Mining with Eve - Process Discovery and Event Structures

Robin Bergenthum, Benjamin Meis

Department of Software Engineering,
FernUniversität in Hagen
{firstname.lastname}@fernuni-hagen.de

Abstract. This short-paper introduces our new process discovery plug-in *Eve*. *Eve* is part of the ProM Tool framework and based on event structures. *Eve* folds an event log to an event structure, before synthesizing a workflow model. The idea to recast process mining operations based on event structures as a behavioral model was suggested by Marlon Dumas and Luciano García-Bañuelos two years ago. This short paper is a sneak-preview to *Eve* sharing insight to its features.

1 Introduction

Process mining [1, 2] is to analyze recorded behavior of a business process to gain knowledge about performance and conformance of the process at hand. In the past decade, a high variety of process mining algorithms and methods have been introduced. Furthermore, there is a big number of academic and commercial tools, contests, and case studies. The homepage *http://www.win.tue.nl/ieeetfpm* of the IEEE Task Force on Process Mining provides a well-sorted introduction.

Process mining is mainly based on two formalisms: *event logs* and *workflow models*. An event log is a set of sequences of tasks recording behavior of a business process. A workflow model is an executable, often Petri net-like, model of a business process. Specifying workflow models we can consider different workflow modeling languages like workflow nets [3], BPMN, EPC, and Activity diagrams.

The four main operations of process mining are (i) *conformance checking*, (ii) *model comparison*, (iii) *deviance mining*, and (iv) *process discovery*. Every process mining operation can be defined as some transformation or comparison between event logs and workflow models. For example, process discovery [4, 5] is to transform an event log to a workflow model, so that it is 'likely' that the event log was produced by the generated model.

Two years ago, Marlon Dumas and Luciano García-Bañuelos suggested to base the fast growing field of process mining to a uniform representation of behavior [15]. They suggested to use event structures [18] to recast all process mining operations. An event structure is a set of partially ordered events together with an additional conflict relation. Thus, we are able to explicitly express concurrency, causality, and conflicts between events. Roughly speaking, paper [15] suggests that event structures serve as a perfect link between event logs and workflow models.

Figure 1 (adapted from [15]) depicts how Dumas and García-Bañuelos recast process mining operations based on events structures: (i) *conformance checking* is to unfold

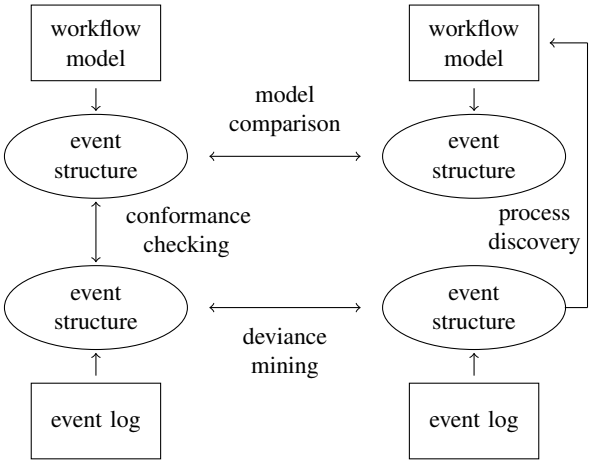

Fig. 1. Process mining operations based on event structures [15].

the workflow model to an event structure and merge the event log into a second event structure. Now, it is easy to compare both generated structures. (ii) *model comparison* is to compare the related event structures of two workflow models instead of comparing the models itself. Likewise, (iii) *deviance mining* is to compare two event structures related to two different event logs. Finally, (iv) *process discovery* is to synthesis a workflow model from an event structure related to an event log.

Event structures seem to be a fitting basis for process mining operations. The main benefit is that we can formalize the operations in a more explicit manner. To justify this claim, in this short-paper, we implement a process discovery algorithm called *Eve* following the suggested approach. We first, fold an event log into an event structure, before synthesizing a workflow model. We implement *Eve* as a plug-in for the most prominent process mining tool ProM [16] and share first insights of its features.

2 Process Discovery with Event Structures

Primus inter pares of all process mining operations is *process discovery*. The goal of process discovery is to generate a workflow model 'fitting' a recorded event log. In the literature, we find different discovery algorithms based on various representations of behavior. Some of the outstanding candidates, each representing a class of algorithms based on the same representation, are the famous α-algorithm [5], the heuristics-miner [19], discovery using state-based regions [14], discovery with regions of languages [8, 7] and folding of partial orders [17, 9]. For an introduction to process discovery we refer the reader to [1, 2]. Here, we assume the reader is familiar with the basic ideas and concepts.

