Article

Translating Workflow Nets to Process Trees: An Algorithmic Approach

Sebastiaan J. van Zelst [1,2], Sander J.J. Leemans [3]

[1] Fraunhofer Institute for Applied Information Technology (FIT), Germany; sebastiaan.van.zelst@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany; s.j.v.zelst@pads.rwth-aachen.de
[3] Queensland University of Technology, Brisbane, Australia; s.leemans@qut.edu.au
* Correspondence: sebastiaan.van.zelst@fit.fraunhofer.de

Version October 29, 2020 submitted to Algorithms

Abstract: Since their introduction, process trees have been frequently used as a process modeling formalism in many process mining algorithms. A process tree is a (mathematical) tree-based model of a process, in which internal vertices represent behavioral control-flow relations and leaves represent process activities. Translation of a process tree into a sound Workflow net is trivial; however, the reverse is not the case. Simultaneously, an algorithm that translates a WF-net into a process tree is of great interest, e.g., the explicit knowledge of the control-flow hierarchy in a WF-net allows one to reason on its behavior more easily. Hence, in this paper, we present such an algorithm, i.e., it detects whether a WF-net corresponds to a process tree, and, if so, constructs it. We prove that, when the algorithm finds a process tree, the language of the process tree is equal to the language of the original WF-net. The experiments conducted show that the algorithm's corresponding implementation has a quadratic time complexity in the size of the WF-net. Furthermore, the experiments show strong evidence of process tree re-discoverability.

Keywords: process trees; Petri nets; workflow nets; process mining.

---

1. Introduction

Process mining [1] is concerned with distilling knowledge of the execution of processes by analyzing the event data generated during the execution of these processes, i.e., stored in modern-day information systems. In the field, different (semi-)automated techniques have been developed that allow one to distill processes knowledge from event data, i.e., ranging from *automated process discovery algorithms* to *conformance checking algorithms*. In automated process discovery, the main aim is to translate observed process behavior, i.e., as stored in the information system, into a process model that accurately describes the behavior of the process. In this context, the discovered process model should strike an adequate balance between accounting for unobserved, yet likely, process behavior (i.e., avoiding overfitting) and being precise (i.e., avoiding underfitting) at the same time. Conformance checking techniques allow us to compute to what degree the observed behavior is in line with a given reference process model (either designed by hand or discovered using an automated process discovery technique). Since processes are the cornerstone of process mining, so are the models that allow us to represent them (and reason about their behavior and quality). As such, various process modeling formalisms exist, e.g., BPMN [2], EPCs [3], etc., some of which are heavily used in practice.

Recently, process trees were introduced [4]. A process tree is a hierarchical representation of a process corresponding to the mathematical notion of a rooted tree, i.e., a connected undirected acyclic graph with a designated root vertex. The internal vertices of a process tree represent how their children relate to each other regarding their control-flow behavior (i.e., their sequential scheduling). The leaves of the tree represent the activities of the process. Consider Fig. 3 (page 6), in which we depict an example process tree. Its root vertex has label $\rightarrow$, specifying that first its leftmost child, i.e., activity a, needs to be executed, secondly its middle child, and, finally, its rightmost child. Its middle child has a $\circlearrowleft$ label, specifying cyclic behavior, i.e., its leftmost child is *always executed*, whereas the execution of its rightmost child stipulates that we need to *repeat* its leftmost child. The $\times$-label in a process tree represents an *exclusive choice*, e.g., vertex $v_{1.3}$ specifies that we either executed activity g or h, yet not both. Finally, the $\wedge$-label refers to concurrency, i.e, the children of a vertex with such a label are allowed to be executed simultaneously, i.e., at the same time. Furthermore, consider the two models depicted in Fig. 1. (page 3). The models represent the same behavior, based on a real-life event log. Clearly, the hierarchy of the process tree allows one to more easily understand the main control-flow of the process.

The previous examples show the relative simplicity at which one can reason on the behavior of a process tree. Furthermore, it is straightforward to translate process trees into other process modeling formalisms, e.g., *Workflow nets (WF-nets)*. By definition, a process tree corresponds to a *sound WF-net*, i.e., a WF-net with desirable behavioral properties, e.g., the absence of deadlocks. The reverse, i.e., translating a given WF-net into a process tree (if possible), is less trivial. At the same time, obtaining such a translation is of great interest, e.g., it allows us to discover control-flow-aware hierarchical structures within a WF-net. Such structures can, for example, be used to hide certain parts of the model, i.e., leading to a more understandable view of the process model. Furthermore, any algorithm optimized for process trees, e.g., by exploiting the hierarchical structure, can also be applied to WF-nets of such a type. For example, in [8], it is shown that the computation time of *alignments* [9], i.e., explanations of observed behavior in terms of a reference model, can be significantly reduced by applying Petri net decomposition on the basis of model hierarchies. Hence, computing a process tree representation of the WF-net can be exploited to reduce the computational complexity of the calculations mentioned.

In this paper, we present an algorithm that determines whether a given WF-net corresponds to a process tree, and, if so, constructs it. We prove that, if the algorithm finds a process tree, the original WF-net is sound, and the obtained process tree's language is equal to the language of the original WF-net. A corresponding implementation, extending the process mining framework PM4Py [10], is publicly available. Using the implementation, we conducted several experiments that show a quadratic time complexity in terms of the WF-net size. Furthermore, our experiments indicate that the algorithm can re-discover process trees, i.e., the process models used to generate the input for the experiments are re-discovered by the algorithm.

