Partially Ordered Event Logs and Concurrency Oracles

Sabine Folz-Weinstein[1], Robin Bergenthum[2] and Christian Beecks[1]

[1] Chair of Data Science, FernUniversität in Hagen, Hagen, Germany
[2] Faculty for Mathematics and Computer Science, FernUniversität in Hagen, Hagen, Germany
sabine.folz-weinstein@fernuni-hagen.de

Process mining analyzes recorded behavior which is represented by event logs. Traditionally, an event log is a multiset of traces, where each trace is a totally ordered sequence of activities [2, 3, 5, 11]. The order of activities in a trace is typically based on their temporal occurrences. However, this temporal order does not necessarily reflect the actual causal order, as causally unrelated activities may also be ordered in time [4]. Yet, the order of activities is crucial for producing high-quality outputs with process mining algorithms. Consequently, there is a growing amount of work suggesting the usage of event logs in which the activities are partially ordered [7, 8, 9, 13].

Using partial orders, we can explicitly express both uncertainty and concurrency [12, 13, 14]. This allows us to remove total order relations for which there is insufficient or unreliable evidence, such as those based on inaccurate or incomparable timestamp information due to manual input or varying time granularities. Additionally, partial orders enable us to model specific properties of the underlying activities, like non-zero duration and timely overlap, which is not possible under the assumption of a total order. Additionally, since a partially ordered trace can "summarize" the behavior of multiple sequential traces, partially ordered logs often provide a more compact representation of the recorded behavior. This compactification can improve algorithm runtimes and reduce space complexity. Finally, partially ordered logs and traces can help identify "use cases" of a process, facilitating (or enabling) user-interactive exploratory process mining.

Up to this point, there is no common understanding of the terms "partially ordered log/trace" within the process mining community, and they are used with very different interpretations, strongly connected to the above-named reasons for use. For example, there exist various different semantics of concurrency. The first semantic dimension relates to certainty or uncertainty regarding the non-order of activities [13]. If we have no evidence for a specific causal order between two or more activities, this could be interpreted as indication that the activities are indeed concurrent, meaning the non-order itself provides causal relation information. Alternatively, it could be interpreted as missing information, suggesting that we simply do not know the order - or whether an order should exist at all. This directly leads to the semantical relation between a partial order and its sequentializations with respect to the represented behavior: Does a process model allow the execution of a partially ordered trace only if it allows all its sequentializations [7, 9, 10], or if it allows any one of its sequentializations [1, 15]? Furthermore, we must distinguish if we are interested in the fact if a model can or cannot execute certain behavior or if we are interested in counting percentages or probabilities, e.g. how many of the sequential traces represented by a partially ordered

trace, or which fraction of a partially ordered log can be executed. Another dimension is that activities can either be considered atomic or to have a certain duration, which leads to different interpretations of concurrency with respect to an interleaving or an overlapping of the activities [13]. Furthermore, there is the dimensionality of scope: concurrency which is detected within one or several traces within an event log can be interpreted to either be valid throughout the log, i.e., on an activity level, or to be valid only at the point within a trace where it is detected, i.e. on an event level [6].

We must consider all these different dimensions when discussing, evaluating, choosing and using methods and algorithms based on partially ordered input and when using concurrency oracles. Obviously, there are different needs and considerations depending on the process mining discipline, especially within discovery and conformance checking. Thus, it is vital to precisely define and explain the semantical interpretation of the term "partially ordered log" when used in an algorithm.

In this work, we define a partially ordered trace as a pair of a trace of length k and a partial order $\prec$ over the k positions of the trace. We define a partially ordered event log as a multiset of partially ordered traces. We discuss various heuristics to add a partial order on top of a sequential event log, called concurrency oracles. We implemented a configurable concurrency oracle ("CCO") tool as pm4py plugin [16]. In its current version, it supports "alpha" and lifecycle-based concurrency detection on event (log-wise) or activity (trace-wise) level, exporting the result as .xes-file with additional partial order information per trace and event. It allows to either retain all cases in the log, to reduce the log to one representative per sequential trace, or to one representative per partially ordered trace and, thus, supports different levels of compactification of the log, depending on the semantical relation between sequential and partially ordered trace required in the specific target algorithm and use case.

