using asymmetric event structures [29], which replace the symmetric conflict relation of PES with an asymmetric one. AES lend themselves to applying folding techniques. The idea of such folding techniques is to identify sets of events in the AES that can be merged into a single event while preserving behavior. This folding of events can be performed when two events with the same label are future-equivalent, meaning they have the same possible continuations. Under such conditions, such events can be merged into a single one thanks to the asymmetric conflict relation. The resulting folded AES is more compact and thus more convenient for the purpose of comparing pairs of process models, insofar as their comparison produces less statements of differences. In [25] we discuss how to derive a canonically folded AES from a PES, which can be used to provide a more compact diagnosis of the differences between two given PES.

We foresee that similar folding techniques can be used more widely to simplify an event structure produced from a given event log, such that the simplified event structure can be used to synthesize a process model. By allowing events to be folded even in situations where some behavior is lost or added, we can strike different tradeoffs between the four quality dimensions of discovered process models mentioned in Section 2 (precision, recall, generalization and complexity). A similar idea has been applied in [30] in order to simplify process models generated by existing automated process discovery methods.

To illustrate how folding can be used to simplify event structures at the expense of precision, consider the event structure $\mathcal{E}$ in Figure 6(a). A Petri net synthesized from this net is shown in Figure 6(b). We note that events e_3:D and e_4:D refer to the same task D. Furthermore, the set of possible futures of e_3:D is included in that of e_4:D – the latter having an additional possible future consisting of an occurrence of F. If we define a rule that folds two events under such conditions, we would fold e_5:E and e_6:E into $\{e_5, e_6\}$:E (with an empty

future) and then we would fold e_3:D and e_4:D into $\{e_3, e_4\}$:D – with $\{e_5, e_6\}$:E and e_7:F as its futures. The resulting event structure then leads to the simpler net in Figure 6(c), which has more behavior (i.e. generalizes) the net in Figure 6(b).



(a) Event structure $\mathcal{E}$ (b) Net synthesized from $\mathcal{E}$ without folding (c) Net synthesized from $\mathcal{E}$ after folding

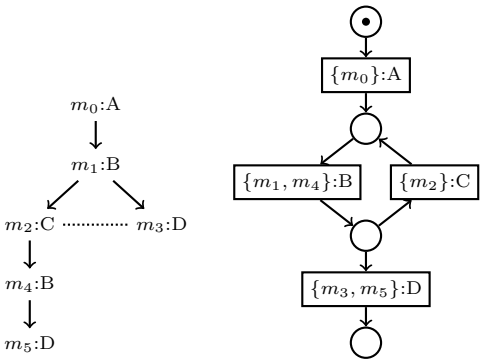
Fig. 6. Nets synthesized from an unfolded and a folded event structure with added behavior

We also foresee that a similar approach can serve to identify opportunities to introduce cycles when synthesizing a Petri net from an event structure. Consider for example the event structure in Figure 7(a). Event m_1:B and m_4:B share a common future consisting of an occurrence of D – with m_1:B having a an additional future consisting of occurrences of C, B and D. Under these conditions, one could generalize the behavior by allowing $m1 : B$ and $m3 : B$ to be folded, so that the net in 7 can be synthesized thereon.

3.5 Process Mining Operations and Event Structures

To recap, we have observed that:

- Event structures can be losslessly derived both business process models via unfoldings of Petri nets
- Event structured can be losslessly derived from event logs via concurrency oracles and merging of runs.
- Event structures allow fine-grained comparison of behavior, which can be materialized as difference statements in natural language or graphical form.



(a) PES with repeated behavior (b) Net generalizing the repeated behavior

Fig. 7. Generalization of repeated behavior as a cycle during net synthesis

– Event structures can be folded (and thus simplified) by merging occurrences of events with the same label and "equivalent futures" – with or without loss of behavioral precision depending on the choice of future equivalence. This folding can be used to trade-off behavioral precision and simplicity.