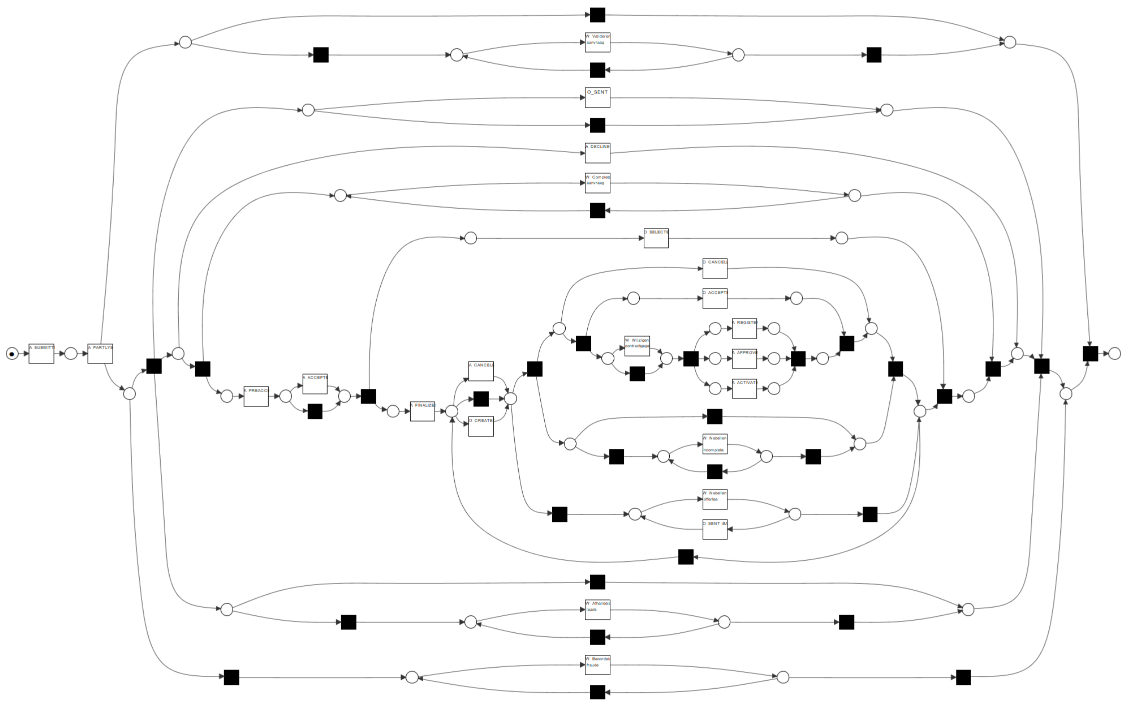
The remainder of this paper is structured as follows. In Section 2, we present preliminary concepts and notation. In Section 3, we present the proposed algorithm, including the proofs w.r.t soundness preservation and language preservation. In Section 4, we evaluate our approach. In Section 5, we discuss related work. In Section 6, we discuss various aspects of our approach, e.g., extensibility, in more detail. Section 7, concludes the paper.

2. Preliminaries

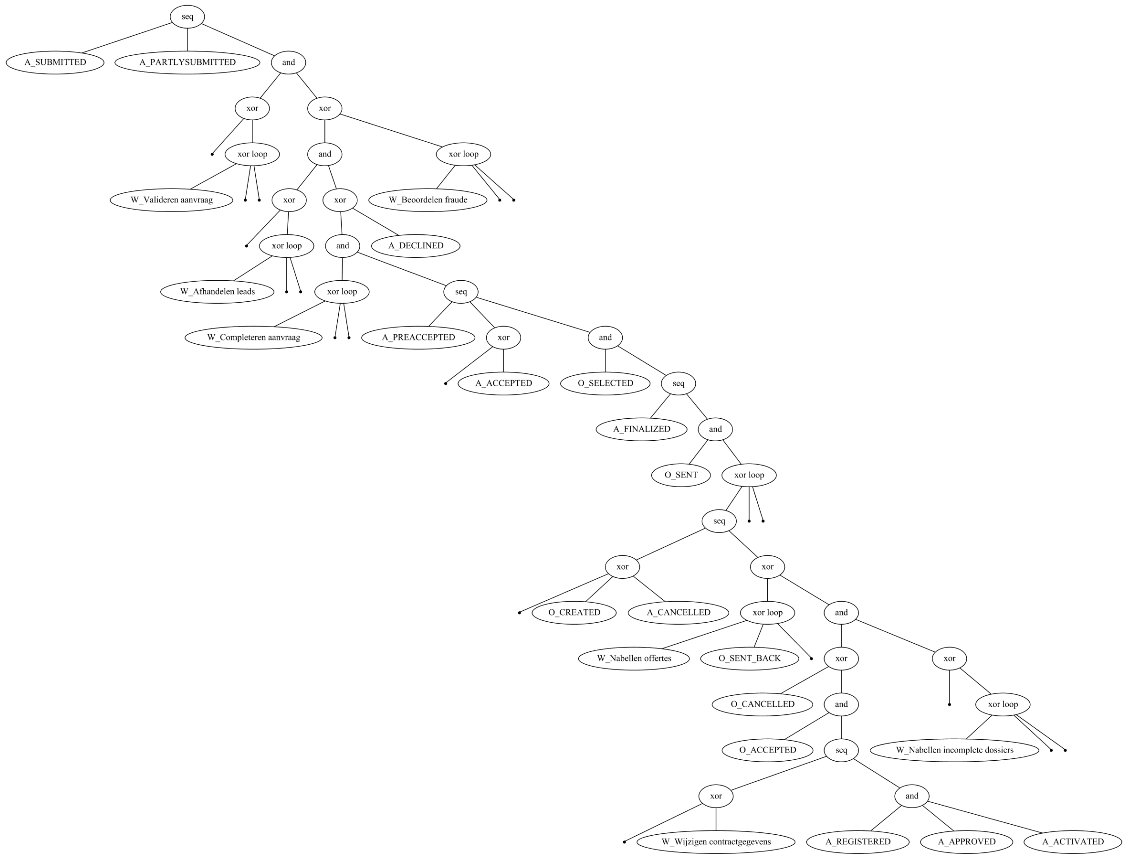
In this section, we present basic preliminary notions that ease the readability of this paper. In Section 2.1, we present the basic notation used in this paper. In Section 2.2, we introduce Workflow nets. Finally, in Section 2.3, we present the notion of process trees and their relation to Workflow nets.

2.1. Basic Notation

Given set X, $\mathcal{P}(X)=\{X' \subseteq X\}$ denotes its powerset. Given a function $f\colon X \rightarrow Y$ and $X' \subseteq X$, we extend function application to sets, i.e., $f(X')=\{y|\exists x{\in}X'(f(x)=y)\}$. Furthermore, $f|_{X'}\colon X' \rightarrow Y$ restricts f to X'. A multiset over set X, i.e., $m\colon X \rightarrow \mathbb{N}\cup\{0\}$, contains multiple instances of an element. We write a multiset as $m=[x_1^i, x_2^j, ..., x_n^k]$, where $m(x_1)=i, m(x_2)=j, ..., m(x_n)=k$, for $i, j, ..., k>1$ (in case, $m(x_i)=1$, we omit its superscript; in case $m(x_i)=0$, we omit x_i). The set of all multisets over X is written



(a) The process, represented as a WF-net.



(b) The process, represented as a Process Tree.

Figure 1. The same process model, obtained by applying the Inductive Miner [5] implementation of ProM [6] on a real event data set [7], in different process modeling formalisms. Because of its hierarchical nature, the process tree formalism easily allows us to spot the main control-flow behavior.

