Translating Workflow Nets into the Partially Ordered Workflow Language

Humam Kourani[1,2][0000−0003−2375−2152], Gyunam Park[1][0000−0001−9394−6513], and Wil M.P. van der Aalst[1,2][0000−0002−0955−6940]

[1] Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven, 53757 Sankt Augustin, Germany
{humam.kourani,gyunam.park,wil.van.der.aalst}@fit.fraunhofer.de
[2] RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Germany

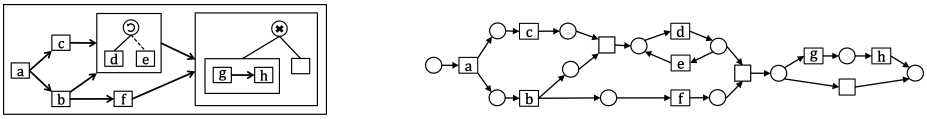
Abstract. The Partially Ordered Workflow Language (POWL) has recently emerged as a process modeling notation, offering strong quality guarantees and high expressiveness. However, its adoption in practice is hindered by the prevalence of standard notations like workflow nets (WF-nets) and BPMN. This paper presents a novel algorithm for transforming safe and sound WF-net into equivalent POWL models. The algorithm recursively identifies structural patterns within the WF-net and translates them into their POWL representation. We formally prove the correctness of our approach, showing that the generated POWL model preserves the language of the input WF-net. Furthermore, we demonstrate the high scalability of our algorithm, and we show its completeness on a subclass of WF-nets that encompasses equivalent representations for all POWL models. This work bridges the gap between the theoretical advantages of POWL and the practical need for compatibility with established notations, paving the way for broader adoption of POWL in process analysis and improvement applications.

Keywords: Workflow Net · Process Modeling · Model Transformation

1 Introduction

The field of process modeling and analysis relies heavily on formal notations to represent and reason about the behavior of complex systems. While standard notations like Petri nets [26], and specifically Workflow Nets (WF-nets) [22], and Business Process Model and Notation (BPMN) [28] have gained widespread adoption, they suffer from limitations in terms of their ability to guarantee desirable quality properties such as *soundness* (i.e., the absence of deadlocks and other anomalies).

The recently introduced Partially Ordered Workflow Language (POWL) [15] addresses these limitations by providing a powerful yet formally sound framework for process modeling. POWL is a hierarchical modeling language that allows for the construction of complex process models by combining smaller submodels using a set of well-defined operators. These operators include exclusive choice



(a) A POWL model. (b) A WF-net.

Fig. 1: Example process models.

(XOR), loop, and partial order. POWL can be viewed as a generalization of process trees [19], a notation widely used in various process mining techniques due to its quality guarantees. POWL preserves these desirable properties while increasing expressiveness through the partial order operator. Partial orders allow for the representation of activities that can be executed concurrently, but may have some ordering restrictions. Fig. 1a illustrates an example POWL model, and Fig. 1b shows a WF-net that captures the same behavior.

The hierarchical nature of POWL offers several advantages. The structured representation can significantly enhance the understandability of complex process models for humans, making it easier to grasp the overall control flow and identify potential areas for improvement. Furthermore, POWL opens up opportunities for developing faster and more efficient techniques for different process mining tasks, similar to how specialized algorithms have been optimized for process trees. Previous work has demonstrated the advantages of POWL in process discovery from data [14,16] and process modeling from text [12].

Despite its demonstrated benefits, the practical adoption of POWL is challenged by the prevalence of standard notations in existing tools and workflows. This motivates the need for a robust conversion from standard notations to POWL, allowing us to harness these benefits without requiring the adjustment of existing tools and practices. While the conversion from POWL to a sound WF-net is relatively straightforward, the inverse transformation presents a more significant challenge. This asymmetry arises from the fact that POWL represents a subclass of sound WF-nets.

This paper proposes an algorithm that translates sound WF-nets into POWL. The proposed algorithm recursively decomposes the input WF-net into its smaller parts, identifies structural patterns corresponding to POWL's constructs, and assembles them into an equivalent POWL model. We formally prove the correctness of this transformation, demonstrating that the generated POWL model captures the language of the original WF-net. Furthermore, we define a subclass of WF-nets that encompasses equivalent representations for all POWL models, and we show the completeness of our approach on this subclass. We implement our algorithm and apply it on large WF-nets to demonstrates its high scalability. This work represents a significant step towards enabling the development of a process analysis and improvement techniques that utilize POWL internally while seamlessly accepting inputs in widely used formats.

The remainder of the paper is structured as follows. After discussing related work in Sec. 2, we introduce necessary preliminaries in Sec. 3. In Sec. 4, we

detail our algorithm for translating WF-nets into POWL. Sec. 5 formally proves the algorithm's correctness and completeness guarantees. Finally, we assess the scalabilty of our approach in Sec. 6, and we conclude the paper in Sec. 7.

2 Related Work

Transformations between process modeling notations have been explored in various contexts. Some research focuses on transformations between different Petri net classes, such as the work on unfolding Colored Petri Nets into standard Place/Transition nets in [4] and the work on reducing free-choice Petri nets to either T-nets (also called marked graphs) or P-nets (also called state machines) in [24]. Other research addresses transformations between different types of process models. For example, [6] proposes an approach for converting BPMN models into Petri nets, [8] discusses translating UML diagrams into BPEL, and [17] explores the mapping of Event-driven Process Chains (EPCs) into colored Petri nets. An overview on different approaches for the translation between workflow graphs and free-choice workflow nets is provided in [7].

Transformations from graph-based formalisms like Petri nets into block-structured languages such as BPEL or process trees have been widely studied. The work on translating WF-nets to BPEL in [25,18] employs a bottom-up strategy, iteratively identifying patterns corresponding to BPEL fragments and substituting each identified pattern with a single transition to continue the recursion. This approach aims to maximize the size of the detected components in each iteration. The approach presented in [27] for translating WF-nets into process trees, while also employing a bottom-up strategy, restricts the search space to patterns of size two. This approach cannot be adapted to POWL due to the presence of advanced partial order constructs that cannot be decomposed into components of size two. The fundamental difference between our algorithm and the aforementioned approaches, besides using different modeling languages, is that our approach employs a top-down strategy to ensure high scalability.

3 Preliminaries

This section introduces fundamental preliminaries and notations.

3.1 Basic Notations

A *multi-set* generalizes the notion of a set by tracking the frequencies of its elements. A multi-set over a set X is expressed as $M = [{x_1}^{c_1}, ..., {x_n}^{c_n}]$ where $x_1, ..., x_n \in X$ are the elements of M (denoted as $x_i \in M$ for $1 \leq i \leq n$) and $M(x_i) = c_i \geq 1$ is the frequency of x_i for $1 \leq i \leq n$.

A sequence of length $n \geq 0$ over a set X is defined as function $\sigma \colon \{1, ..., n\} \to X$, and we express it as $\sigma = \langle \sigma(1), ..., \sigma(n) \rangle$. The set of all sequences over X is denoted by X^*. The concatenation of two sequences σ_1 and σ_2 is expressed as

$\sigma_1 \cdot \sigma_2$, e.g., $\langle x_1 \rangle \cdot \langle x_2, x_1 \rangle = \langle x_1, x_2, x_1 \rangle$. For two sets of sequences L_1 and L_2, we write $L_1 \cdot L_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in L_1 \ \wedge \ \sigma_2 \in L_2\}$.

Let $\prec \subseteq X \times X$ be a binary relation over a set X. We use $x_1 \prec x_2$ to denote $(x_1, x_2) \in \prec$ and $x_1 \not\prec x_2$ to denote $(x_1, x_2) \notin \prec$. We define the *transitive closure* of $\prec$ as $\prec^+ = \{(x, y) \mid \exists_{x_1,...,x_n \in X} \ x = x_1 \ \wedge \ y = x_n \ \wedge \forall_{1 \leq i < n}, x_i \prec x_{i+1}\}$.

A *strict partial order* (*partial order* for short) over a set X is a binary relation that is *irreflexive* ($x \not\prec x$ for all $x \in X$) and *transitive* ($x_1 \prec x_2 \wedge x_2 \prec x_3 \Rightarrow x_1 \prec x_3$). Irreflexivity and transitivity imply *asymmetry* ($x_1 \prec x_2 \Rightarrow x_2 \not\prec x_1$). For $n \geq 2$, we use $\mathcal{O}^n$ to denote the set of all partial orders over $\{1, \ldots, n\}$. Let $X = \{x_1, ..., x_n\}$ be a set of size $n \geq 2$ and $\prec \in \mathcal{O}^n$. Then we write $\prec(x_1, ..., x_n)$ to denote the partial order $\prec'$ defined over X as follows: $i \prec j \Leftrightarrow x_i \prec' x_j$ for all $i, j \in \{1, \ldots, n\}$.