In this paper, we introduce our ProM tool process discovery plug-in *Eve*. We do not claim that *Eve* is faster, prettier, or leads to better results, than all its predecessors, but *Eve* is based on the ideas presented in paper [15]. Our goal is to prove 'by example' that the idea of using event structures as a link between event logs and workflow models is indeed very valuable - not only for a theoretical point of view.

2.1 From an Event Log to an Event Structure

The input to every process discovery algorithm is an event log.

Definition 1 (Event Log). *Let T be a finite set of actions and let C be a finite set of cases. We call an element $e \in (T \times C)$ a task and call a sequence of tasks $\sigma \in (T \times C)^*$ an event log. Fix a case $c \in C$, we denote $p_c : (T \times C) \to T$ as $p_c(t, c') = t$ if $c = c'$ and $p_c(t, c') = \lambda\ else$ (λ the empty word). Let $\sigma = e_1, \ldots, e_n$ be an event log, we define the language of σ as $L(\sigma) = \{p_c(e_1) \ldots p_c(e_i) | i \leq n, c \in C\} \subseteq T^*$.*

The first step of our discovery algorithm is to deduce a concurrency relation from an event log to merge the language of the log to a set of partial orders. Thus, we need an appropriate so-called *concurrency oracle*. At this point, we either consider concurrency at the level of tasks or concurrency at the level of actions. Most discovery algorithms consider concurrent actions, i.e. if two actions are concurrent, the related tasks are concurrent in every case. Using event structures it is possible to also specify concurrency at the level of tasks. For most practical examples this may be an overkill and often leads to models that are very *precise* but lack *generalization* and *readability*. However, a concurrency oracle can discover concurrent tasks if the log contains appropriate additional data like e.g. life-cycles, localities, or flow of resources. We refer the reader to [6] for a more detailed discussion.

To tackle different levels of concurrency, we implement three different concurrency oracles in *Eve*. The *'later than'-oracle* of *Eve* requires additional data. *Eve* applies this oracle as soon as the tasks of an event log have a set of predecessors defining a *'later than'*-relation on the level of task. In that case, *Eve* deduces a related concurrency relation. The *life-cycle oracle* requires additional information about life-cycles of tasks. If the lifespans of two tasks intersect, *Eve* deduces a concurrency relation for both related actions. The *α-oracle* requires no additional data. Like suggested in paper [15], the conflict and concurrency matrix of the α-algorithm [5] is used to define a concurrency relation on the level of actions.

Eve choses the most precise oracle if the required data is recorded in the event log, i.e. we prefer the *'later than'-oracle* to the *life-cycle oracle* and the *life-cycle oracle* to the *α-oracle*. In any case, we apply one of these oracles to transform the log into a set of partial orders.

Definition 2 (Labeled Partial Order). *Let T be a set of labels. A labeled partial order (lpo) is a triple $lpo = (V, <, l)$ where V is a finite set of events, $< \subseteq V \times V$ is a transitive and irreflexive relation, and the labeling function $l : V \to T$ assigns a label to every event.*

The second step of our discovery algorithm is to add the set of labeled partial orders to one and initially empty event structure. We replay all partial orders in the event structure, adding new events, conflicts, and dependencies as we go. Roughly speaking, we merge similar prefixes of partial orders and extend the conflict relation whenever needed.

Definition 3 (Labeled Prime Event Structure). *Let T be a set of labels. A labeled prime event structure (event structure) is a tuple $(V, <, \#, l)$ where $(V, <, l)$ is an lpo and $\# \subseteq V \times V$ is an irreflexive, symmetric relation satisfying $e \# e' \wedge e' < e'' \Rightarrow e \# e''$.*

Just like a partial order, an event structure is able to explicitly express concurrency between events. Two events occur concurrently if they are neither ordered by the <-relation nor ordered by the #-relation. Maximal sets of events so that no pair is in the #-relation are so-called consistency-sets. After the folding procedure of *Eve*, the set of all consistency-sets is the set of partial orders of the language of the event log. Yet, the event structure is a much more compact representation.

2.2 From an Event Structure to a Workflow Model

The third step of *Eve* is to clean the produced event structure before we translate this structure into a workflow model. At this point, for most examples, the event structure already generalizes the event log. Still, the event structure may contain *noise* and may be *incomplete*. Paper [15] suggests to fold the event structure. Roughly speaking, to merge events carrying the same label. This is a very brute force generalization of the recorded behavior, but after such folding the event structure can be easily be transformed into a workflow model.

After some testing, we implement a more flexible approach based on frequencies in *Eve*. We count the number of cases of the event log relating to each consistency-set of the event structure. Starting by the consistency-set with the highest frequency we use a region based approach to synthesize a related workflow model. We use the theory of compact regions [10, 11] based on compact tokenflows [13, 12], but restrict the more general approach to workflow nets. More precisely, we synthesize a Petri net without arc weights, we only allow one *input place* if possible, for every transition there is a path from an input place to this transition, and every transition of the set of considered consistency-set can fire at least once. We add consistency-sets, ordered by their frequencies, as long as we can synthesize such a workflow model. If the model 'breaks', we stop adding consistency-sets and take the last valid model as an approximation to the initial event structure. The output of *Eve* is the generated workflow model, as well as the percentage of cases considered.