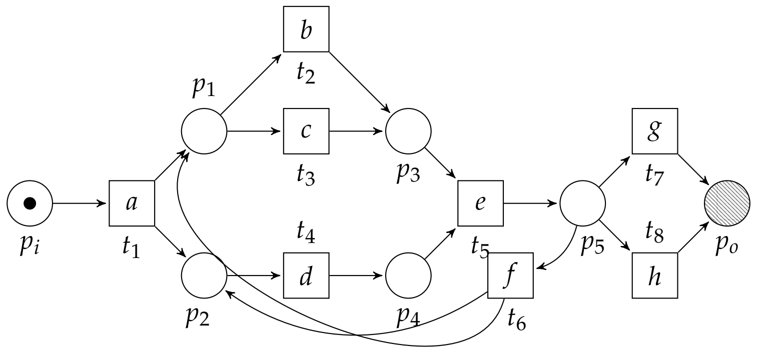


Figure 2. WF-net W_1 [1] with initial marking $[p_i]$ and final marking $[p_o]$.

as $\mathcal{M}(X)$. Given $m{\in}\mathcal{M}(X)$, we write $x{\in_+}m$ if $m(x){>}0$, and, $x{\notin_+}m$ if $m(x){=}0$, and, $\overline{m}{=}\{x|x{\in_+}m\}$. For example, the multiset $[x^2, y]$ consists of two instances of x, one instance of y, and zero instances of z. The sum of two multisets m_1, m_2 is written as $m_1{\uplus}m_2$, e.g., $[x^2, y]{\uplus}[x^3, y, z]{=}[x^5, y^2, z]$, their difference is written as $m_1{-}m_2$, e.g., $[x^2, y]{-}[x, y, z]{=}[x]$. A set is considered a multiset in which each element appears only once. Hence, we also apply the operations defined for multisets on sets, and, on combinations of sets and multisets, e.g., $\{x, y, z\}{\uplus}[x^2]{=}[x^3, y, z]$.

A sequence is an ordered collection of elements, e.g., a sequence σ of length n over base set X is a function $\sigma\colon \{1, ..., n\}{\to}X$. We write $|\sigma|$ to denote the length of σ, e.g., $|\sigma|{=}n$. We write $\sigma{=}\langle\sigma(1), \sigma(2), ..., \sigma(|\sigma|)\rangle$, where $\sigma(i)$ denotes the element at position i, $(1{\leq}i{\leq}|\sigma|)$. ϵ denotes the empty sequence, i.e., $|\epsilon|{=}0$. We extend the notion of element inclusion to sequences, e.g., $x{\in}\langle x, y, z\rangle$. X^* denotes the set of all sequences over members of set X. Concatenation of sequences $\sigma, \sigma'{\in}X^*$ is written as $\sigma{\cdot}\sigma'$. We let $\sigma{\leftrightarrows}\sigma'$ denote the set of all possible order-preserving merges, i.e., the *shuffle operator*, of σ and σ', e.g., given $\sigma_1{=}\langle b, p\rangle$, $\sigma_2{=}\langle m\rangle$, then $\sigma_1{\leftrightarrows}\sigma_2{=}\ \{\langle b, p, m\rangle, \langle b, m, p\rangle, \langle m, b, p\rangle\}$. It is easy to see that $\sigma{\leftrightarrows}\sigma'{=}\sigma'{\leftrightarrows}\sigma$ (the operator is commutative). We extend the shuffle operator to sets (and overload notation), i.e., given $S, S'{\in}X^*$, $S{\leftrightarrows}S'{=}\{\sigma{\in}\sigma_1{\leftrightarrows}\sigma_2 \mid \sigma_1{\in}S_1{\wedge}\sigma_2{\in}S_2\}$. Note that, $(\sigma{\leftrightarrows}\sigma'){\leftrightarrows}\{\sigma''\} = \{\sigma\}{\leftrightarrows}(\sigma'{\leftrightarrows}\sigma'')$ (associative), hence, we write the application of the shuffle operation on n sequences as $\sigma_1{\leftrightarrows}\sigma_2{\leftrightarrows}\cdots{\leftrightarrows}\sigma_n$. Similarly, we write $S_1{\leftrightarrows}S_2{\leftrightarrows}\cdots{\leftrightarrows}S_n$ for sets of sequences $S_1, S_2, ...S_n{\in}X^*$. Given a function $f\colon X{\to}Y$ and a sequence $\sigma{\in}X^*$, we overload notation for function application, i.e., $f(\sigma){=}\langle f(\sigma(1), f(\sigma(2)), ..., f(\sigma|\sigma|)\rangle$. We extend the notion of sequence application to sets of sequences, i.e., given $f\colon X \to Y$ and $X'{\subseteq}X^*$, $f(X'){=}\{\sigma{\in}Y^* \mid \exists\sigma'{\in}X'\ (f(\sigma'){=}\sigma)\}$. Furthermore, given $X'{\subseteq}X$ and a sequence $\sigma{\in}X^*$, we define $\sigma_{\downarrow_{X'}}$, where (recursively) $\epsilon_{\downarrow_{X'}}{=}\epsilon$, $(\langle x\rangle{\cdot}\sigma)_{\downarrow_{X'}}{=}x \cdot \sigma_{\downarrow_{X'}}$ if $x{\in}X'$ and $(\langle x\rangle{\cdot}\sigma)_{\downarrow_{X'}}{=}\sigma_{\downarrow_{X'}}$ if $x{\notin}X'$.

2.2. Workflow Nets

Workflow nets (WF-nets) [11] extend the more general notion of Petri nets [12]. A Petri net is a *directed bipartite graph* containing two types of vertices, i.e., places and transitions. We visualize places as circles, whereas we visualize transitions as boxes. Places only connect to transitions and vise-versa. Consider Fig. 2, depicting an example Petri net (which is also a WF-net). We let $N{=}(P, T, F, \ell)$ denote a labeled Petri net, where, P denotes a set of places, T denotes a set of transitions and $F{\subseteq}(P{\times}T) \cup (T{\times}P)$ represents the arcs. Furthermore, given a set of labels Σ and the symbol $\tau{\notin}\Sigma$, $\ell\colon T{\to}\Sigma{\cup}\{\tau\}$ is the net's labelling function, e.g., in Fig. 2, $\ell(t_1){=}a$, $\ell(t_2){=}\ b$, etc. Given an element $x{\in}P{\cup}T$, $\bullet x = \{y \mid (y, x){\in}F\}$ denotes the *pre-set* of x, whereas $x\bullet{=}\ \{y|(x, y){\in}F\}$ denotes its *post-set*, e.g., in Fig. 2, $\bullet t_1{=}\ \{p_i\}$ and $p_1\bullet{=}\ \{t_2, t_3\}$. We lift the $\bullet$-notation to the level of sets, i.e., given $X{\subseteq}P{\cup}T$, $\bullet X{=}\ \{y \mid \exists x{\in}X\ (y{\in}\bullet x)\}$ and $X\bullet{=}\ \{y \mid \exists x{\in}X\ (y{\in}x\bullet)\}$. Let $N{=}(P, T, F, \ell)$ be a Petri net and let $P'{\subseteq}P$, $T'{\subseteq}T$ and $F' = F{\cap}((P'{\times}T') \cup (T'{\times}P'))$. The Petri net $N'{=}\ (P', T', F', \ell|_{T'})$ is a *subnet* of N, written $N'{\sqsubseteq}N$. In the context of this paper, we refer to a subnet $N'{\sqsubseteq}N$ as a *fragment* if it is *weakly connected*.