Let $\sigma_1, ..., \sigma_n \in X^*$ be $n \geq 2$ sequences over a set X and $\prec \in \mathcal{O}^n$. The *order-preserving shuffle operator* $\sqcup\!\sqcup_\prec$ generates the set of sequences resulting from interleaving $\sigma_1, ..., \sigma_n$ while preserving the partial order $\prec$ of the sequences and the sequential order within each sequence. For example, let $\sigma_1 = \langle a, b \rangle$, $\sigma_2 = \langle c \rangle$, $\sigma_3 = \langle d, e \rangle$, and $\prec = \{(1, 2), (1, 3)\} \in \mathcal{O}^3$. Then, $\sqcup\!\sqcup_\prec(\sigma_1, \sigma_2, \sigma_3) = \{\langle a, b, c, d, e \rangle, \langle a, b, d, c, e \rangle, \langle a, b, d, e, c \rangle\}$.

For a set X, a *partition* of X of size $n \geq 1$ is a set of subsets $P = \{X_1, ..., X_n\}$ such that $X = X_1 \cup ... \cup X_n$, $X_i \neq \emptyset$, and $X_i \cap X_j = \emptyset$ for $1 \leq i < j \leq n$. For any $x \in X$, we write P_x to denote the subset of the partition (also called *part*) that contains x, i.e., $P_x \in \{X_1, ..., X_n\}$ and $x \in P_x$. For example, let $P = \{\{a, b\}, \{c\}\}$ be a partition of $\{a, b, c\}$ of size 2. Then, $P_a = P_b = \{a, b\}$ and $P_c = \{c\}$.

3.2 Workflow Nets

We use Σ to denote the set of all activities. We use $\tau \notin \Sigma$ to denote the *silent activity*, which is used to model a choice between executing or skipping a path in process model, for example. To enable creating process models with duplicated activities, we introduce the notion of *transitions*, and we use $\mathcal{T}$ to denote the set of all transitions. Each transition is mapped to an activity, denoted as the *label* of the transition. We use $l \colon \mathcal{T} \to \Sigma \cup \{\tau\}$ to denote the labeling function.

A *Petri net* is a directed bipartite graph consisting of two types of nodes: *places* and *transitions*. Transitions represent instances of activities, while places are used to model dependencies between transitions.

Definition 1 (Petri Net). *A Petri net is a triple $N = (P, T, F)$, where $T \subset \mathcal{T}$ is a finite set of transitions, P is a finite set of places such that $T \cap P = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.*

Let $N = (P, T, F)$ be a Petri net. We define the following notations:

- For $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ is the *pre-set* of x, and $x \bullet = \{y \mid (x, y) \in F\}$ is the *post-set* of x.
- For $T' \subseteq T$, we define the *projection* of P on T' as $P|_{T'} = \{p \in P \mid (\bullet p \cup p \bullet) \cap T' \neq \emptyset\}$.
- For $P' \subseteq P$ and $T' \subseteq T$, we define the *projection* of F on P and T as $F|_{P', T'} = F \cap ((P' \times T') \cup (T' \times P'))$.

- For $T' \subseteq T$, two places $p, p' \in P$ are *equivalent with respect to* T', denoted as $p \approx_{T'} p'$, iff $(\bullet p \cap T' = \bullet p' \cap T') \ \wedge \ (p\bullet \cap T' = p'\bullet \cap T')$.

Places hold *tokens*, and a transition is considered *enabled* if each of its preceding places has at least one token. *Firing* an enabled transition consumes one token from each of its preceding places and produces a token in each of its succeeding places. A *marking* is a multi-set of places indicating the number of tokens in each place. A Petri net is called *safe* if each place in the net cannot hold more than one token. Next, we define three subclasses of Petri nets. More details on these classes can be found in [21,5,22].

Definition 2 (Marked Graph, Free-Choiceness, Workflow Net). *Let* $N = (P, T, F)$ *be a Petri net. Then, the following holds:*

- N *is a marked graph iff for any* $p \in P$: $|\bullet p| \leq 1 \ \wedge \ |p\bullet| \leq 1$.
- N *is free-choice iff for any* $t_1, t_2 \in T$: $(\bullet t_1 \cap \bullet t_2 \neq \emptyset) \Rightarrow (\bullet t_1 = \bullet t_2)$.
- N *is a workflow net (WF-net) iff places* $N_{source}, N_{sink} \in P$ *exist such that:*
  - ***Unique source:*** $\{N_{source}\} = \{p \in P \mid \bullet p = \emptyset\}$.
  - ***Unique sink:*** $\{N_{sink}\} = \{p \in P \mid p\bullet = \emptyset\}$.
  - ***Connectivity:*** *each node is on a path from* N_{source} *to* N_{sink}.

Let $N = (P, T, F)$ be a WF-net. We use N_{source} to denote the unique source place and N_{sink} to denote the unique sink place. Furthermore, we define:

- For $T' \subseteq T$, $\triangleright T' = \{p \in P \mid T' \cap p\bullet \neq \emptyset \ \wedge \ (p = N_{source} \ \vee \ (T \setminus T') \cap \bullet p \neq \emptyset)\}$ is the set of *entry points* of T'.
- For $T' \subseteq T$, $T'\triangleright = \{p \in P \mid T' \cap \bullet p \neq \emptyset \ \wedge \ (p = N_{sink} \ \vee \ (T \setminus T') \cap p\bullet \neq \emptyset)\}$ is the set of *exit points* of T'.
- For a partition $G = \{T_1, \ldots, T_n\}$ of T, the *execution order* of G within N is the binary relation $order(N, G) = \{(i, j) \mid i, j \in \{1, \ldots, n\} \wedge (T_i\triangleright \cap \triangleright T_j) \neq \emptyset\}$.

WF-nets may suffer from quality anomalies (e.g., transitions that can never be enabled). WF-nets without such undesirable properties are called *sound*.

Definition 3 (Soundness). *Let* $N = (P, T, F)$ *be a WF-net.* N *is sound iff the following conditions hold:*

- ***No dead transitions:*** *for each transition* $t \in T$*, there exists a marking* M *reachable from* $[N_{source}]$ *that enables* t.
- ***Option to complete:*** *for every marking* M *reachable from* $[N_{source}]$*, there exists a firing sequence leading from* M *to* $[N_{sink}]$.
- ***Proper completion:*** $[N_{sink}]$ *is the only marking reachable from* $[N_{source}]$ *with at least one token in* N_{sink}.

The WF-net shown in Fig. 1b is sound. Note that the option to complete implies proper completion.

3.3 POWL Language

A POWL model is constructed recursively from a set of activities, combined either as partial orders or using the control flow operators $\times$ and $\circlearrowleft$. The operator

$\times$ models an exclusive choice between submodels, while $\circlearrowleft$ model cyclic behavior between two submodels: the *do-part* is executed first, and each time the *redo-part* is executed, it is followed by another execution of the do-part. In a partial order, all submodels are executed, while respecting the given execution order.

POWL models are defined in [15] over activities. We redefine POWL models over transitions to allow for models with multiple instances of the same activity.

Definition 4 (POWL Model). *POWL models are defined as follows:*

- *Any transition $t \in \mathcal{T}$ is a POWL model.*
- *Let $\psi_1, ..., \psi_n$ be $n \geq 2$ POWL models.*
  - *$\times(\psi_1, ..., \psi_n)$ is a POWL model.*
  - *$\circlearrowleft(\psi_1, \psi_2)$ is a POWL model.*
  - *For any partial order $\prec \in \mathcal{O}^n$, $\prec(\psi_1, ..., \psi_n)$ is a POWL model.*

Fig. 1a shows an example POWL model. The language a POWL model is defined recursively based on the semantics of its operators.