These operations on event structures can be used to recast the previously reviewed process mining problems as follows:

– Deviance mining can be achieved by computing an event structure from each of the sub-logs induced by the labeling function (e.g. the log of "positive" cases and the log of "negative" cases) and comparing the two logs. In other words, this is a log-to-log comparison problem. In [26] we have empirically shown that the difference diagnostics produced in this way is more compact (less and simpler statements) than deviance diagnostics obtained using sequence classification techniques.
– Conformance checking an event log against a process model can be achieved by computing an event structure from the model, another from the log and comparing the resulting event structures to enumerate their differences. In other words, conformance checking is a model-to-log comparison problem.
– Automated process discovery of a process model from an event log can be achieved by: (i) computing an event structure from a log; (ii) transforming the resulting event structure via folding rules that achieve a trade-off between simplicity and behavioral accuracy (measured by means of fitness and precision); and (iii) utilizing the information in the resulting event structure in order to supplement a model synthesis algorithm.

Similarly although not covered in this paper, business process drift analysis can also be recast as a log-to-log comparison problem: comparing the log

before and after a hypothesized change point. Meanwhile, model repair can be approached as a problem of repairing an event structure, by adding and deleting a minimal amount of relations and events in such a way that the new event structure is an accurate (or more accurate) representation of the runs in the log than the original model.

The above relations between operations on event structures and process mining operations are summarized in Figure 8.

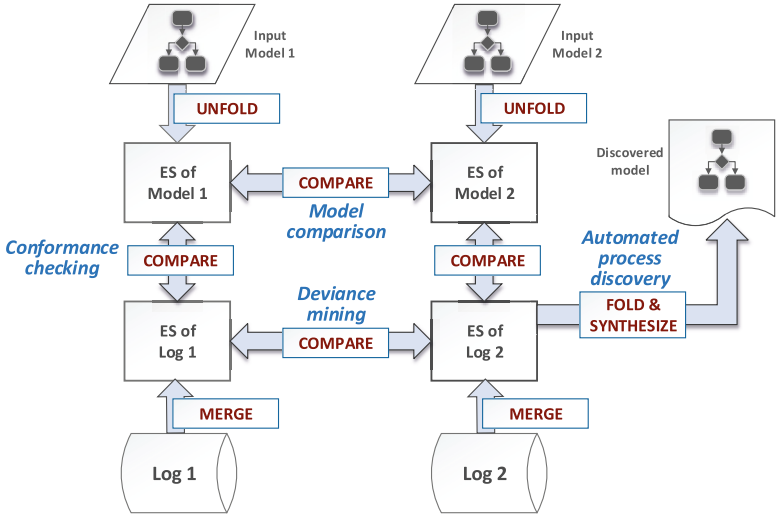


Fig. 8. Process mining operations recast as operations on Event Structures (ES)

4 Outlook

We have outlined a vision for a principled approach to process mining based on event structures as a unified representation of process models and event logs. The realization of this vision however requires a number of challenges to be addressed.

A key challenge is handling repeated behavior. Complete prefix unfoldings allow us to fully capture the behavior of a cyclic (bounded) process model. In turn, the information in this prefix unfolding can be directly encoded in an event structure. However, comparing event structures obtained from process models with those obtained from event logs is challenging because in the event structure derived from a log repeated behavior is not explicitly captured: It exists by virtue of a sub-event structure appearing multiple times in the event structure. Similarly, synthesizing a process model from the event structure derived from a log requires being able to detect and isolate repeated behavior.

Regarding automated process discovery, defining the right folding rules to simplify the event structure produced from the log is a crucial step. These folding

rules will have to trade off simplicity versus accuracy – a tradeoff that existing process mining algorithms try to strike as well. In this respect, event structures merely provide us a way of studying such tradeoffs from a new perspective. Also, the use of event structures for process model synthesis would require new techniques to be developed or existing ones to be heavily adapted, e.g. adapting existing synthesis techniques based on merging of runs [31].

Finally, scalability might become a challenge, not so much in the log-to-log comparison case [26], but rather when cycle detection and folding operations come into play.

In summary, the proposed vision offers numerous opportunities to revisit long-standing challenges in the field of process mining from an angle that will hopefully allow algorithms and theoretical results to be reused across different problems and use cases.