Furthermore, a fragment $N'{\sqsubseteq}N$ is *place-bordered* iff the only vertices of N' (i.e., members of $P'{\cup}T'$) that are connected to vertices that do not belong to N' (i.e., members of $P{\cup}T\setminus(P'{\cup}T')$) are places, i.e., $\{x{\in}P'{\cup}T' \mid (x,y){\in}F\setminus F' {\vee} (y,x){\in}F\setminus F'\}{\subseteq}P'$. Furthermore, we refer to $P'_i{=}\{x{\in}P' | (y,x){\in}F\setminus F'\}$ and $P'_o{=}\{x{\in}P' | (x,y){\in}F\setminus F'\}$ to the *input* and *output places* of the place-bordered fragment. For example, in Fig. 2, the subnet formed by places p_1, p_2, p_3, p_4, p_5, transitions t_2, t_3, t_4, t_5 and the arcs $(p_1,t_2),(p_1,t_3),\dots,(t_5,p_5)$ is a place-bordered fragment. Observe that, if we remove place p_5 (and the corresponding arc (t_5,p_5)), the subnet is still a fragment, yet, no longer place-bordered. If we also remove t_5 and the arcs (p_3,t_5) and (p_4,t_5), the subnet is not a fragment as it is no longer weakly connected. We let $\mathcal{N}$ denote the universe of Petri nets.

The *state* of a Petri net is expressed by means of a *marking*, i.e., a multiset of places. A marking is visualized by drawing the corresponding number of dots in the place(s) of the marking, e.g., the marking in Fig. 2 is $[p_i]$ (one black dot is drawn inside place p_i). Given a Petri net $N = (P,T,F,\ell)$ and marking $M{\in}\mathcal{M}(P)$, (N,M) denotes a *marked net*. Given a marked net (N,M), a transition $t{\in}T$ is *enabled*, written $(N,M)[t\rangle$, if $\forall p{\in}{\bullet}t\,(M(p){>}0)$. If a transition is *not enabled* in marking M, we write $(N,M)[t\!\!\!/\rangle$. An enabled transition can *fire*, leading to a new marking $M'{=}(M{-}{\bullet}t)\uplus t{\bullet}$, written $(N,M)\xrightarrow{t}(N,M')$. A sequence of transition firings $\sigma{=}\langle t_1,t_2,...,t_n\rangle$ is a *firing sequence* of (N,M), yielding marking M', written $(N,M)\xrightarrow{\sigma}\!\!\!\!\rightarrow(N,M')$, iff $\exists M_1,M_2,...,M_{n-1}{\in}\mathcal{M}(P)$ s.t. $(N,M)\xrightarrow{t_1}(N,M_1)\xrightarrow{t_2}(N,M_2)\cdots(N,M_{n-1})\xrightarrow{t_n}(N,M')$. We write $(N,M)\xrightarrow{\sigma}\!\!\!\!\rightarrow\circ$, in case σ is a firing sequence in (N,M), yet, we are not interested in the marking it leads to. In some cases, we simply write $(N,M){\rightsquigarrow}(N,M')$, if $\exists\sigma{\in}T^*\left((N,M)\xrightarrow{\sigma}\!\!\!\!\rightarrow(N,M')\right)$. $\mathcal{L}_{\mathcal{N}}(N,M,M') = \left\{\sigma{\in}T^* \mid (N,M)\xrightarrow{\sigma}\!\!\!\!\rightarrow(N,M')\right\}$ denotes all firing sequences starting from marking M, leading to marking M'. The *labeled-language* of N, conditional to markings M and M', is defined as $\mathcal{L}^v_{\mathcal{N}}(N,M,M'){=}\ell(\mathcal{L}_{\mathcal{N}}(N,M,M'))_{\downarrow\Sigma}$. $\mathcal{R}(N,M) = \left\{M'{\in}\mathcal{M}(P) \mid \exists\sigma{\in}T^*\left((N,M) \xrightarrow{\sigma}\!\!\!\!\rightarrow (N,M')\right)\right\}$ denotes the reachable markings.

Given a Petri net $N{=}(P,T,F,\ell)$, and a designated initial and final marking $M_i,M_f{\in}\mathcal{M}(P)$, the triple $SN{=}(N,M_i,M_f)$ denotes a *system net*. As system net $SN{=}(N,M_i,M_f)$ is formed by N, we write SN as a replacement for N, e.g., (SN,M) denotes a marked system net. Clearly, $\mathcal{R}(SN,M)$, $\mathcal{L}_{\mathcal{N}}(SN,M,M')$, etc., are readily defined for arbitrary markings $M,M'{\in}\mathcal{M}(P)$. The *language* of SN is referred to as $\mathcal{L}_{\mathcal{N}}(SN,M_i,M_f)$, for which we simply write $\mathcal{L}_{\mathcal{N}}(SN)$ (respectively $\mathcal{L}^v_{\mathcal{N}}(SN)$,$\mathcal{R}(SN)$, etc.), i.e., we drop M_i and M_f from the notation as they are clear from context. $\mathcal{SN}$ denotes the universe of system nets.

A WF-net is a special type of Petri net, i.e., it has one unique start and one unique end place. Furthermore, every place/transition in the net is on a path from the start to the end place. We formally define a WF-net as follows

Definition 1 (Labeled Workflow net (WF-net))**.** *Let Σ denote the universe of (activity) labels, let $\tau{\notin}\Sigma$ and let $\ell\colon T{\to}\Sigma{\cup}\{\tau\}$. Let $N = (P,T,F,\ell){\in}\mathcal{N}$ and let $p_i{\neq}p_o{\in}P$. Tuple $W{=}(P,T,F,p_i,p_o,\ell)$ is a Workflow net (WF-net), iff:*

1. ${\bullet}p_i{=}\emptyset \wedge \nexists p{\in}P\setminus\{p_i\}\,({\bullet}p{=}\emptyset)$*; p_i is the unique* source place.
2. $p_o{\bullet}{=}\emptyset \wedge \nexists p{\in}P\setminus\{p_o\}\,(p{\bullet}{=}\emptyset)$*; p_o is the unique* sink place.
3. *Each element $x{\in}P{\cup}T$ is on a path from p_i to p_o.*

We let $\mathcal{W}$ denote the universe of WF-nets.