Definition 5 (POWL Semantics). *The language of a POWL model ψ is recursively defined as follows:*

- *$\mathcal{L}(t) = \{\langle a \rangle\}$ for $t \in \mathcal{T}$ with $l(t) = a \in \Sigma$.*
- *$\mathcal{L}(t) = \{\langle\rangle\}$ for $t \in \mathcal{T}$ with $l(t) = \tau$.*
- *Let $\psi_1, ..., \psi_n$ be $n \geq 2$ POWL models with $L_i = \mathcal{L}(\psi_i)$ for $1 \leq i \leq n$.*
  - *$\mathcal{L}(\times(\psi_1, ..., \psi_n)) = \bigcup_{1 \leq i \leq n} L_i$.*
  - *$\mathcal{L}(\circlearrowleft(\psi_1, \psi_2)) = L_1 \cdot (L_2 \cdot L_1)^*$.*
  - *For $\prec \in \mathcal{O}^n$, $\mathcal{L}(\prec(\psi_1, ..., \psi_n)) = \{\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n) \mid \forall_{1 \leq i \leq n}\ \sigma_i \in L_i\}$.*

3.4 Semi-Block-Structured WF-nets

POWL is less expressive than WF-nets, meaning that not all WF-nets have equivalent POWL models. To clearly define the scope of our WF-net to POWL translation algorithm, we extend the concept of *block-structured* workflow nets defined in [19]. A block-structured WF-net is a sound WF-net that can be divided recursively into parts having single entry and exit points. In other words, there must be a unique mapping between every place/transition with multiple outgoing arcs and every place/transition with multiple incoming arcs to mark the start and end of a block.

POWL can represent all block-structured WF-nets, and the introduction of the partial order operator in POWL extends its expressiveness beyond this class. We can relax the block-structure requirements since the partial order operator allows for the concurrent executions of transitions without imposing block structure on them. In other words, the block structure is only required for decision points (i.e., places) due to the usage of the process tree operators $\times$ and $\circlearrowleft$.

Definition 6 (Semi-Block-Structured WF-nets). *Let $N = (P, T, F)$ be a WF-net. N is* semi-block-structured *iff it satisfies the following conditions:*

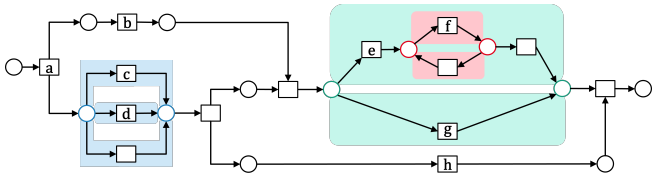


Fig. 2: A semi-block-structured WF-net with three blocks, highlighted in different colors.

- *N is sound and safe.*
- ***Explicit decision points:*** *For each $(x, y) \in F : |\bullet y| = 1 \vee |x\bullet| = 1$.*
- ***Unique mapping between split and join decision points:*** *Let $P_{split} = \{p \in P \mid |p\bullet| > 1\}$ and $P_{join} = \{p \in P \mid |\bullet p| > 1\}$. There exists a bijective mapping $\mathcal{B} : P_{split} \to P_{join}$ such that for each $(p, p') \in \mathcal{B}$: $|p\bullet| = |\bullet p'|$.*
- ***Disjoint subnets between decision points:*** *For each pair $(p, p') \in \mathcal{B}$, let $k = |p\bullet| = |\bullet p'|$. There exist k disjoint, non-empty sets of transitions $T_1, \ldots, T_k \subseteq T$ such that for each i $(1 \leq i \leq k)$, $P_i = P|_{T_i}$, and $F_i = F|_{P_i, T_i}$:*
  - *$P_i \cap P|_{T \setminus T_i} = \{p, p'\}$.*
  - *$N_i = (P_i, T_i, F_i)$ is a workflow net with $\{N_{i_{source}}, N_{i_{sink}}\} = \{p, p'\}$.*

The WF-net in Fig. 2 is semi-block-structured WF-net but not block structured. Note that a semi-block-structured WF-net with no blocks is a marked graph. Furthermore, a semi-block-structured WF-net is free-choice due to the explicit decision points requirement. Therefore, we can conclude that semi-block-structured WF-nets are a subset of sound free-choice WF-nets, a superset of block-structured WF-nets, and superset of sound marked graph WF-nets.

It is trivial to observe that, for any POWL model, there exists at least one equivalent semi-block-structured WF-net. Furthermore, after introducing our conversion algorithm, we will show in Sec. 5.4 its completeness on this class of WF-nets; i.e., we will show the our algorithm successfully converts any semi-block-structured WF-net into a POWL model that captures the same language.

4 Transforming Workflow Nets into POWL

This section presents a recursive algorithm for transforming safe and sound WF-nets into equivalent POWL models. First, the WF-net can be preprocessed by applying a set of reduction rules. Then, the algorithm checks whether the WF-net forms a structural pattern that corresponds to a POWL component (i.e., choice, loop, or partial order). The identified pattern is then translated into its corresponding POWL representation, and the WF-net is projected onto subsets of transitions to continue the recursion.

4.1 Preprocessing

To expand the applicability of our algorithm, the input WF-net can be preprocessed by applying a series of reduction rules. This is an optional step, aiming

(a) Introducing explicit XOR split places. (b) Introducing explicit XOR join places.

Fig. 3: Reduction rules illustrated with examples.

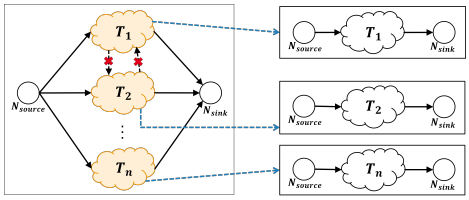


Fig. 4: XOR pattern and projection.

at bringing the WF-net closer to the structure expected in the subsequent steps of the algorithm. Numerous reduction rules have been proposed in the literature, such as those presented in [5,1,2,20]. Any reduction rule can be applied as long as it preserves the essential structural properties of the WF-net, namely its language, safeness, and soundness. In Fig. 3, we illustrate reduction rules for introducing explicit places for decision points. Additional reduction rules are introduced in Sec. 4.3 to enable the detection of special loop structures.

4.2 Identifying Choices

This section addresses partitioning a WF-net into smaller subnets such that the given WF-net corresponds to an XOR structure over the identified parts.

We first introduce the concept of *transition reachability*, capturing the notion of one transition being reachable from another through a path in the WF-net.

Definition 7 (Transition Reachability). *Let $N = (P, T, F)$ be a Petri net. The* transition reachability relation $\rightsquigarrow \subseteq T \times T$ *is defined as follows for $t, t' \in T$:*

$$t \rightsquigarrow t' \Leftrightarrow (t, t') \in F^+.$$

An *XOR pattern*, as illustrated in Fig. 4, represents a WF-net where transitions can be partitioned such that exactly one part can be executed in a single process instance. Intuitively, transitions belonging to different parts cannot be reachable from each other.

Definition 8 (XOR Pattern). *Let $N = (P, T, F)$ be a safe and sound WF-net. Let $G = \{T_1, \ldots, T_n\}$ be a partition of transitions of size $n \geq 2$. The tuple (N, G) is called an XOR pattern iff for all $(t, t') \in T \times T$:*

$$\text{if } t \rightsquigarrow t', \text{ then } G_t = G_{t'}.$$

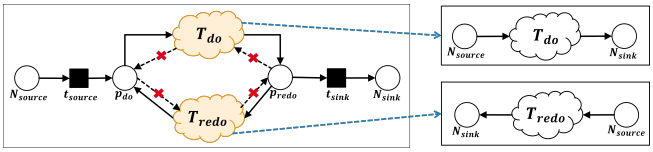


Fig. 5: Loop pattern and projection.

For a WF-net N, we use $Partition_{\times}(N)$ to denote the partition generated by iteratively grouping transitions based on their reachability relation, aligning with Def. 8. Note that $(N, Partition_{\times}(N))$ is an XOR pattern if $|Partition_{\times}(N)| \geq 2$.

After identifying an XOR pattern, the WF-net is projected onto the different parts, creating several subnets for the recursive application of the algorithm. The projection is achieved by selecting the relevant places and flow relations, isolating the chosen part.

Definition 9 (XOR Projection). *Let (N, G) be an XOR pattern with $N = (P, T, F)$ and $T' \in G$ be a part. The XOR projection of N on T' is defined as $Project_{\times}(N, T') = (P', T', F')$ where $P' = P|_{T'}$ and $F' = F|_{P', T'}$.*

4.3 Identifying Loops

This section addresses partitioning a WF-net into subnets such that the given WF-net corresponds to a loop structure over the identified do- and redo-parts.

