POWL: Partially Ordered Workflow Language (Extended Abstract)*

Humam Kourani1,2, Sebastiaan van Zelst3

1Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany
2RWTH Aachen University, Aachen, Germany
3Celonis 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 1b, uses control-flow operators like →, ×, +, and ⊙ 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.

Partial orders provide an effective and compact represen-
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 \(c\), \(d\), \(e\), \(f\) 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 sub-models 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 \(\circ\) 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 \((x_1, x_2) \not\in \prec\). 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 \(x_1 \not\prec x_2\), then \(x_2 \not\prec x_1\)). We refer to \(\rho = (X, \prec)\) as a partially ordered set (poset). We use \(\Sigma\) to denote the universe of activities, and we use \(\tau \not\in \Sigma\) 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 \(\circ\) 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 \(\circ\) to model a do-redo loop of two POWL models: the first sub-model (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, sub-models 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 \(\psi_1\) and \(\psi_2\) be two POWL models. \(\circ(\psi_1, \psi_2)\) 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-
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 \( \sigma_i \) for \( 1 \leq i \leq n \). For example, \( L_1 = [(a, b, c)^2, (a, b, d)] \) is an event log that comprises the trace \( (a, b, c) \) with a frequency of \( 3 \) and another trace \( (a, b, d) \) 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, discoverability 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 (IMp) 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
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 IMₚ provide.

![Figure 3: Example illustrating the discovery of a POWL model from an event log.](image)

<table>
<thead>
<tr>
<th>Log</th>
<th>#act.</th>
<th>Time (sec)</th>
<th>Precision IM</th>
<th>Precision IMp</th>
<th>Simplicity IM</th>
<th>Simplicity IMp</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPI-17</td>
<td>8</td>
<td>2</td>
<td>0.37</td>
<td>0.68</td>
<td>0.67</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>18</td>
<td>0.34</td>
<td>0.65</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>BPI-18</td>
<td>8</td>
<td>26</td>
<td>0.35</td>
<td>0.65</td>
<td>0.66</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>60</td>
<td>0.21</td>
<td>0.61</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>BPI-19</td>
<td>8</td>
<td>3</td>
<td>0.62</td>
<td>0.78</td>
<td>0.64</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>6</td>
<td>0.55</td>
<td>0.7</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>BPI-20-1</td>
<td>8</td>
<td>e</td>
<td>0.4</td>
<td>0.4</td>
<td>0.65</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.55</td>
<td>0.61</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>BPI-20-2</td>
<td>8</td>
<td>e</td>
<td>0.5</td>
<td>0.53</td>
<td>0.67</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.47</td>
<td>0.51</td>
<td>0.65</td>
<td>0.69</td>
</tr>
<tr>
<td>BPI-20-3</td>
<td>8</td>
<td>e</td>
<td>0.51</td>
<td>0.51</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.33</td>
<td>0.35</td>
<td>0.65</td>
<td>0.67</td>
</tr>
<tr>
<td>BPI-20-4</td>
<td>8</td>
<td>e</td>
<td>0.43</td>
<td>0.39</td>
<td>0.63</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.27</td>
<td>0.35</td>
<td>0.58</td>
<td>0.68</td>
</tr>
<tr>
<td>Sepsis</td>
<td>8</td>
<td>e</td>
<td>0.75</td>
<td>0.75</td>
<td>0.63</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.49</td>
<td>0.49</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>Fine</td>
<td>8</td>
<td>e</td>
<td>0.51</td>
<td>0.51</td>
<td>0.64</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>e</td>
<td>0.34</td>
<td>0.35</td>
<td>0.64</td>
<td>0.65</td>
</tr>
<tr>
<td>Hospital</td>
<td>8</td>
<td>e</td>
<td>0.34</td>
<td>0.34</td>
<td>0.64</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>e</td>
<td>0.6</td>
<td>0.6</td>
<td>0.65</td>
<td>0.66</td>
</tr>
</tbody>
</table>

On one hand, IMₚ demonstrated improved time performance compared to IM in certain scenarios (e.g., BPI Challenge 2017). Conversely, IMₚ 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 IMₚ with the log of 12 activities. This highlights how increasing the number of activities can significantly degrade time performance. These findings were expected since IMₚ 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, IMₚ yielded simpler models than IM. In 18 cases, the model discovered by IMₚ achieved a higher simplicity score compared to only one instance where IM led to a simpler model. Similarly, IMₚ generally obtained higher precision values. For example, the model discovered by IMₚ 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 IMₚ to identify local dependencies between activities that cannot be captured in process trees. Although IMₚ 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: In-