Observe that, a WF-net is a system net with $M_i{=}[p_i]$ and $M_f{=}[p_o]$ Hence, since a WF-net is formed by an underlying Petri net, and, has a well-defined initial and final marking, i.e., $[p_i]$ and $[p_o]$, we write $\mathcal{L}_{\mathcal{N}}(W)$ (respectively $\mathcal{L}^v_{\mathcal{N}}(W)$,$\mathcal{R}(W)$, etc.) as a shorthand notation for $\mathcal{L}_{\mathcal{N}}(W,[p_i],[p_o])$.

Of particular interest are *sound WF-nets*, i.e., WF-nets that are guaranteed to be free of *deadlocks*, *livelocks* and *dead transitions*. We formalize the notion of *soundness* as follows.

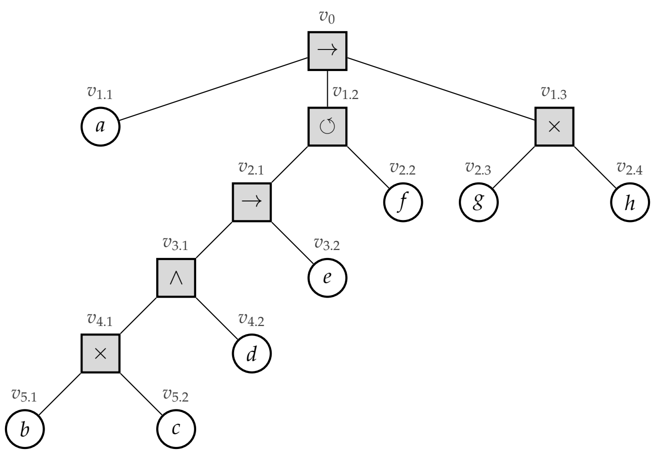


Figure 3. Process tree [1], describing the same language as the WF-net in Fig. 2.

Definition 2 (Soundness)**.** *Let* $W{=}(P,T,F,p_i,p_o,\ell){\in}\mathcal{W}$. *W is* sound *iff:*

1. $(W,[p_i])$ *is* safe, *i.e.,* $\forall M{\in}\mathcal{R}(W,[p_i])\,(\forall p{\in}P\,(M(p){\leq}1))$,
2. $[p_o]$ *can always be reached, i.e.,* $\forall M{\in}\mathcal{R}(W,[p_i])\,((W,M){\rightsquigarrow}(W,[p_o]))$.
3. *Each* $t{\in}T$ *is enabled, at some point, i.e.,* $\forall t{\in}T\,(\exists M{\in}\mathcal{R}(W,[p_i])\,(M[t\rangle))$.

Observe that, the Petri net depicted in Fig. 2 is a sound WF-net, i.e., it adheres to all three requirements of Definition 2.

2.3. Process Trees

Process trees allow us to model processes, that comprise a control-flow hierarchy. A process tree is a mathematical tree, where the internal vertices are *operators* and leaves are (non-observable) *activities*. Operators specify how their children, i.e., sub-trees, need to be combined from a control-flow perspective. Several operators can be defined, however, in this work, we focus on four basic operators, i.e., the $\rightarrow$, $\times$, $\wedge$ and $\circlearrowleft$-operator. The *sequence* operator ($\rightarrow$) specifies sequential behavior. First its left-most child is executed, then its second left-most child, ..., and finally its right-most child. For example, the root operator in Fig. 3 specifies that first activity *a* is executed, then its second sub-tree ($\circlearrowleft$) and then its third sub-tree ($\times$). The *exclusive choice* operator ($\times$), specifies an exclusive choice, i.e., one (and exactly one) of its sub-trees is executed. Concurrent/parallel behavior is represented by the *concurrency operator* ($\wedge$), i.e., all children are executed simultaneously/in any order. Finally, we represent repeated behavior by the *loop operator* $\circlearrowleft$. Whereas the $\rightarrow$, $\times$ and $\wedge$-operator have an arbitrary number of children, the $\circlearrowleft$-operator has two children.[1] Its left child (the "do-child") is *always* executed. Secondly, executing its right child (the "re-do-child") is optional. After executing the re-do-child, we again execute the do-child. We are allowed to repeat this, yet, we always finish with the do-child.

For example, consider Fig. 3, in which we depict an example process tree (describing the same language as the WF-net in Fig. 2). The root of the tree, i.e., v_0, is a sequence operator, specifying that first its left-most child ($v_{1.1}$) needs to be executed. Its middle child, i.e., $v_{1.2}$, represents a loop operator. The left sub-tree of the loop operator (i.e., having vertex $v_{2.1}$ as its root) is always executed. Vertex $v_{2.2}$ represents the "redo" part of the loop operator. The last part of the tree is represented by $v_{1.3}$, i.e., specifying a choice construct between executing activity *g* or *h*.

Definition 3 (Process Tree)**.** *Let* Σ *denote the universe of (activity) labels and let* $\tau{\notin}\Sigma$. *Let* $\bigoplus$ *denote the universe of* process tree operators. *A process tree Q, is defined (recursively) as any of:*

[1] Note that various definitions of the loop operator exist, i.e., with three/an arbitrary number of children (e.g. [1, Definition 3.13]). However, all of these definitions can be rewritten into the binary loop operator, as described here.