We first define the concept of *in-between places reachability* to identify the transitions that lie on paths between two specific places in a WF-net.

Definition 10 (In-Between Places Reachability). *Let $N = (P, T, F)$ be a Petri net. The set of reachable transitions between two places $p, p' \in P$, denoted as $R_{PP}(p, p')$, is defined as follows:*

$$t \in R_{PP}(p, p') \Leftrightarrow \exists t_1, \ldots, t_n \in T \text{ and } p_1, \ldots, p_{n+1} \in P$$

such that

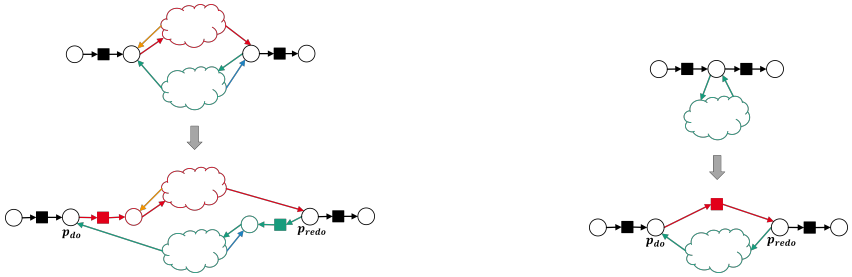
$$p_1 = p \ \wedge \ p_{n+1} = p' \ \wedge \ t \in \{t_1, \ldots, t_n\},$$

and for each i $(1 \leq i \leq n)$:

$$p_i \neq p' \ \wedge \ (p_i, t_i) \in F \ \wedge \ (t_i, p_{i+1}) \in F.$$

An *loop pattern*, as illustrated in Fig. 5, represents a WF-net where transitions can be partitioned into three parts: do-part, redo-part, and silent transitions for loop entry and exit. A loop pattern must also include two special places, p_{do} and p_{redo}, which mark the entry and exit points of the loop. The do-part consists of transitions reachable from p_{do} to p_{redo}, while the redo-part consists of transitions reachable from p_{redo} to p_{do}.

Definition 11 (Loop Pattern). *Let $N = (P, T, F)$ be a safe and sound WF-net. Let $G = \{T_{do}, T_{redo}, T_{\tau}\}$ be a partition of transitions of size 3. The tuple (N, G) is called a loop pattern iff places $p_{do}, p_{redo} \in P$ exist such that:*



(a) Satisfying the requirement *6* of Def. 11. (b) Handling self-loops.

Fig. 6: Reduction rules for enabling loop pattern detection.

1. *Two transitions* t_{source} *and* t_{sink} *exist such that* $t_{source} \neq t_{sink}$, $T_\tau = \{t_{source}, t_{sink}\}$, *and* $\mathcal{L}(t_{source}) = \mathcal{L}(t_{sink}) = \tau$.
2. $N_{source}\bullet = \{t_{source}\}$ *and* $\bullet N_{sink} = \{t_{sink}\}$.
3. $\bullet t_{source} = \{N_{source}\}$ *and* $t_{source}\bullet = \{p_{do}\}$.
4. $\bullet t_{sink} = \{p_{redo}\}$ *and* $t_{sink}\bullet = \{N_{sink}\}$.
5. $T_{do} = R_{PP}(p_{do}, p_{redo})$ *and* $T_{redo} = R_{PP}(p_{redo}, p_{do})$.
6. $\bullet p_{do} \cap T_{do} = \emptyset$ *and* $\bullet p_{redo} \cap T_{redo} = \emptyset$.
7. $p_{redo}\bullet \cap T_{do} = \emptyset$ *and* $p_{do}\bullet \cap T_{redo} = \emptyset$.

Note that a WF-net satisfying the requirements *1* - *5* of Def. 11 can be preprocessed, as illustrated in Fig. 6a, to satisfy *6* without affecting its language, soundness, or safeness. Furthermore, the requirement *7* is implied by *1* - *5*.

Identifying Self-Loops with Missing Do-Part. A WF-net may represent a self-loop with a single place marking both the start and end of the loop. In such cases, all activities belong to the redo-part, which can be skipped or repeated. To enable the detection of these loops, we restructure the WF-net, as illustrated in Fig. 6b, by adding a silent transition to represent the do-part.

After detecting a loop pattern, the WF-net is projected into the do-part and redo-parts, creating two subnets, and the recursion continues on the created subnets. As illustrated in Fig. 5, the loop projection is done by selecting the appropriate subset of transitions and adjusting the flow relation to correctly connect the subnet to the source and sink place.

Definition 12 (Loop Projection). *Let* (N, G) *be an loop pattern with* $N = (P, T, F)$; $T_{do}, T_{redo} \in G$; *and* $p_{do}, p_{redo} \in P$ *as defined in Def. 11. The loop projection of* N *on* $T' \in \{T_{do}, T_{redo}\}$ *is* $Project_{\circlearrowleft}(N, T') = (P', T', F')$ *where*

$$P' = (P|_{T'} \setminus \{p_{do}, p_{redo}\}) \cup \{N_{source}, N_{sink}\}$$

and

$$F' = F|_{P',T'} \cup \{(N_{source}, t) \mid t \in T' \wedge (p_{start}, t) \in F\}$$
$$\cup \{(t, N_{sink}) \mid t \in T' \wedge (t, p_{end}) \in F\}$$

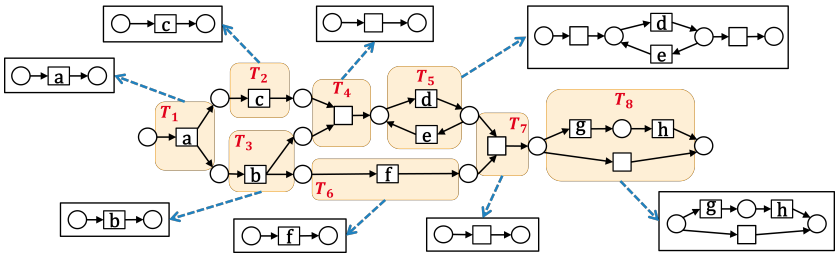


Fig. 7: An example illustrating a partial order pattern $(N, \{T_1, \ldots, T_8\})$ and the partial order projection of the WF-net N on the identified parts.

with

$$(p_{start}, p_{end}) = \begin{cases} (p_{do}, p_{redo}) & \text{if } T' = T_{do}, \\ (p_{redo}, p_{do}) & \text{if } T' = T_{redo}. \end{cases}$$

4.4 Identifying Partial Orders

This section addresses partitioning a WF-net into smaller subnets such that the given WF-net corresponds to a partial order over the identified parts.

A *partial order pattern* represents a WF-net with a partition of transitions such that all parts are executed and the execution order forms a partial order.

Definition 13 (Partial Order Pattern). *Let $N = (P, T, F)$ be a safe and sound WF-net. Let $G = \{T_1, \ldots, T_n\}$ be a partition of transitions of size $n \geq 2$. The tuple (N, G) is called a partial order pattern iff the following conditions hold:*

1. *For all $t, t' \in T$ and $p \in P$:*
$$\text{if } t, t' \in \{t \in T \mid \exists_{t_1, t_2 \in p^\bullet}\ t_1 \rightsquigarrow t \wedge t_2 \not\rightsquigarrow t\}, \text{ then } G_t = G_{t'}.$$
2. *$\prec = order(N, G)^+$ is a partial order.*
3. ***Unique local start:** for all $i \in \{1, \ldots, n\}$ and $p, p' \in {\triangleright}T_i$: $p \approx_{T_i} p'$.*
4. ***Unique local end:** for all $i \in \{1, \ldots, n\}$ and $p, p' \in T_i{\triangleright}$: $p \approx_{T_i} p'$.*

The first condition of Def. 13 states that if a place has outgoing flows leading to different transitions, then all such transitions must fall into the same part, as the place represents a decision point (i.e., there is a potential choice or loop within this part). The second condition requires that the transitive closure of the execution order between the parts of the partition forms a partial order. The third and fourth conditions ensure that the identified parts form cleanly separable components that can be executed independently with well-defined entry and exit points. Fig. 7 shows a partial order pattern detected for the WF-net from Fig. 1b.

For a WF-net N, we use $Partition_{order}(N)$ to denote the partition generated by iteratively grouping transitions based on their reachability relations with respect to decision points, aligning with the first condition of Def. 13. Note that $(N, Partition_{order}(N))$ is a partial order pattern if $|Partition_{order}(N)| \geq 2$ and the remaining three requirements of Def. 13 are met.