Acknowledgments. The vision presented in this paper owes a lot to our discussions with Abel Armas-Cervantes, Nick van Beest, Dirk Fahland and Marcello La Rosa. Some of the definitions and examples used in this paper are taken from our joint previous work. This research is partly supported by the Estonian Research Council via the Institutional Grants scheme and ERDF via the Estonian Centre of Excellence in Computer Science.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011 Workshops, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)
3. Object Management Group: Business Process Model and Notation (BPMN) Version 2.0. Technical report, Object Management Group Final Adopted Specification (2011). http://www.omg.org/spec/BPMN/2.0/
4. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. Theoretical Computer Science **13**, 85–108 (1981)
5. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Heidelberg (2014)
6. Bose, R.J.C., van der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Transactions on Neural Networks and Learning Systems **25**(1), 154–171 (2014)
7. Folino, F., Greco, G., Guzzo, A., Pontieri, L.: Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. Data Knowl. Eng. **70**(12), 1005–1029 (2011)
8. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9), 1128–1142 (2004)
9. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM, pp. 310–317. IEEE (2011)

10. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inform. **94**(3–4), 387–412 (2009)
11. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded petri nets. IEEE Trans. Computers **59**(3), 371–384 (2010)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013)
13. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Inf. Syst. **37**(7), 654–676 (2012)
14. Reijers, H., Mendling, J.: A study into the factors that influence the understandability of business process models. IEEE T. Syst. Man Cy. A **41**(3), 449–462 (2011)
15. Rozinat, A.: Process Mining Conformance and Extension. PhD thesis, Technische Universiteit Eindhoven (2010)
16. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: EDOC, pp. 55–64. IEEE (2011)
17. Fahland, D., van der Aalst, W.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
18. Nguyen, H., Dumas, M., La Rosa, M., Maggi, F.M., Suriadi, S.: Mining business process deviance: a quest for accuracy. In: Meersman, R., Panetto, H., Dillon, T., Missikoff, M., Liu, L., Pastor, O., Cuzzocrea, A., Sellis, T. (eds.) OTM 2014. LNCS, vol. 8841, pp. 436–445. Springer, Heidelberg (2014)
19. Suriadi, S., Wynn, M.T., Ouyang, C., ter Hofstede, A.H.M., van Dijk, N.J.: Understanding process behaviours in a large insurance company in australia: a case study. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 449–464. Springer, Heidelberg (2013)
20. Lakshmanan, G.T., Rozsnyai, S., Wang, F.: Investigating clinical care pathways correlated with outcomes. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 323–338. Springer, Heidelberg (2013)
21. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: a taxonomy of patterns. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 159–175. Springer, Heidelberg (2009)
22. Lo, D., Cheng, H., Han, J., Khoo, S.C., Sun, C.: Classification of software behaviors for failure detection: a discriminative pattern mining approach. In: KDD, pp. 557–566. ACM (2009)
23. Carmona, J., Gavaldà, R.: Online techniques for dealing with concept drift in process mining. In: Hollmén, J., Klawonn, F., Tucker, A. (eds.) IDA 2012. LNCS, vol. 7619, pp. 90–102. Springer, Heidelberg (2012)
24. Esparza, J., Römer, S., Vogler, W.: An improvement of mcmillan's unfolding algorithm. Formal Methods in System Design **20**(3), 285–310 (2002)
25. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Behavioral comparison of process models based on canonically reduced event structures. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 267–282. Springer, Heidelberg (2014)
26. van Beest, N., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. Eprint no. 83018. Queensland University of Technology (2015)
27. Cook, J.E., Wolf, A.L.: Event-based detection of concurrency. In: FSE, pp. 35–45. ACM (1998)

28. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Bp-diff: a tool for behavioral comparison of business process models. In: Limonad, L., Weber, B. (eds.) Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014). CEUR Workshop Proceedings, vol. 1295, pp. 1–6. CEUR-WS.org (2014)
29. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri Nets, Asymmetric Event Structures, and Processes. Information and Computation **171**, 1–49 (2001)
30. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Inf. Syst. **38**(4), 585–605 (2013)
31. van Dongen, B.F., Desel, J., van der Aalst, W.M.P.: Aggregating causal runs into workflow nets. T. Petri Nets and Other Models of Concurrency **6**, 334–363 (2012)