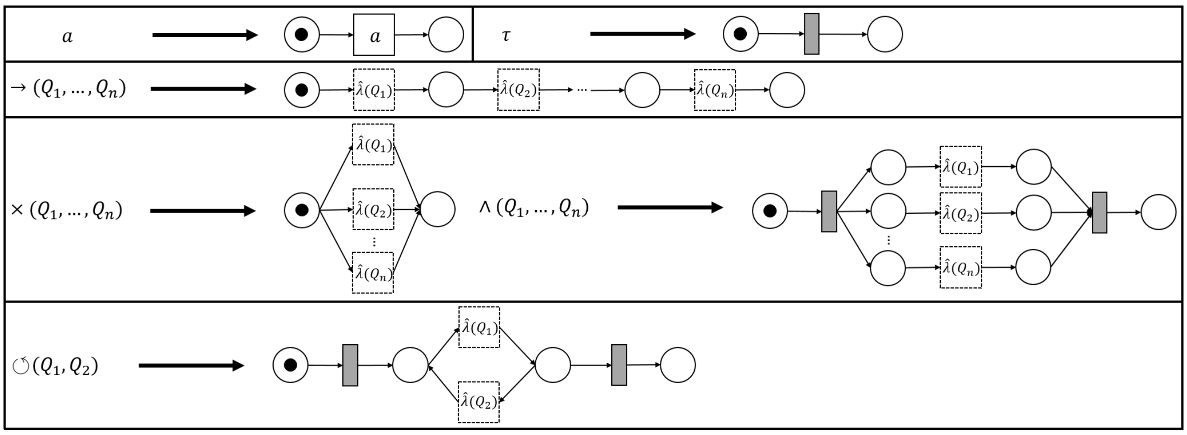


Figure 4. Instantiations of λ (Definition 5). The λ-functions for operators are defined recursively, using the $\hat{\lambda}$-values of their children, i.e., a place "entering"/"exiting" a $\hat{\lambda}(Q_i)$ fragment, connects to $p_i\bullet/\bullet p_o$ (respectively) of $\lambda(Q_i)$.

1. *$x\in\Sigma\cup\{\tau\}$; an (non-observable) activity,*
2. *$\oplus(Q_1,...,Q_n)$, for $\oplus\in\bigoplus$, $n\geq1$, where $Q_1,...,Q_n$ are process trees;*

We let $\mathcal{Q}$ denote the universe of process trees.

Given a process tree $Q\in\mathcal{Q}$, its language is of the form $\mathcal{L}_{\mathcal{Q}}(Q)\subseteq\Sigma^*$, which is recursively defined in terms of of the languages of the children of a process tree. For example, the language of the $\rightarrow$-operator, is formed by concatenating any element of the language of its first child, with any element of its second child, etc. We formally define the language of a process tree as follows.

Definition 4 (Process Tree Language). *Let $Q\in\mathcal{Q}$ be a process tree. The language of Q, i.e., $\mathcal{L}_{\mathcal{Q}}(Q)\subseteq\Sigma^*$, is defined recursively as:*

$$\mathcal{L}_{\mathcal{Q}}(Q)=\{\epsilon\},\text{ if } Q=\tau$$
$$\mathcal{L}_{\mathcal{Q}}(Q)=\{\langle a\rangle\}\text{ if } Q=a\in\Sigma$$
$$\mathcal{L}_{\mathcal{Q}}(Q)=\{\sigma=\sigma_1\cdot\sigma_2\cdots\sigma_n \mid \sigma_1\in\mathcal{L}_{\mathcal{Q}}(Q_1),\sigma_2\in\mathcal{L}_{\mathcal{Q}}(Q_2),...,\sigma_n\in\mathcal{L}_{\mathcal{Q}}(Q_n)\}\text{ if } Q=\rightarrow(Q_1,Q_2,...,Q_n)$$
$$\mathcal{L}_{\mathcal{Q}}(Q)=\bigcup_{i=1}^{n}\mathcal{L}_{\mathcal{Q}}(Q_n)\text{ if } Q=\times(Q_1,Q_2,...,Q_n)$$
$$\mathcal{L}_{\mathcal{Q}}(Q)=\mathcal{L}_{\mathcal{Q}}(Q_1)\leftleftharpoons\mathcal{L}_{\mathcal{Q}}(Q_2)\cdots\leftleftharpoons\mathcal{L}_{\mathcal{Q}}(Q_n)\text{ if } Q=\wedge(Q_1,Q_2,...,Q_n)$$
$$\mathcal{L}_{\mathcal{Q}}(Q)=\{\sigma_1\cdot\sigma'_1\cdot\sigma_2\cdot\sigma'_2\cdots\sigma_n|n\geq1\wedge\forall1\leq i\leq n\left(\sigma_i\in\mathcal{L}_{\mathcal{Q}}(Q_1)\right)\wedge\forall1\leq i<n\left(\sigma'_i\in\mathcal{L}_{\mathcal{Q}}(Q_2)\right)\}\text{ if } Q=\circlearrowleft(Q_1,Q_2)$$

The process tree operators that we consider in this paper ($\rightarrow$, $\times$, $\wedge$ and $\circlearrowleft$) are easily translated to sound WF-nets, cf. Fig. 4. Hence, we define a generic process tree to WF-net translation function, s.t., the language of the two is the same.

Definition 5 (Process Tree Transformation Function). *Let $Q\in\mathcal{Q}$ be a process tree. A process tree transformation function λ, is a function $\lambda\colon\mathcal{Q}\rightarrow\mathcal{W}$, s.t., $\mathcal{L}^v_{\mathcal{N}}(\lambda(Q))=\mathcal{L}_{\mathcal{Q}}(Q)$. We let $\hat{\lambda}\colon\mathcal{Q}\rightarrow\mathcal{N}$, where, given $\lambda(Q)=W=(P,T,F,p_i,p_o,\ell)$, $\hat{\lambda}(Q)=(P',T,F',\ell)$, with, $P'=p\setminus\{p_i,p_o\}$ and $F'=F\setminus\left(\{(p_i,t)\in F\}\cup\{(t,p_o)\in F\}\right)$.*

Given an arbitrary process tree $Q\in\mathcal{Q}$, there are several ways to translate it to a sound WF-net W, s.t., $\mathcal{L}^v_{\mathcal{N}}(W)=\mathcal{L}_{\mathcal{Q}}(Q)$, i.e., instantiating λ and $\hat{\lambda}$. As an example, consider the translation functions, depicted in Fig. 4. Note that, each transformation function in Fig. 4, is sound by construction. Interestingly, recursively inserting the $\hat{\lambda}$-generated fragments of the sub-trees of a given process tree, corresponds to the sequential application of WF-net composition, as described in [13, Section 7]. Hence, we deduce that their recursive composition is also a sound [13, Theorem 3.3]. Note that, in the remainder of this paper, we explicitly assume the use of λ, as presented in Fig. 4.[2]

3. Translating Workflow Nets to Process Trees

In this section, we describe our approach. In Section 3.1, we sketch the main idea of the approach, using a small example. In Section 3.2, we present PTree-nets, i.e., Petri nets with $rng(\ell){=}\mathcal{Q}$, which we exploit in our approach. In Section 3.3, we present Petri net fragments, used to identify process tree operators within the net, together with a generic reduction function. Finally, in Section 3.4, we provide an algorithmic description that allows us to find process trees, including correctness proofs.

