Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)
Sister Conferences Best Papers Track

POWL: Partially Ordered Workflow Language (Extended Abstract)*

Humam Kourani[1,2], **Sebastiaan van Zelst**[3]
[1]Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany
[2]RWTH Aachen University, Aachen, Germany
[3]Celonis Labs GmbH, Munich, Germany
humam.kourani@fit.fraunhofer.de, s.vanzelst@celonis.com

Abstract

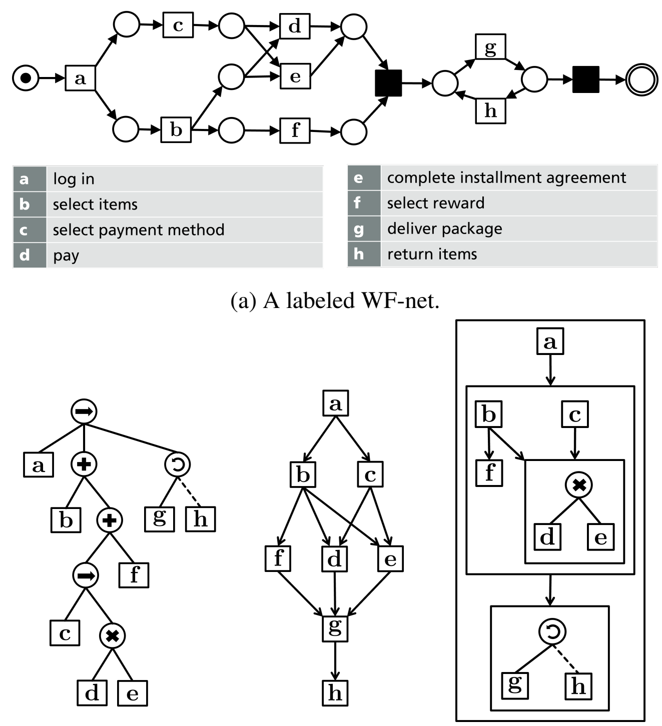
Processes in real-life scenarios tend to inherently establish partial orders over their constituent activities. This makes partially ordered graphs viable for process modeling. While partial orders capture both concurrent and sequential interactions among activities in a compact way, they fall short in modeling choice and cyclic behavior. To address this gap, we introduce the Partially Ordered Workflow Language (POWL), a novel language for process modeling that combines traditional hierarchical modeling languages with partial orders. In a POWL model, sub-models are combined into larger ones either as partial orders or using control-flow operators that enable the representation of choice and loop structures. This integration of hierarchical structure and partial orders not only offers an effective solution for process modeling but also provides quality guarantees that make POWL particularly suitable for the automated discovery of process models.

1 Introduction

Process modeling is essential for understanding, analyzing, and optimizing business operations. Process models, which can be manually created or automatically discovered from data, provide insights into process execution. By highlighting issues and bottlenecks within processes, process models enable organizations to automate and optimize their processes.

Different process modeling languages are employed in the field of business process management. *Petri nets*, especially their subclass *Workflow nets (WF-nets)*, exemplify a widely recognized formal notation for process modeling [van Hee *et al.*, 2013]. WF-nets may encounter behavioral quality issues, e.g., it is possible to generate WF-nets with unreachable parts. In contrast, process trees [Leemans, 2022] offer a hierarchical modeling approach ensuring soundness by design, albeit limited to representing hierarchical behavior.

To illustrate this, consider the process of an online shopping transaction illustrated as a WF-nets in Figure 1a. In this process, the item selection (activity b) and choosing a payment method (c) can happen in any order. After item selection (b), the user picks a free reward (f), a step dependent on the total purchase value yet independent from the payment activities (c, d, and e). A process tree, as demonstrated in Figure Figure 1b, uses control-flow operators like $\rightarrow$, $\times$, $+$, and $\circlearrowleft$ to represent sequential activities, exclusive choices, concurrency, and loops, respectively. Process trees struggle to accurately model the complex dependencies between b, c, d, e, and f as such non-hierarchical dependencies cannot be precisely captured using control-flow operators. For example, the tree in Figure 1b allows for selecting the reward and/or completing the payment before selecting the items.