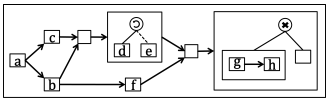


Fig. 8: The POWL model generated by Alg. 1 for the WF-net from Fig. 1b.

Normalization. Before defining the projection for partial order patterns, we introduce the concept of normalization. Let N be a Petri net with known unique start and end places $p_s \in P$ and $p_e \in P$, respectively. Then P is normalized into a new Petri net $Normalize(N, p_s, p_e)$ by (i) inserting a new start place and connecting it to p_s through a silent transition in case $\bullet p_s \neq \emptyset$ and (ii) inserting a new end place and connecting it to p_e through a silent transition in case $p_e\bullet \neq \emptyset$. Normalization aims at ensuring conformance with the requirements of WF-nets (c.f. Def. 2) by adding new source and sink places if needed.

After detecting a partial order pattern, the WF-net is projected on the identified parts as illustrated in the example shown in Fig. 7. This projection is done by selecting the appropriate subset of transitions, adding unique start and end places to represent the entry and exit points of the part, adjusting the flow relation accordingly, and applying normalization if needed.

Definition 14 (Partial Order Projection). *Let (N, G) be a partial order pattern with $N = (P, T, F)$. Let $T' \in G$ be a part. The partial order projection of N on T' is $Project_{order}(N, T') = Normalize(N', p_s, p_e)$ where $p_s, p_e \notin P$ are two fresh places and $N' = (P', T', F')$ is constructed as follows:*

- $P' = (P|_{T'} \setminus (\triangleright T' \cup T' \triangleright)) \cup \{p_s, p_e\}$.
- $F' = F|_{P', T'}$
  $\cup \{(p_s, t) \mid (p, t) \in F \text{ for } p \in \triangleright T'\} \cup \{(t, p_s) \mid (t, p) \in F \text{ for } p \in \triangleright T'\}$
  $\cup \{(p_e, t) \mid (p, t) \in F \text{ for } p \in T' \triangleright\} \cup \{(t, p_e) \mid (t, p) \in F \text{ for } p \in T' \triangleright\}$.

4.5 WF-Net to POWL Converter

Alg. 1 converts a safe and sound WF-net into an equivalent POWL model. First, the algorithm checks whether the WF-net consists of a single transition (base case). If a base case is not detected, the algorithm attempts to identify an XOR, loop, or partial order pattern. If a pattern is found, the algorithm projects the WF-net on the identified parts, recursively converts the created subnets into POWL models, and combines them using the appropriate POWL operator. If no pattern is detected, the algorithm returns *null*, indicating that the WF-net cannot be converted into a POWL model. Note that the order in which the algorithm searches for the different patterns is irrelevant, as at most one pattern can match the WF-net's structure at a time.

Example Applications: Fig. 8 shows the POWL model generated by applying Alg. 1 on the WF-net from Fig. 1b. The generated POWL model, while structurally different from the manually crafted POWL model in Fig. 1a, is semantically equivalent. Fig. 9 illustrates three examples where Alg. 1 returns null:

Input: A safe and sound WF-net $N = (P, T, F)$.
Output: A POWL model or *null* if no translation is possible.

Function `ConvertNetToPOWL(`N`)`**:**
  if $|T| = 1$ *with* $T = \{t\}$, $|P| = 2$, *and* $F = \{(N_{source}, t), (t, N_{sink})\}$ **then**
    return t
  $G \leftarrow Partition_{\times}(N)$
  if (N, G) *is an XOR pattern* **then**
    for $T_i \in G = \{T_1, \ldots, T_n\}$ **do**
      $\psi_i \leftarrow$ `ConvertNetToPOWL(`$Project_{\times}(N, T_i)$`)`
    return $\times(\psi_1, \ldots, \psi_n)$
  if *a loop pattern exists with* $p_{do}, p_{redo} \in P$ *as described in Def. 11* **then**
    $\psi_{do} \leftarrow$ `ConvertNetToPOWL(`$Project_{\circlearrowleft}(N, R_{PP}(p_{do}, p_{redo}))$`)`
    $\psi_{redo} \leftarrow$ `ConvertNetToPOWL(`$Project_{\circlearrowleft}(N, R_{PP}(p_{redo}, p_{do}))$`)`
    return $\circlearrowleft(\psi_{do}, \psi_{redo})$
  $G \leftarrow Partition_{order}(N)$
  if (N, G) *is a partial order pattern* **then**
    $\prec \leftarrow order(N, G)^{+}$
    for $T_i \in G = \{T_1, \ldots, T_n\}$ **do**
      $\psi_i \leftarrow$ `ConvertNetToPOWL(`$Project_{order}(N, T_i)$`)`
    return $\prec(\psi_1, \ldots, \psi_n)$
  return *null*

Algorithm 1: Conversion of a WF-Net into a POWL Model.

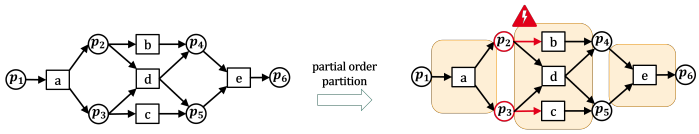
- The first WF-net (Fig. 9b), while free-choice, its decision points are not arranged in a block structure, and no equivalent POWL model exists for its language. The algorithm attempts to find XOR or partial order patterns but ultimately produces partitions of size 1, leading to the return of null.
- The second WF-net (Fig. 9b) exhibits a choice between activities a and b, followed by a non-free-choice between d and e that is influenced by the preceding choice. This long-term dependency choice cannot be represented in POWL. Alg. 1 attempts to identify a partial order pattern, resulting in a partition that violates the requirements of Def. 13, e.g., no unique local end in the first part $T_1 = \{a, b\}$ since $\bullet p_3 \cap T_1 = \{a\} \neq \{b\} = \bullet p_4 \cap T_1$.
- In the third WF-net (Fig. 9c), the places p_2 and p_3 model a choice between d or executing both b and c concurrently. When attempting to identify a partial order pattern, the requirements are violated, e.g., no unique local start in the second part $T_2 = \{b, c, d\}$ since $p_2\bullet \cap T_2 = \{b, d\} \neq \{c, d\} = p_3\bullet \cap T_2$. However, applying the reduction rules from Fig. 3 before applying Alg. 1 enables the successful conversion into POWL, as illustrated in Fig. 10.

5 Correctness and Completeness Guarantees

In this section, we prove the correctness and completeness guarantees of Alg. 1. Our proof strategy for correctness relies on structural induction on the WF-net. We show that for each pattern (XOR, loop, and partial order), the projection operation preserves safeness and soundness, allowing for the recursive application of the algorithm on the created subnets (Sec. 5.1). Then, we demonstrate that combining the languages of these subnets using the respective POWL operators

(a) Free-choice WF-net with non-block-structured decision points.

(b) Non-free-choice WF-net with a long-term dependency between choices.



(c) Non-free-choice WF-net where decision points combine choice with concurrency.

Fig. 9: Examples where Alg. 1 returns null.

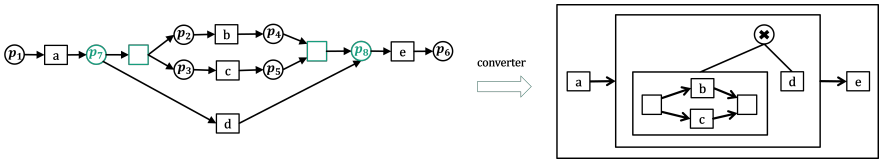


Fig. 10: Result of applying the reduction rules (c.f. Fig. 3) to the WF-net from Fig. 9c, followed by a successful conversion into POWL using Alg. 1.

accurately reflects the language of the original WF-net (Sec. 5.2). We combine these findings to prove the overall correctness of the algorithm (Sec. 5.3). Finally, we show the completeness of our algorithm on semi-block-structured WF-nets (Sec. 5.4).

5.1 Projection Structural Guarantees

This section proves that the XOR, loop, and partial order projections, when applied to safe and sound WF-nets, produce safe and sound WF-nets. This ensures that the recursive calls in Alg. 1 are always applied to valid inputs.