3 Conclusion and Future Work

Our ProM plug-in *Eve* is based on event structures as a formal model for process discovery. The generated workflow model 'fits' the recorded event log. On the one hand, the three different *concurrency oracles* and the considered restricted net class lead to *simple* models slightly *generalizing* the recorded behavior. Using compact regions together with *frequencies* we get a model with a high *fitness* because all considered consistency-sets are executable in the generated workflow net.

Eve is part of the 'nightly build' of ProM Tools at *www.promtools.org*. We encourage the reader to download ProM and try *Eve*. Please find links, additional examples, and a short description at *www.fernuni-hagen.de/sttp/forschung/eve*.

After a first round of testing, we are very happy with the produced results. First experiments are promising and suggest that it is beneficial to perform more specific experiments comparing *Eve* to other process discovery approaches in the near future. There is a lot of potential for fine tuning the four different steps (i.e. oracle, merge,

clean, synthesis) performed by *Eve*. This short-paper is more like a sneak-preview introducing *Eve* as a promising concept for further research.

References

[1] van der Aalst, W. M. P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] IEEE Task Force on Process Mining: *Process Mining Manifesto*. Business Process Management Workshops, LNBIP 99, Springer, 2012, 169–194.

[3] van der Aalst, W. M. P.: *The Application of Petri Nets to Workflow Management*. Journal of Circuits, Systems, and Computers 8(1), 1998, 21–66.

[4] van der Aalst, W. M. P.; van Dongen, B. F.: *Discovering Petri Nets from Event Logs*. ToPNoC VII, LNCS 7480, Springer, 2013, 372–422.

[5] van der Aalst, W. M. P.; Weijters, T.; Maruster, L.: *Workflow Mining: Discovering Process Models from Event Logs*. IEEE Trans. Knowl. Data Eng. 16(9), 2004, 1128–1142.

[6] Armas-Cervantes, A.; La Rosa, M. ;Dumas M.: *Local Concurrency Detection in Business Process Event Logs*. QUT ePrints 102438, 2016.

[7] Bergenthum, R.; Desel, J.; Mauser, S.: *Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language*. ToPNoC III, LNCS 5800, Springer, 2009, 216–243.

[8] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: *Process Mining Based on Regions of Languages*. Business Process Management 2007, LNCS 4714, Springer, 2007, 375–383.

[9] Bergenthum, R.; Mauser, S.: *Folding partially ordered runs*. Workshop ART 2011, CEUR 725, 2011, 52–62.

[10] Bergenthum, R.: *Compact Regions for Place/Transition Nets*. Workshop ATAED 2015, CEUR 1371, 2015, 112–116.

[11] Bergenthum, R.: *Synthesizing Petri Nets from Hasse Diagrams*. submitted to BPM 2017.

[12] Bergenthum, R.; Lorenz, R.: *Verification of Scenarios in Petri Nets Using Compact Tokenflows*. Fundamenta Informaticae 137, IOS Press, 2015, 117–142.

[13] Bergenthum, R.: *Faster Verification of Partially Ordered Runs in Petri Nets Using Compact Tokenflows*. Petri Nets 2013, LNCS 7927, Springer, 2013, 330–348.

[14] Carmona, J.; Cortadella, J.; Kishinevsky, M.: *New Region-Based Algorithms for Deriving Bounded Petri Nets*. IEEE Trans. Computers 59(3), 2010, 371–384.

[15] Dumas, M.; García-Bañuelos, L.: *Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs*. Petri Nets 2015, LNCS 9115, Springer, 2015, 33–48.

[16] van Dongen, B.; Alves de Medeiros, A. K.; Verbeek, H. M. W.; Weijters, A. J. M. M.; van der Aalst, W.M.P.: *The ProM framework: A New Era in Process Mining Tool Support*. Petri Nets 2005, LNCS 3536, Springer, 2015, 444–454.

[17] van Dongen, B.; van der Aalst, W. M. P.: *Multi-Phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets*. Applications of Petri Nets to Coordination, Workflow and Business Process Management 2005, 2015, 35–58.

[18] Nielsen, M.; Plotkin, G.; Winskel, G.: *Petri Nets, Event Structures and Domains, Part I*. Theoretical Computer Science 13, Elsevier, 1981, 85-108.

[19] Weijters, A. J. M. M.; Ribeiro, J. T. S.: *Flexible Heuristics Miner (FHM)*. CIDM, IEEE 2011, 310-317.