3.1. Overview

The core idea of the approach concerns searching for fragments in the given WF-net that represent behavior that is expressible as a process tree. The patterns we look for bear significant similarity with the translation patterns defined in Fig. 4, i.e., they are a *strongly generalized reverse* of those patterns. When we find a pattern, we replace it with a smaller net fragment representing the process tree that was identified. We continue to search for patterns in the reduced net until we are not able to find any more patterns. As we prove in Section 3.4, in case the final WF-net contains just one transition, its label carries a process tree with the same labeled-language as the original WF-net.

Consider Fig. 5, in which we sketch the basic idea of the algorithm, applied on the example WF-net W_1 (Fig. 2). First, the algorithm detects two *choice constructs*, i.e., one between the transitions labeled b and c, and one between the transitions labeled g and h. The algorithm replaces the fragments by means of two new transitions, carrying labels $\times(b,c)$ and $\times(g,h)$ respectively (Fig. 5a). Subsequently, a concurrent construct is detected, i.e., between the transitions labeled $\times(b,c)$ and d. Again, the pattern is replaced (Fig. 5b). A sequential pattern is detected and replaced (Fig. 5c), after which a loop construct is detected (Fig. 5d). The resulting process tree, i.e., carried by the remaining transition in Fig. 5e, $\rightarrow(a,\circlearrowleft(\rightarrow(\wedge(\times(b,c),d),e),f),\times(g,h))$, is equal to Fig. 3.

3.2. PTree-Nets and their Unfolding

As indicated, we aim to find Petri net fragments in the WF-net representing behavior equivalent to a process tree. As illustrated in Section 3.1, the patterns found in the WF-net are replaced by transitions with a label carrying a corresponding process tree. In the upcoming section, we present four different fragment characterizations, corresponding to the basic process tree operators considered. However, in this section, we first briefly present PTree-nets, i.e., a trivial extension of Petri nets, in which labels are process trees.

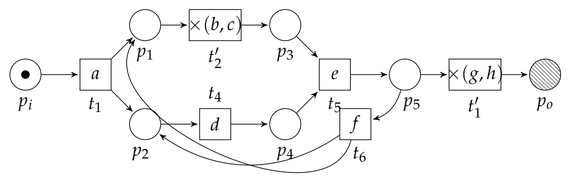
Definition 6 (Process Tree-labeled Petri-net (PTree-net))**.** *Let $\mathcal{Q}$ denote the universe of process trees. Let P denote a set of places, let T denote a set of transitions, let $F{\subseteq}(P{\times}T){\cup}(T{\times}P)$ denote the arc relation and let $\kappa\colon T{\rightarrow}\mathcal{Q}$. Tuple $N{=}(P,T,F,\kappa)$ is a Process Tree-labeled Petri net (PTree-net). $\mathcal{N}_{\mathcal{Q}}$ denotes the universe of PTree-nets.*

Given $N{\in}\mathcal{N}_{\mathcal{Q}}$, for any marking M,M' we have $\kappa(\mathcal{L}_{\mathcal{N}}(N,M,M')){\in}\mathcal{Q}^*$, and, $\mathcal{L}_{\mathcal{Q}}(\kappa(\mathcal{L}_{\mathcal{N}}(N,M,M'))){\in}\Sigma^*$, i.e., the definition of $\mathcal{L}_{\mathcal{Q}}$ ignores $\tau{\notin}\Sigma$.[3] Clearly, since PTree-nets extend the labelling function to $\mathcal{Q}$, PTree-System-nets, and, PTree-WF-nets are readily defined. We let $\mathcal{SN}_{\mathcal{Q}}$ and $\mathcal{W}_{\mathcal{Q}}$ represent their respective universes. Note that, we use a different symbol to indicate whether a labeling function maps to $\mathcal{Q}$ or $\Sigma{\cup}\{\tau\}$, i.e., $\kappa\colon T{\rightarrow}\mathcal{Q}$, whereas $\ell\colon T{\rightarrow}\Sigma{\cup}\{\tau\}$.

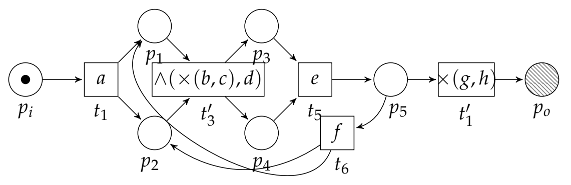
---

[2] Note that, certain proofs presented later build upon the recursive nature of the translations ad presented in Fig. 4.

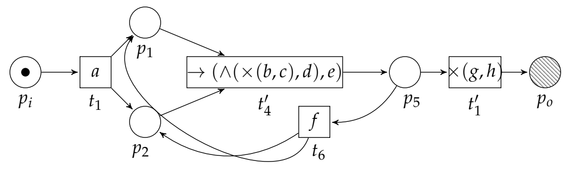
[3] Observe: $\mathcal{L}^{v}_{\mathcal{N}}(N,M,M'){=}\kappa(\mathcal{L}_{\mathcal{N}}(N,M,M'))_{\downarrow_{\Sigma}}{=}\mathcal{L}_{\mathcal{Q}}(\kappa(\mathcal{L}_{\mathcal{N}}(N,M,M')))$.



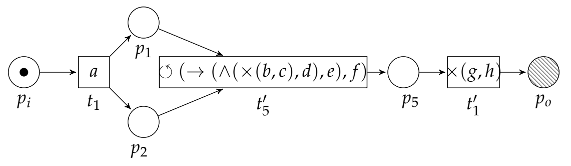
(a) Result of the first two rounds of the algorithm. The first two patterns that can be found are choice constructs, between b and c, and, g and h, respectively.



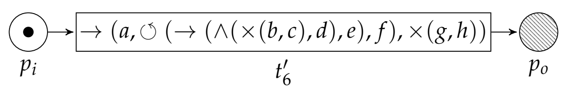
(b) Result of the third round of the algorithm on the running example. We find a concurrent construct between transition t'_2 and t_4.



(c) Result of the fourth round of the algorithm. We find a sequential construct.



(d) Result of the fifth round of the algorithm. We find a loop construct.



p_i
$\rightarrow(a,\circlearrowleft(\rightarrow(\wedge(\times(b,c),d),e),f),\times(g,h))$
t'_6
p_o

(e) Result of the final round of the algorithm. We find a sequence construct.

Figure 5. Application of the algorithm on the running example, i.e., W_1. The label of t'_6, i.e., $\kappa(t'_6)$, depicted in Fig. 5e, is the resulting process tree. The resulting process tree, i.e., $\rightarrow(a,\circlearrowleft(\rightarrow(\wedge(\times(b,c),d),e),f),\times(g,h))$, is equal to Fig. 3.