(a) A labeled WF-net.

(b) A process tree. (c) A partial order. (d) A POWL model.

Figure 1: Process models for an online shopping process. For simplicity, we visualize the transitive reduction of partial orders.

Partial orders provide an effective and compact represen-

*This extended abstract is based on the paper "POWL: Partially Ordered Workflow Language" presented at the 21st International Conference on Business Process Management (BPM 2023).

tation of the execution order of activities within processes. In partial order, activities are assumed to be concurrent (i.e., can be executed in any order) unless order restrictions are explicitly defined. For instance, Figure 1c shows a graph representation of a partial order. The edge $(b \rightarrow f)$ indicates that selecting the reward is only possible after completing the item selection, while selecting the reward (f) and setting the payment method (c) can be executed in any order as they are not connected in the graph. While partial order are able to model the non-hierarchical dependencies between b, c, d, e, and f in our process, they fall short in representing cyclic behavior (in our example, the loop of delivering and returning items) as a partial order is irreflexive. Moreover, partial orders cannot describe choice behavior (such as the choice between payment and setting up an installment plan in our example) since all activities in a partial order are assumed to be executed."

The introduction of Partially Ordered Workflow Language (POWL) aims to integrate the best of hierarchical modeling languages with the flexibility of partial orders. A POWL model is a hierarchical model generated by combining submodels either as partial orders or using control-flow operators. Figure 1d shows a POWL model that precisely describes the behavior of our example process. This model captures the complex dependencies between b, c, $\{d, e\}$, and f as a partial order, and at the same time, it uses the control-flow operators $\times$ and $\circlearrowleft$ the model the choice of d and e and the and the loop between g and h respectively. Despite their support for partial orders, POWL models are guaranteed to be sound by construction, which makes POWL particularly suitable for the automated discovery of process models from data.

The remainder of the paper is structured as follows. We discuss related work in Section 2, and then POWL is introduced in Section 3. In Section 4, we present an approach that demonstrates the feasibility of employing POWL in process discovery, and we evaluate this discovery approach on real-life data in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

The POWL language is introduced in [Kourani and van Zelst, 2023], and significant advancements in the discovery of POWL models are reported in [Kourani *et al.*, 2023]. Furthermore, [Kourani *et al.*, 2024a; Kourani *et al.*, 2024b] investigate leveraging the synergy between POWL and large language models, highlighting the capability of POWL in facilitating process modeling through innovative artificial intelligence techniques.

A comprehensive overview of various languages utilized for process modeling is provided in [Schnieders *et al.*, 2004]. Efforts to combine diverse modeling notations have led to innovative hybrid models. For instance, [van der Aalst *et al.*, 2023] describes a hybrid Petri net featuring formal and informal constraints between activities, while [Slaats *et al.*, 2016] explores combining imperative and declarative models.

A survey on the application of partial orders in data representation and process modeling is provided in [Leemans *et al.*, 2022]. In [Mannila *et al.*, 1997; Leemans and van der Aalst, 2014], the discovery of frequent partially ordered sets of events is explored. *Prime event structures* are defined in [Nielsen *et al.*, 1981] as partially ordered graphs extended with conflict relations, and [Dumas and García-Bañuelos, 2015] proposes approaches for the discovery of prime event structures. In [Mokhov and Yakovlev, 2010], *Conditional partial order graphs* are defined as families of partial orders, and they are employed in process discovery in [Mokhov and Carmona, 2015]. Both prime event structures and conditional partial order graphs address the challenge of modeling choices within partial orders, yet the integration of cyclic behavior remains a challenge for these notations. Lastly, [Ouyang *et al.*, 2007] introduces *BPEL* as a flow language that combines web service components through control-flow constructs, and it additionally supports adding order constraints between parallel components. Despite BPEL's advanced capabilities, its complexity positions it closer to a programming tool for web service implementation rather than a user-friendly modeling language [van der Aalst *et al.*, 2005].