We evaluate the generation of partially ordered logs with our "CCO" tool using different combinations of parameters and several well-known BPI challenge logs, which illustrate the importance of a configurable concurrency oracle tool and a clear distinction of the purpose of the generated partially ordered log. If we have, for example, an event log with atomic activities like the BPI 2019 log which models a purchase order process, we have activities which are very likely to be concurrent in their execution throughout the log like the recording of the reception of goods or services and the recording of the reception of an invoice. Assuming that a partial order represents all its sequentializations, we may be interested in a compactification of the log. In this case, we observe that the 251.734 cases of 11.973 sequential traces fold together to 5.851 partially ordered traces, i.e., about 50% reduction of variants. In this example, it also makes sense to keep only one trace representative and, thus, have a significantly smaller log for further analysis. However, if we have an event log with lifecycle information like the BPI 2017 log modeling a loan application process, we may mainly be interested in adding concurrency information on event level and keep all trace variants, e.g. for conformance checking. Here, we observe that the 31.509 cases of 6.844 sequential traces fold together to 6.566 partially ordered traces due to isomorphy, i.e., almost no reduction in size, but a qualitative enhancement. The CCO allows us to experiment with different parameter combinations and pre-analyze the detected concurrency, e.g. concerning differences, specific patterns or validity.

References

1. van der Aa, H., Leopold, H., Weidlich, M.: Partial order resolution of event logs for process conformance checking. In: Decision Support Systems, Volume 136, Article no. 113347, Elsevier (2020).
2. van der Aalst, W. M. P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011).
3. van der Aalst, W. M. P.: Process Mining: Data Science in Action (https://doi.org/10.1007/978-3-662-49851-4). Springer (2016).
4. van der Aalst, W. M. P., Santos, L. F. R.: May I take your order? On the Interplay Between Time and Order in Process Mining. In: Business process management workshops, BPM 2021, pp. 99-110. Springer (2021).
5. van der Aalst, W. M. P., Carmona, J.: Process Mining Handbook. Springer (2022).
6. Armas-Cervantes, A., Dumas, M., La Rosa, M., Maaradji, A.: Local Concurrency Detection in Business Process Event Logs. In: ACM Transactions on Internet Technology, vol. 19, no. 1, pp. 1 – 23 (2019).
7. Bergenthum, R.: Prime Miner - Process Discovery using Prime Event Structures. ICPM 2019, pp. 41 – 48 (2019).
8. 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, pp. 33 – 48. Springer (2015).
9. Folz-Weinstein, S., Bergenthum, R., Desel, J., Kovár, J.: ILP² Miner – Process Discovery for Partially Ordered Event Logs Using Integer Linear Programming. In: Gomes, L., Lorenz, R. (eds.) Application and Theory of Petri Nets and Concurrency, PETRI NETS 2023, LNCS 13929, pp. 59 - 76, Springer (2023).
10. Grabowski, J.: On Partial Languages. In: Fundamenta Informaticae, vol. 4, no. 2, pp. 427 – 498. IOS Press (1981).
11. IEEE Task Force on Process Mining: Process Mining Manifesto. Business Process Management Workshops, Lecture Notes in Business Inormation Processing, vol. 99, pp. 169 – 194. Springer (2012).
12. Janicki, R., Koutny, M.: Structure of Concurrency. In: Theoretical Computer Science 112, no. 1, pp. 5 – 52. Elsevier (1993).
13. Leemans, S. J. J., van Zelst, S. J., Lu, X.: Partial-order-based process mining: a survey and outlook. Knowledge and Information Systems, vol. 65, pp. 1 – 29. Springer (2023).
14. Pratt, V.: Modelling Concurrency with Partial Orders. In: International Journal of Parallel Programming 15, pp. 33 – 71. (1986).
15. Sommers, D., Sidorova, N., van Dongen, B.: Conformance Checking with Model Projections - Rethinking Log-Model Alignments for Processes with Interacting Objects. In Application and Theory of Petri Nets and Concurrency, PETRI NETS 2024, pp. 61-82. Springer (2024).
16. Configurable Concurrency Oracle Tool repository, https://github.com/sabinefw/ConfigurableConcurrencyOracleTool , last access 2024/08/07