Lemma 1 (XOR Projection Structural Guarantees). *Let (N, G) be an XOR pattern with $N = (P, T, F)$ and $G = \{T_1, \ldots, T_n\}$. Let $N_i = (P_i, T_i, F_i) = Project_{\times}(N, T_i)$ for each i $(1 \leq i \leq n)$. Then N_i is a safe and sound WF-net.*

Proof. **(1) N_i is a WF-net:** By construction, no place $p \in P_i \setminus \{N_{source}, N_{sink}\}$ is connecting transitions from both T_i and $T \setminus T_i$ in N. Therefore, all transitions and places in N_i remain connected on a path from N_{source} to N_{sink}.

(2) We prove that N_i is safe and sound. N_i can be replaced by a silent transition $t^+ \notin T$ in N generating another WF-net $N' = (P', T', F')$ as follows:

- $T' = (T \setminus T_i) \cup \{t^+\}$.
- $P' = P|_{T \setminus T_i}$.
- $F' = F|_{P', T \setminus T_i} \cup \{(N_{source}, t^+), (t^+, N_{sink})\}$.

N can be reconstructed from N' as described in [23, Theorem 3.4] by replacing t^+ by N_i if $P_i \cap P' = \{N_{source}, N_{sink}\}$. This condition is satisfied since no place connecting transitions from both T_i and $T \setminus T_i$ exists in N. Therefore, N_i is safe and sound by [23, Theorem 3.4].

Lemma 2 (Loop Projection Structural Guarantees). *Let (N, G) be an loop pattern with $N = (P, T, F)$; $T_{do}, T_{redo} \in G$; and $p_{do}, p_{redo} \in P$ as defined in Def. 11. Let $N_{do} = (P_{do}, T_{do}, F_{do}) = Project_{\circlearrowleft}(N, T_{do})$ and $N_{redo} = (P_{redo}, T_{redo}, F_{redo}) = Project_{\circlearrowleft}(N, T_{redo})$. Then N_{do} and N_{redo} are safe and sound WF-nets.*

Proof. **(1) We show that N_{do} is a WF-net (analogous proof for N_{redo}).**

(1.1) Unique source and sink: Assume for contradiction that there exists a place $p \in P_{do} \setminus \{N_{source}, N_{sink}\}$ such that $\bullet p = \emptyset$ or $p\bullet = \emptyset$ in N_{do}. This place must be connecting transitions from both T_{do} and T_{redo} since $\bullet p \neq \emptyset$ and $p\bullet \neq \emptyset$ in N. Without loss of generality, assume that there exist transitions $t \in T_{do}$ and $t' \in T_{redo}$ where $(t, p) \in F$ and $(p, t') \in F$. Since $t \in T_{do} = R_{PP}(p_{do}, p_{redo})$, there exists a path starting from p_{do} to t, without passing through p_{redo} in-between. From t, the path can continue through p to reach t'. From t', we know that we can eventually reach p_{redo} (we reach p_{do} first and then take any path from p_{do} to p_{redo}). Combining all of these sequences ($p_{do} \to \ldots \to t \to p \to t' \to ... \to p_{redo}$) implies that $t' \in T_{do}$. This contradicts our assumption that $t' \in T_{redo}$.

(1.2) Connectivity: We showed that no place is connecting transitions from both T_{do} and T_{redo} except the loop entry and exit places. Therefore, all transitions and places in N_{do} remain connected on a path from N_{source} to N_{sink}.

(2) Safeness and soundness: The proof that N_{do} and N_{redo} are safe and sound is analogous to the safeness and soundness proof of Lemma 1.

Lemma 3 (Partial Order Projection Structural Guarantees). *Let (N, G) be a partial order pattern with $N = (P, T, F)$ and $G = \{T_1, \ldots, T_n\}$. Let $N_i = (P_i, T_i, F_i) = Project_{order}(N, T_i)$ for each i ($1 \leq i \leq n$). Then N_i is a safe and sound WF-net.*

Proof. **(1) N_i is a WF-net:** By construction, every node in N_i is on a path from p_s and p_e. The applied normalization ensures that additional source and/or sink places are inserted in case $\bullet p_s \neq \emptyset$ and/or $p_e\bullet \neq \emptyset$, respectively.

(2) We prove that N_i is sound.

(2.1) No dead transitions: Consider any transition $t \in T_i$. Since N is sound, there exists a reachable marking M in N that enables t. Consider the marking M_i in N_i defined as follows for $p \in P_i$:

$$M_i(p) = \begin{cases} M(p) & \text{if } p \in P, \\ 1 & \text{if } p = p_s \text{ and } M(p) = 1 \text{ for all } p \in {\triangleright}T_i, \\ 1 & \text{if } p = p_e \text{ and } M(p) = 1 \text{ for all } p \in T_i{\triangleright}, \\ 0 & \text{otherwise (for places added by normalization).} \end{cases}$$

The projection operation only removes places and transitions that are not in T_i and replaces the connections to $\triangleright T_i$ and $T_i \triangleright$ with p_s and p_e, respectively. Thus, any firing sequence leading to M in N can be transformed into a firing sequence leading to M_i in N_i by removing transitions not in T_i (and potentially firing the additional silent transition added by normalization). Therefore, M_i is reachable from $[N_{i_{source}}]$ in N_i. The unique local start and end properties (c.f. Def. 13) ensure that if t needs to consume a token from a place $p \in \triangleright T_i$ (or $p \in T_i \triangleright$) in N, then all other places in $\triangleright T_i$ (or in $T_i \triangleright$, respectively) must be included in M as well. This implies that M and M_i agree on all places that are needed to enable t, including start and end places. Therefore, t is enabled in M_i.

(2.2) Option to complete: Consider any marking M_i reachable from $[N_{i_{source}}]$ in N_i. Due to the unique local start and end properties (c.f. Def. 13), there must exist a reachable marking M in N that enables the transitions in T_i in the same way as M_i does in N_i, i.e., M is defined for $p \in P|_{T_i}$ as follows:

$$M(p) = \begin{cases} M_i(p_s) & \text{if } p \in \triangleright T_i, \\ M_i(p_e) & \text{if } p \in T_i \triangleright, \\ M_i(p) & \text{otherwise.} \end{cases}$$

Since N is sound, there exists a firing sequence σ from M to $[N_{sink}]$ in N. We can construct a corresponding firing sequence σ_i from M_i to $[N_{i_{sink}}]$ in N_i by taking only the transitions in σ that belong to T_i (and potentially firing the additional silent transition added by normalization).

(3) N_i is safe: Assume, for the sake of contradiction, that there exist a reachable marking M_i in N_i and a place $p \in P_i$ such that $M_i(p) \geq 2$. There must exist a reachable marking M in N that enables the transitions in T_i in the same way as M_i does in N_i (c.f. the proof of "option to complete"). Then there exists $p' \in P|_{T_i}$ such that $M(p') = M_i(p) \geq 2$. This violates the safeness of N.

5.2 Pattern Language Preservation Guarantees

This section establishes the language preservation guarantees of the identified patterns. We prove that the XOR, loop, and partial order patterns, when translated into their corresponding POWL representations, result in POWL models that have the same language as the original WF-net.

Lemma 4 (XOR Pattern Language Preservation). *Let (N, G) be an XOR pattern and $G = \{T_1, \ldots, T_n\}$. Let $\psi_1, \ldots, \psi_n$ be POWL models such that $\mathcal{L}(\psi_i) = \mathcal{L}(Project_{\times}(N, T_i))$ for each i $(1 \leq i \leq n)$. Then, $\mathcal{L}(\times(\psi_1, \ldots, \psi_n)) = \mathcal{L}(N)$.*

Proof. Since (N, G) forms an XOR Pattern, N enforces a choice of transitions from exactly one part $T_i \in G$ to be executed. $Project_{\times}(N, T_i)$ captures exactly the executions involving the transitions in T_i. Therefore, we conclude:

$$\mathcal{L}(N) = \bigcup_{i=1}^{n} \mathcal{L}(Project_{\times}(N, T_i)).$$

By combining this equality with Def. 5 and the assumption that $\mathcal{L}(\psi_i) = \mathcal{L}(Project_{\times}(N, T_i))$ for each $1 \leq i \leq n$, we get: $\mathcal{L}(N) = \mathcal{L}(\times(\psi_1, \ldots, \psi_n))$.