3 POWL Language

In this section, we introduce POWL and we illustrate an approach for transforming POWL models into sound WF-nets.

Notation Let $\prec \subseteq X \times X$ be a 2-ary relation over a set X. For $(x_1, x_2) \in X \times X$, we write $x_1 \prec x_2$ to denote that $(x_1, x_2) \in \prec$, and we write $x_1 \not\prec x_2$ to denote that $(x_1, x_2) \notin \prec$. We call $\prec$ a *partial order* if it is *irreflexive* (i.e., $x \not\prec x$ for all $x \in X$) and *transitive* (i.e., if $x_1 \prec x_2$ and $x_2 \prec x_3$, then $x_1 \prec x_3$). Irreflexivity and transitivity imply *asymmetry* (i.e., if $x_1 \prec x_2$, then $x_2 \not\prec x_1$). We refer to $\rho = (X, \prec)$ as a *partially ordered set (poset)*. We use Σ to denote the universe of activities, and we use $\tau \notin \Sigma$ to denote the *silent activity* (also called the *invisible activity*), which is used to model skippable parts and self-loops [van der Aalst, 2016].

A POWL model represents a process as a partially ordered graph, augmented with control-flow operators to depict choices and loops. We define three types of POWL models. First, the *base case* comprises single activities. For the second type, the control-flow operators $\times$ and $\circlearrowleft$ are employed to combine sub-models into a larger POWL model. We use $\times$ to model an exclusive choice of POWL models, and we use $\circlearrowleft$ to model a do-redo loop of two POWL models: the first submodel (*do-part*) is executed first, and every execution of the second sub-model (*redo-part*) invokes a subsequent execution of the do-part. For the third type of POWL models, submodels are combined as a partially ordered set. Unconnected nodes in a partial order indicate concurrency, while connections between nodes represent sequential dependencies.

Definition 1 (POWL Model)**.** *A POWL model is defined recursively as follows:*

- *Any activity $a \in \Sigma \cup \{\tau\}$ is a POWL model.*
- *Let ψ_1 and ψ_2 be two POWL models. $\circlearrowleft(\psi_1, \psi_2)$ is a POWL model.*
- *Let $P = \{\psi_1, ..., \psi_n\}$ be a set of $n \geq 2$ POWL models.*
  - *$\times(\psi_1, ..., \psi_n)$ is a POWL model.*
  - *A poset $\rho = (P, \prec)$ is a POWL model.*

For the formal semantics of POWL models, we refer to [Kourani and van Zelst, 2023]. POWL models can be recur-

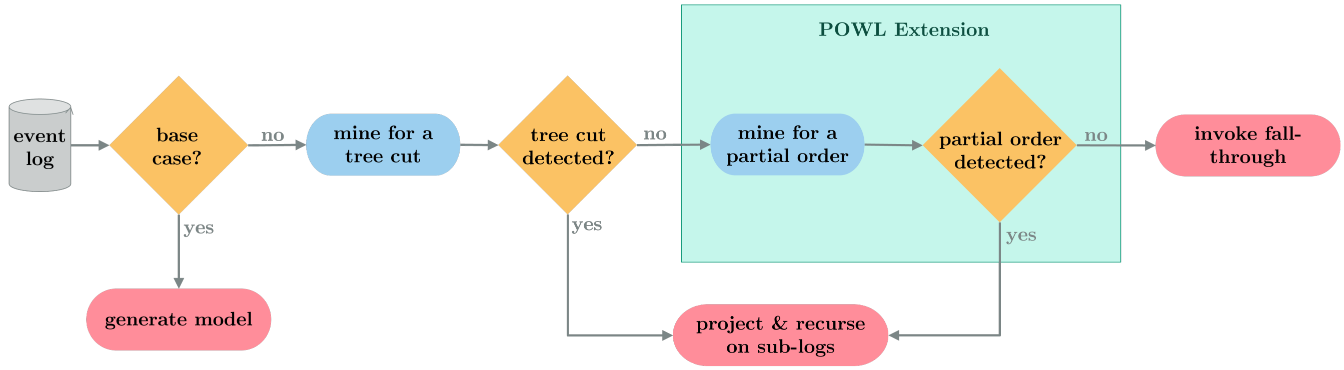


Figure 2: Approach for the discovery of POWL models from event logs.

sively transformed into WF-nets, and the generated workflow net is guaranteed to be sound [Kourani and van Zelst, 2023].

4 Discovery of POWL Models

This section explores the practical application of POWL in the realm of process discovery by adapting the inductive miner [Leemans, 2022] to generate POWL models.

4.1 Event Log

In the context of process discovery, data is assumed to be presented as an *event log*. We define an event log as a multi-set comprising sequences of activities, called *traces*, that encapsulate the execution flow of different process instances. We write an event log as a multi-set $L = [{\sigma_1}^{f_1}, ..., {\sigma_n}^{f_n}]$ where f_i refers to the frequency of the trace σ_i for $1 \leq i \leq n$. For example, $L_1 = [\langle a, b, c\rangle^3, \langle a, b, d\rangle^2]$ is an event log that comprises the trace $\langle a, b, c\rangle$ with a frequency of 3 and another trace $\langle a, b, d\rangle$ with a frequency of 2.

4.2 Inductive Miner

The inductive miner [Leemans, 2022] is an approach widely used for process discovery due to its high scalability and formal guarantees. These guarantees include soundness, rediscoverability of specific structures, and perfect fitness (i.e., all behaviors observed in the event log are included within the model's language).

Employing a recursive, top-down strategy, the algorithm seeks a *cut* by identifying a behavioral pattern in the event log and partitioning the activities accordingly. Recognizing four distinct cuts corresponding to the process tree operators, it recursively constructs a process tree that models the identified patterns. Subsequent to a cut detection, the event log is projected onto the activity partitions, generating sub-logs, and the algorithm is recursively applied on the sub-logs until reaching a *base case*. A base case, defined as a log containing a single activity, trivially translates into a process tree node.

In scenarios where neither a base case nor a cut is detected, the Inductive Miner employs a *fall-through function*. This function always returns a cut, potentially encompassing behavior more general than what is observed in the event log, yet ensuring the continuity of recursion. For a detailed exposition of the inductive miner, we refer to [Leemans, 2022].

4.3 Mining for Partial Orders

As illustrated in Figure 2, we adapt the inductive miner to discover POWL models by additionally mining for partial orders. This additional step is performed after no standard process tree cuts are discovered and before invoking the fall-through function. Our discovery approach involves generating candidate partial orders over different partitionings of activities and validating them through predefined rules to ensure a high conformance between the partial order and the input event log. Upon identifying a *valid* partial order, the event log is accordingly projected, and the recursive procedure progresses with the derived sub-logs. In the absence of a valid order, the fall-through function is triggered. Note that in cases where a sequence or concurrency cut is detected, the cut is transformed into a partial order as POWL models do not support the operators $\rightarrow$ and $+$. For more details on the partial order cut detection step and the applied validity rules, we refer to [Kourani and van Zelst, 2023]. Note that this discovery approach was extended in [Kourani *et al.*, 2023] to improve scalability on large data sets.

Figure 3 shows an example application of our discovery approach on an event log. In the first iteration, a partial order is discovered, and the event log is projected on the subsets of activities $\{a, b\}$, $\{c\}$, and $\{d\}$. Afterwards, a choice cut between the activities a and b is detected.

5 Evaluation

The implementation of our approach for discovering POWL models is available in the open-source Python library PM4Py (http://pm4py.org/). We evaluated our approach (IM_P) on real-life event logs, and we compared the resulting models with the models discovered by the base inductive miner (IM).