Lemma 5 (Loop Pattern Language Preservation). *Let (N, G) be a loop pattern with $N = (P, T, F)$; $T_{do}, T_{redo} \in G$; and $p_{do}, p_{redo} \in P$ as defined in Def. 11. Let ψ_{do} and ψ_{redo} be POWL models such that $\mathcal{L}(Project_{\circlearrowleft}(N, T_{do})) = \mathcal{L}(\psi_{do})$ and $\mathcal{L}(Project_{\circlearrowleft}(N, T_{redo})) = \mathcal{L}(\psi_{redo})$. Then, $\mathcal{L}(\circlearrowleft(\psi_{do}, \psi_{redo})) = \mathcal{L}(N)$.*

Proof. Let $N_{do} = Project_{\circlearrowleft}(N, T_{do})$ and $N_{redo} = Project_{\circlearrowleft}(N, T_{redo})$. Due to the soundness of N_{do} and N_{redo} and the absence of places connecting transitions from both the do- and redo-parts except p_{do} and p_{redo} (c.f., the proof of Lemma 2), we conclude that the language of N consists of sequences that can be segmented into subsequences of complete executions of N_{do} and N_{redo}, interleaved as follows:

$$\mathcal{L}(N) = \mathcal{L}(N_{do}) \cdot (\mathcal{L}(N_{redo}) \cdot \mathcal{L}(N_{do}))^*.$$

By combining this equality with Def. 5 and the assumption that $\mathcal{L}(\psi_{do}) = \mathcal{L}(N_{do}) \ \wedge \ \mathcal{L}(\psi_{redo}) = \mathcal{L}(N_{redo})$, we get: $\mathcal{L}(N) = \mathcal{L}(\circlearrowleft(\psi_{do}, \psi_{redo}))$.

Lemma 6 (Partial Order Pattern Language Preservation). *Let (N, G) be a partial order pattern with $N = (P, T, F)$, $G = \{T_1, \ldots, T_n\}$, and $\prec \in \mathcal{O}^n$ as defined in Def. 13. Let $\psi_1, \ldots, \psi_n$ be POWL models such that $\mathcal{L}(Project_{order}(N, T_i)) = \mathcal{L}(\psi_i)$ for each i $(1 \leq i \leq n)$. Then, $\mathcal{L}(\prec(\psi_1, \ldots, \psi_n)) = \mathcal{L}(N)$.*

Proof. Let $N_i = Project_{order}(N, T_i)$ for each i $(1 \leq i \leq n)$. By combining the semantics of partial orders (c.f. Def. 5) with the assumption that $\mathcal{L}(\psi_i) = \mathcal{L}(N_i)$ for each i $(1 \leq i \leq n)$, we can write:

$$\mathcal{L}(\prec(\psi_1, \ldots, \psi_n)) = \{\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n) \mid \forall_{1 \leq i \leq n} \sigma_i \in \mathcal{L}(N_i)\}.$$

(1) Proof for $\mathcal{L}(N) \subseteq \{\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n) \mid \forall_{1 \leq i \leq n} \sigma_i \in \mathcal{L}(N_i)\}$: Let $\sigma \in \mathcal{L}(N)$ be any firing sequence of N. We construct subsequences $\sigma_1, ..., \sigma_n$ by projecting σ onto T_i for each i $(1 \leq i \leq n)$. We need to show that σ can be expressed as a shuffle of $\sigma_1, ..., \sigma_n$, respecting the partial order $\prec$. This can be derived by proving the following three key points:

- All parts $T_i \in G$ are present within σ.
- Each σ_i is a firing sequence from N_i (i.e., $\sigma_i \in \mathcal{L}(N_i)$).
- The partial order between the subsequences is preserved in σ (i.e., $\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n)$).

(1.1) All parts $T_i \in G$ are present within σ: Assume, for the sake of contradiction, that there exists a part $T_i \in G$ not present within σ, i.e., $\sigma_i = \langle\rangle$. Since N is a WF-net, this implies that there must be a *decision point* $p \in P$ where some outgoing paths from p eventually lead to transitions in T_i and other outgoing paths from p do not. This violates the first condition of Def. 13, which states that if a place has outgoing flows leading to different transitions, then all such transitions must fall into the same part in G.

(1.2) Each σ_i is a firing sequence from N_i: The unique local start and end properties (c.f. Def. 13) ensure each subnet is executed independently from start to end within N. Therefore, σ_i must be a firing sequence from N_i.

(1.3) The partial order $\prec$ is preserved in σ: Assume $i \prec j$ for any $i, j \in \{1, \ldots, n\}$. Since $\prec = order(N, G)^+$, transitions from T_i are executed first, producing tokens that are needed to eventually enable T_j. Suppose, for the

sake of contradiction, that after the execution of transitions from T_j, transitions from T_i are re-enabled (i.e., tokens are produced in $\triangleright T_i$). Then we have two possible scenarios:

- (i) The re-enabling of T_i does not depend on the completion of T_j (i.e., it does not require the consumption of tokens from $T_j\triangleright$): This means that we can perform a full execution of the subnet of T_i and reach the subnet of T_j again before its completion, violating safeness.
- (ii) The re-enabling of T_i depends on the completion of T_j (i.e., it requires the consumption of tokens from $T_j\triangleright$): This implies the existence of a sequence of dependencies in the execution order $j \prec \ldots \prec i$. By transitivity, $j \prec i$ holds. This violates the asymmetry requirement of partial orders since $i \prec j$.

(2) Proof for $\{\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n) \mid \forall_{1 \leq i \leq n} \sigma_i \in \mathcal{L}(N_i)\} \subseteq \mathcal{L}(N)$: Consider any sequence $\sigma \in \sqcup\!\sqcup_{\prec}(\sigma_1, ..., \sigma_n)$ where $\sigma_i \in \mathcal{L}(N_i)$ for $1 \leq i \leq n$. We showed that all parts $T_i \in G$ must be visited in N (c.f. the proof of 1.1). Due to the unique local start and end properties in N (c.f. Def. 13), each subnet can be executed independently in N, without violating the execution order $\prec = order(N, G)^+$. Therefore, the interleaved sequence σ constitutes a valid firing sequence in N.

5.3 Overall Correctness Guarantee

In this section, we prove the correctness of Alg. 1. Specifically, we show that the algorithm, if successfully producing a POWL model, then the POWL model has the same language as the input WF-net.

Theorem 1 (Correctness). *Let $N = (P, T, F)$ be a safe and sound WF-net. If Alg. 1 successfully converts N into a POWL model ψ, then $\mathcal{L}(N) = \mathcal{L}(\psi)$.*

Proof. We prove the theorem by induction on the number of transitions in N.

- **Base case:** The theorem trivially holds for a WF-net that contains a single transition.
- **Inductive hypothesis:** For $n > 1$, assume the theorem holds for all safe and sound WF-net with fewer transitions than n (i.e., $|T| < n$).
- **Inductive step ($|T| = n$):** We consider the different cases in the algorithm:
  - **XOR pattern:** Suppose an XOR pattern (N, G) with $G = \{T_1, \ldots, T_n\}$ is identified. For each $T_i \in G$, N is projected onto T_i to obtain $N_i = Project_{\times}(N, T_i)$. By Lemma 1, each N_i is a safe and sound WF-net with fewer transitions than n. By the inductive hypothesis, the POWL model ψ_i obtained from N_i satisfies $\mathcal{L}(\psi_i) = \mathcal{L}(N_i)$. The algorithm returns $\psi = \times(\psi_1, \ldots, \psi_n)$. By Lemma 4, $\mathcal{L}(\psi) = \mathcal{L}(N)$.
  - **Loop or partial order pattern:** The proof is analogous to the XOR case, using the appropriate lemmas for structural guarantees (Lemma 2 or Lemma 3) and language equivalence (Lemma 5 or Lemma 6).
  - **No pattern is detected:** If no pattern is detected, then the algorithm returns *null*, which does not contradict the theorem.

By induction, the theorem holds for all safe and sound WF-nets successfully converted by Alg. 1.

5.4 Completeness Guarantee on Semi-Block-Structured WF-Nets

In this section, we show that Alg. 1 is complete when applied to semi-block-structured WF-nets (c.f. Def. 6).