Setup

We transformed all discovered models into WF-nets in order to enable the assessment of their quality using traditional conformance-checking techniques [Carmona *et al.*, 2018]. For each model, we computed three conformance-checking scores: *fitness*, *precision*, and *simplicity*. Fitness [Berti and van der Aalst, 2019] measures the extent to which the observed behavior in the event log can be reproduced by the

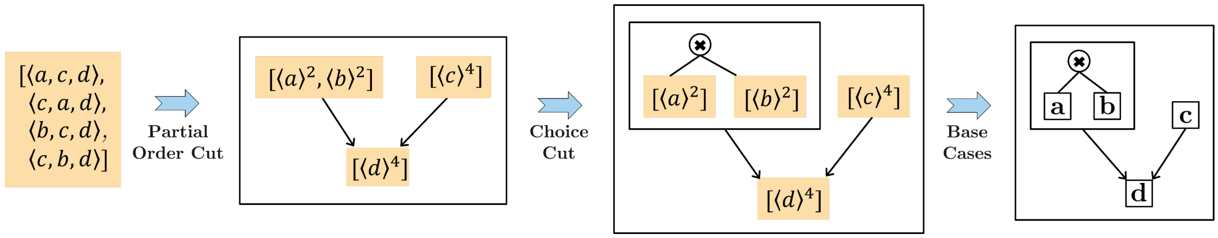


Figure 3: Example illustrating the discovery of a POWL model from an event log.

Log	#act.	Time (sec) IM	Time (sec) IM_P	Precision IM	Precision IM_P	Simplicity IM	Simplicity IM_P
BPI-17	8	2	1	0.37	0.68	0.67	0.74
	12	18	6	0.2	0.34	0.65	0.68
BPI-18	8	26	16	0.35	0.32	0.65	0.63
	12	60	112	0.2	0.21	0.61	0.63
BPI-19	8	3	3	0.62	0.78	0.64	0.67
	12	6	3	0.55	0.7	0.64	0.64
BPI-20-1	8	ϵ	ϵ	0.4	0.4	0.65	0.66
	12	ϵ	38	0.5	0.54	0.61	0.67
BPI-20-2	8	ϵ	ϵ	0.5	0.53	0.67	0.71
	12	ϵ	ϵ	0.47	0.51	0.65	0.69
BPI-20-3	8	ϵ	ϵ	0.51	0.51	0.67	0.67
	12	ϵ	280	0.33	0.35	0.65	0.67
BPI-20-4	8	ϵ	ϵ	0.43	0.39	0.63	0.68
	12	ϵ	277	0.23	0.35	0.58	0.68
BPI-20-5	8	ϵ	ϵ	0.75	0.75	0.63	0.68
	12	ϵ	41	0.49	0.49	0.6	0.67
Sepsis	8	ϵ	ϵ	0.51	0.51	0.64	0.65
	12	ϵ	ϵ	0.34	0.35	0.64	0.65
Fine	8	ϵ	ϵ	0.76	0.76	0.66	0.67
	11	ϵ	7	0.58	0.58	0.62	0.63
Hospital	8	ϵ	ϵ	0.78	0.78	0.67	0.67
	12	ϵ	2	0.6	0.6	0.65	0.66

Table 1: Evaluation results. Higher precision and simplicity values are highlighted in green. Time values below 1 second are replaced by ϵ.

model; *precision* [Munoz-Gama and Carmona, 2010] assesses how accurately the model avoids allowing behavior that is not seen in the event log, ensuring that the model does not overgeneralize; and *simplicity* evaluates the structural complexity of the model based on the average number of arcs per node.

Our evaluation encompasses several real-life event logs, including a sepsis cases log from a hospital [Mannhardt, 2016], a road traffic fine management system log [de Leoni and Mannhardt, 2015], a hospital billing system log [Mannhardt, 2017], and the BPI Challenge logs from 2017 to 2020 [van Dongen, 2017; van Dongen and Borchert, 2018; van Dongen, 2019; van Dongen, 2020]. The logs are pre-filtered to retain the most frequent activities, capped at 8 and 12 activities.

Results

Table 1 reports the discovery time and conformance-checking scores for all models. We omit the fitness values as all models achieved a fitness of 1 due to the perfect fitness guarantee both IM and IM_P provide.

On one hand, IM_P demonstrated improved time performance compared to IM in certain scenarios (e.g., BPI Challenge 2017). Conversely, IM_P was more time-intensive in other instances. For example, both approaches required less than 1 second for the BPI Challenge 2020-3 log of 8 activities, while the time increased to 280 seconds for IM_P with the log of 12 activities. This highlights how increasing the number of activities can significantly degrade time performance. These findings were expected since IM_P employs a brute force method to generate and validate candidate partial orders. Note that efforts to address this scalability concern were made in [Kourani *et al.*, 2023].

Overall, IM_P yielded simpler models than IM. In 18 cases, the model discovered by IM_P achieved a higher simplicity score compared to only one instance where IM led to a simpler model. Similarly, IM_P generally obtained higher precision values. For example, the model discovered by IM_P for the BPI Challenge 2017 event log with 8 activities achieved a precision of 0.68, whereas IM's model achieved 0.37. In this scenario, the support of partial orders enabled IM_P to identify local dependencies between activities that cannot be captured in process trees. Although IM_P generally produced more precise models compared to IM (for 11 logs), there were two exceptions. For instance, in the BPI Challenge 2018 event log with 8 activities, the precision decreased from 0.35 to 0.32. This case exemplifies where invoking the fall-through function yielded better results than detecting a partial order. To continue the recursion, the fall-through function returned a concurrency cut between an activity occurring at most once in every trace and the remaining activities.

6 Conclusion

This paper presents the Partially Order Workflow Language (POWL), a novel language for business process modeling. A POWL model is a hierarchical model generated by combining sub-models either as partial orders or using standard control-flow operators for modeling choice and loop structures. Moreover, we propose an approach that demonstrates the feasibility of utilizing POWL in process discovery, and we evaluate our approach on real-life event logs. The evaluation showcases the ability of POWL to uncover complex process structures unattainable with traditional hierarchical modeling languages.

References

[Berti and van der Aalst, 2019] Alessandro Berti and Wil M. P. van der Aalst. Reviving token-based replay: Increasing speed while improving diagnostics. In Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, Satellite event of Petri Nets 2019 and ACSD 2019*, volume 2371 of *CEUR Workshop Proceedings*, pages 87–103. CEUR-WS.org, 2019.

[Carmona *et al.*, 2018] Josep Carmona, Boudewijn van Dongen, Andreas Solti, and Matthias Weidlich. Conformance checking. *Switzerland: Springer.[Google Scholar]*, 56, 2018.

[de Leoni and Mannhardt, 2015] M. (Massimiliano) de Leoni and Felix Mannhardt. Road Traffic Fine Management Process, 2015.

[Dumas and García-Bañuelos, 2015] Marlon Dumas and Luciano García-Bañuelos. Process mining reloaded: Event structures as a unified representation of process models and event logs. In *Application and Theory of Petri Nets and Concurrency - 36th International Conference, Proceedings*, volume 9115 of *LNCS*, pages 33–48. Springer, 2015.

[Kourani and van Zelst, 2023] Humam Kourani and Sebastiaan J. van Zelst. POWL: partially ordered workflow language. In Chiara Di Francescomarino, Andrea Burattin, Christian Janiesch, and Shazia Sadiq, editors, *Business Process Management - 21st International Conference, BPM 2023, Utrecht, The Netherlands, September 11-15, 2023, Proceedings*, volume 14159 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2023.

[Kourani *et al.*, 2023] Humam Kourani, Daniel Schuster, and Wil M. P. van der Aalst. Scalable discovery of partially ordered workflow models with formal guarantees. In *5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23-27, 2023*, pages 89–96. IEEE, 2023.

[Kourani *et al.*, 2024a] Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Process Modeling With Large Language Models. In *Enterprise, Business-Process and Information Systems Modeling - 25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024, Limassol, Cyprus, June 3-4, 2024, Proceedings*. Springer, 2024. In press.