Theorem 2 (Completeness). *Let N be a semi-block-structured WF-net. Alg. 1 successfully converts N into a POWL model that has the same language as N.*

Proof. We distinguish between three cases:

- **Case 1: N has a single transition:** This matches the base case of the algorithm.
- **Case 2: N corresponds to an XOR or loop pattern:** Projecting the net on each part yields another semi-block-structured WF-net.
- **Case 3: N does not correspond to an XOR or loop pattern:** The algorithm creates a partition $G = Partition_{order}(N)$ where transitions within the same block are grouped into the same part of the partition. The transitive closure of the execution order $order(N, G)^+$ must form a partial order because (i) substituting each part with a single transition turn the net into a marked graph and (ii) soundness implies acyclicity for marked graph WF-nets. Since the top-level blocks have unique entry and exit points, the unique local start and end requirements of Def. 13 are also met. Therefore, a partial order pattern is detected. Projecting N on each part yields either (i) a base case for single transitions or (ii) a semi-block-structured WF-net that corresponds to an XOR or loop pattern for the blocks.

After projection, sub-nets are recursively handled in the same manner. Thus, the algorithm successfully produces a POWL model. The language equivalence follows by Theorem 1.

6 Implementation and Scalability Assessment

To assess the scalability of Alg. 1, we implemented it, incorporating the reduction rules illustrated in Fig. 3 and Fig. 6. We then performed two experiments (code and data are available at https://github.com/humam-kourani/WF-net-to-POWL). In the first experiment, we utilized the process tree generator from [9,10] to generate 1000 process trees. These trees were then translated into WF-nets using PM4Py [3], resulting in a diverse set of WF-nets varying in size from 21 to 370 transitions and 15 to 305 places. In the second experiment, we used the ground truth WF-nets of the 20 processes from [11], which were originally derived from POWL models. For comparison, we also applied the WF-net to process tree converter from [27] in both experiments.

Results. In the first experiment, Alg. 1 successfully generated POWL models for all 1000 WF-nets, which was expected since process trees represent a subclass of POWL. The experiment demonstrated the high scalability of our approach, as illustrated in Fig. 11. Conversion times for our approach ranged from 0.002 to 2.48 seconds, whereas the tree-based converter from [27] required between 0.013

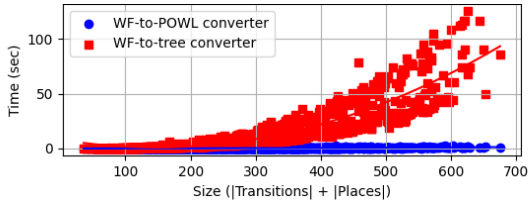


Fig. 11: Comparison of conversion times between Alg. 1 and the process tree converter from [27] on the 1000 WF-nets dataset.

and 126 seconds. These results highlight the efficiency of our algorithm's top-down decomposition strategy over the bottom-up approach used by the process tree converter. In the second experiment, while our algorithm was successful on all WF-nets, the tree-based converter was only able to convert 17 out of the 20 WF-nets into process trees.

In summary, our experiments demonstrate the advantage of our approach in both supporting a broader range of structures and its superior performance compared to the process tree-based converter. Note that the implemented algorithm is also available in ProMoAI [13] (https://promoai.streamlit.app/), powering the redesign feature for improving existing process models via large language models.

7 Conclusion

This paper introduced a novel algorithm for translating safe and sound Workflow Nets (WF-nets) into the Partially Ordered Workflow Language (POWL). The algorithm leverages the hierarchical structure of POWL by recursively identifying patterns within the WF-net that correspond to POWL's operators. We formally proved the correctness of our approach, showing that the resulting POWL model preserves the language of the original WF-net. Furthermore, we demonstrated the high scalability of the proposed algorithm and showed its completeness on semi-block-structured WF-nets, a subclass that contains equivalent workflow-nets for any POWL model.

This work paves the way for broader adoption of POWL in different process mining applications. The main avenue for future work is the development of optimized process mining techniques that leverage the structural properties of POWL, such as efficient algorithms for conformance checking. Furthermore, we aim to provide a platform that allows users to visualize process models in the POWL language and implements new methods for interactive process analysis and improvement.

References

1. Gérard Berthelot. Transformations and decompositions of nets. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Central Models and*

Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986, volume 254 of *Lecture Notes in Computer Science*, pages 359–376. Springer, 1986.
2. Gérard Berthelot and Lri-Iie. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, pages 19–40, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.
3. Alessandro Berti, Sebastiaan J. van Zelst, and Daniel Schuster. PM4Py: A process mining library for Python. *Softw. Impacts*, 17:100556, 2023.
4. Silvano Dal-Zilio. MCC: A tool for unfolding colored petri nets in PNML format. In Ryszard Janicki, Natalia Sidorova, and Thomas Chatain, editors, *Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Paris, France, June 24-25, 2020, Proceedings*, volume 12152 of *Lecture Notes in Computer Science*, pages 426–435. Springer, 2020.
5. Jorg Desel and Javier Esparza. *Free choice Petri nets*. Number 40. Cambridge university press, 1995.
6. Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.
7. Cédric Favre, Dirk Fahland, and Hagen Völzer. The relationship between workflow graphs and free-choice workflow nets. *Inf. Syst.*, 47:197–219, 2015.
8. Tracy Gardner. UML modelling of automated business processes with a mapping to BPEL4WS. *Orientation and Web Services*, 30, 2003.
9. Toon Jouck and Benoît Depaire. PTandLogGenerator: A generator for artificial event data. In Leonardo Azevedo and Cristina Cabanillas, editors, *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016*, volume 1789 of *CEUR Workshop Proceedings*, pages 23–27. CEUR-WS.org, 2016.
10. Toon Jouck and Benoît Depaire. Generating artificial data for empirical analysis of control-flow discovery algorithms - A process tree and log generator. *Bus. Inf. Syst. Eng.*, 61(6):695–712, 2019.
11. Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Evaluating large language models on business process modeling: Framework, benchmark, and self-improvement analysis. *CoRR*, abs/2412.00023, 2024.
12. Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Process modeling with large language models. In Han van der Aa, Dominik Bork, Rainer Schmidt, and Arnon Sturm, editors, *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*, volume 511 of *Lecture Notes in Business Information Processing*, pages 229–244. Springer, 2024.
13. 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, August 3-9, 2024*, pages 8708–8712. ijcai.org, 2024.
14. 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.
15. 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 2023, Proceedings*, volume 14159 of *LNCS*, pages 92–108. Springer, 2023.
16. Humam Kourani, Sebastiaan J. van Zelst, Daniel Schuster, and Wil M. P. van der Aalst. Discovering partially ordered workflow models. *Inf. Syst.*, 128:102493, 2025.
17. Peter Langner, Christoph Schneider, and Joachim Wehler. Petri net based certification of event-driven process chains. In Jörg Desel and Manuel Silva Suárez, editors, *Application and Theory of Petri Nets 1998, 19th International Conference, ICATPN '98, Lisbon, Portugal, June 22-26, 1998, Proceedings*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer, 1998.
18. Kristian Bisgaard Lassen and Wil M. P. van der Aalst. WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I*, volume 4275 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2006.
19. Sander J. J. Leemans. *Robust Process Mining with Guarantees - Process Discovery, Conformance Checking and Enhancement*, volume 440 of *LNBIP*. Springer, 2022.
20. Tadao Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, 1989.
21. Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*. Springer, 1998.
22. Khodakaram Salimifard and Mike Wright. Petri net-based modelling of workflow systems: An overview. *Eur. J. Oper. Res.*, 134(3):664–676, 2001.
23. Wil M. P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management, Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer, 2000.
24. Wil M. P. van der Aalst. Reduction using induced subnets to systematically prove properties for free-choice nets. In Didier Buchs and Josep Carmona, editors, *Application and Theory of Petri Nets and Concurrency - 42nd International Conference, PETRI NETS 2021, Virtual Event, June 23-25, 2021, Proceedings*, volume 12734 of *Lecture Notes in Computer Science*, pages 208–229. Springer, 2021.
25. Wil M. P. van der Aalst and Kristian Bisgaard Lassen. Translating unstructured workflow processes to readable BPEL: theory and implementation. *Inf. Softw. Technol.*, 50(3):131–159, 2008.
26. 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.
27. Sebastiaan J. van Zelst and Sander J. J. Leemans. Translating workflow nets to process trees: An algorithmic approach. *Algorithms*, 13(11):279, 2020.
28. Mark von Rosing, Stephen White, Fred Cummins, and Henk de Man. Business process model and notation - BPMN. In Mark von Rosing, Henrik von Scheel, and August-Wilhelm Scheer, editors, *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I*, pages 429–453. Morgan Kaufmann/Elsevier, 2015.