[Kourani *et al.*, 2024b] Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. ProMoAI: Process Modeling with Generative AI. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, 03-09 August 2024*. ijcai.org, 2024. In press.

[Leemans and van der Aalst, 2014] Maikel Leemans and Wil M. P. van der Aalst. Discovery of frequent episodes in event logs. In *Data-Driven Process Discovery and Analysis - 4th International Symposium, Revised Selected Papers*, volume 237 of *LNBIP*, pages 1–31. Springer, 2014.

[Leemans *et al.*, 2022] Sander J.J. Leemans, Sebastiaan J. van Zelst, and Xixi Lu. Partial-order-based process mining: a survey and outlook. *Knowl Inf Syst*, 2022.

[Leemans, 2022] Sander J. J. Leemans. *Robust Process Mining with Guarantees - Process Discovery, Conformance Checking and Enhancement*, volume 440 of *LNBIP*. Springer, 2022.

[Mannhardt, 2016] Felix Mannhardt. Sepsis Cases - Event Log, 2016.

[Mannhardt, 2017] Felix Mannhardt. Hospital Billing - Event Log, 2017.

[Mannila *et al.*, 1997] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[Mokhov and Carmona, 2015] Andrey Mokhov and Josep Carmona. Event log visualisation with conditional partial order graphs: from control flow to data. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, Satellite event of Petri Nets 2015 and ACSD 2015*, volume 1371 of *CEUR Workshop Proceedings*, pages 16–30. CEUR-WS.org, 2015.

[Mokhov and Yakovlev, 2010] Andrey Mokhov and Alexandre Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Trans. Computers*, 59(11):1480–1493, 2010.

[Munoz-Gama and Carmona, 2010] Jorge Munoz-Gama and Josep Carmona. A fresh look at precision in process conformance. In Richard Hull, Jan Mendling, and Stefan Tai, editors, *BPM 2010, Hoboken, NJ, USA. Proceedings*, volume 6336 of *LNCS*, pages 211–226. Springer, 2010.

[Nielsen *et al.*, 1981] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theor. Comput. Sci.*, 13:85–108, 1981.

[Ouyang *et al.*, 2007] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Sci. Comput. Program.*, 67(2-3):162–198, 2007.

[Schnieders *et al.*, 2004] Arnd Schnieders, Frank Puhlmann, and Mathias Weske. Process modeling techniques. *PESOA Report TR*, 1, 2004.

[Slaats *et al.*, 2016] Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio Maria Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Proceedings*, volume 10033 of *LNCS*, pages 531–551, 2016.

[van der Aalst *et al.*, 2005] Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. ter Hofstede, Nick Russell, H. M. W. Verbeek, and Petia Wohed. Life after BPEL? In *Formal Techniques for Computer Systems and Business Processes, EPEW 2005 and WS-FM 2005, Proceedings*, volume 3670 of *LNCS*, pages 35–50. Springer, 2005.

[van der Aalst *et al.*, 2023] Wil M. P. van der Aalst, Riccardo De Masellis, Chiara Di Francescomarino, Chiara Ghidini, and Humam Kourani. Discovering hybrid process models with bounds on time and complexity: When to be formal and when not? *Information Systems*, 116:102214, 2023.

[van der Aalst, 2016] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[van Dongen and Borchert, 2018] Boudewijn van Dongen and Florian Borchert. BPI Challenge 2018, 2018.

[van Dongen, 2017] Boudewijn van Dongen. BPI Challenge 2017, 2017.

[van Dongen, 2019] Boudewijn van Dongen. BPI Challenge 2019, 2019.

[van Dongen, 2020] Boudewijn van Dongen. BPI Challenge 2020, 2020.

[van Hee *et al.*, 2013] Kees M. van Hee, Natalia Sidorova, and Jan Martijn E. M. van der Werf. Business process modeling using Petri nets. *Trans. Petri Nets Other Model. Concurr.*, 7:116–161, 2013.