Since a PTree-net contains process trees as its labels, which can be translated into a Petri net fragment, we define a PTree-net *unfolding*, cf. Definition 7, which maps a PTree-net onto a corresponding conventional Petri net.

Definition 7 (PTree-net Unfolding). *A PTree-net unfolding $\Lambda\colon \mathcal{N}_Q \rightarrow \mathcal{N}$ is a function where, given $N=(P,T,F,\kappa)\in\mathcal{N}_Q$, $\Lambda(N)=(P',T',F',\ell)$, with:*

Let $\lambda(\kappa(t))=(P_t,T_t,F_t,p_{i_t},p_{o_t},\ell_t)$ and $\hat{\lambda}(\kappa(t))=(\hat{P}_t,\hat{T}_t,\hat{F}_t,\hat{\ell}_t), \forall t\in T$,

1. $P'=P\cup\bigcup\limits_{t\in T}\hat{P}_t$,
2. $T'=\bigcup\limits_{t\in T}\hat{T}_t$,
3. $F'=\bigcup\limits_{t\in T}\hat{F}_t\cup\bigcup\limits_{t\in T}\{(p,t)\mid p\in\bullet t\wedge t\in p_{i_t}\bullet\}\cup\bigcup\limits_{t\in T}\{(t,p)\mid p\in t\bullet\wedge t\in\bullet p_{o_t}\}$,
4. $\ell=\bigcup\limits_{t\in T}\hat{\ell}_t$.[4]

Observe that, under the assumption that we use the instantiation of λ as shown in Fig. 4, indeed, each transition in the unfolding of a PTree-net has a corresponding label in $\Sigma\cup\{\tau\}$. Furthermore, note that, unfolding the WF-net in Fig. 5a yields the original model in Fig. 2. The unfolding of the other WF-nets in Fig. 5 yields a different WF-net. However, the language of all unfolded WF-nets remains equal to the language of the WF-net in Fig. 2.

[4] Since functions are binary Cartesian products, we write set operations here.

3.3. Pattern Reduction

In this section, we describe four patterns used to identify and replace process tree behavior. Furthermore, we propose a corresponding overarching reduction function, which shows how to reduce a PTree-WF-net containing any of these patterns. However, first, we present the general notion of a *feasible pattern*. Such a feasible pattern is a system net, formed by a *place-bordered fragment* of a given PTree-net. Furthermore, the language of the unfolding of the system net needs to be equal to the language of the process tree it represents. We formalize the notion of a feasible pattern as follows.

Definition 8 (Feasible Pattern). *Let $\bigoplus$ denote the universe of process tree operators. Let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$, let $N'{=}(P',T'{=}\{t_1,...,t_n\},F',\kappa|_{T'}){\sqsubseteq}N$ be a* place-bordered fragment *of N ($N'{\sqsubseteq}N$) with corresponding input places P_i' and output places P_o'. Let $M_i{=}P_i$ and $M_f{=}P_o$. Given $\oplus{\in}\bigoplus$, $SN{=}(N',M_i,M_f){\in}\mathcal{SN}_Q$ is a feasible $\oplus$-pattern, written $\theta_\oplus(N,SN)$, iff:*

$$\mathcal{L}_Q\left(\oplus\left(\kappa\left(t_1\right),...,\kappa\left(t_n\right)\right)\right)=\mathcal{L}^v_{\mathcal{N}}\left(\Lambda\left(N'\right),M_i,M_f\right) \tag{1}$$

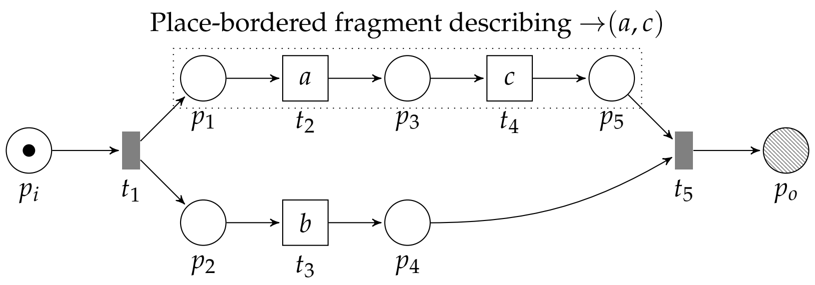
Observe that any place-bordered fragment of a WF-net that describes the same *local language* as its corresponding process tree representation is a feasible pattern. As such, any feasible pattern is *locally language-preserving*. Transforming such a detected pattern within the given WF-net is straightforward, i.e., we add a new transition t' to the WF-net with label $\oplus\left(\kappa\left(t_1\right),...,\kappa\left(t_n\right)\right)$ and pre-set P_i and post-set P_o. For example, consider the reduction of the choice construct between transitions t_2 and t_3 of Fig. 2, i.e., depicted in Fig. 5a, in which places p_1 and p_3 serve as the pre and post set of the newly added transition t_2' with label $\times(b,c)$. We formally define the notion of feasible pattern reduction as follows.

Definition 9 (Pattern Reduction). *Let $\oplus{\in}\{\rightarrow,\times,\wedge,\circlearrowleft\}$, let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$, let $N'{=}(P',T'{=}\{t_1,...,t_n\},F',\kappa|_{T'}){\sqsubseteq}N$ be a* place-bordered fragment *of N with corresponding input places P_i and output places P_o. Let $M_i{=}P_i$, $M_f{=}P_o$ and let $SN{=}\left(N',M_i,M_f\right)$ s.t. $\theta_\oplus(N,SN)$. We let $\Theta_\oplus(N,SN){=}N''{=}(P'',T'',F'',\kappa')$ denote the $\theta_\oplus(N,SN)$-reduced PTree-net, with, for $t'{\notin}T$:*

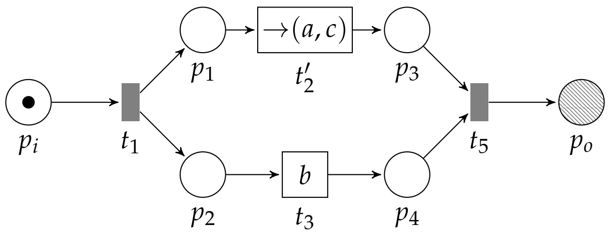
$$\begin{aligned}P''&=(P\setminus P')\cup P_i\cup P_o\\ T''&=\left(T\setminus T'\right)\cup\{t'\}\\ F''&=(F\setminus F')\cup\{(p,t')|p{\in}P_i\}\cup\{(t',p)|p{\in}P_o\}\\ \kappa'&=\kappa|_{T\setminus T'}\cup\{(t',\oplus(\kappa(t_1),...,\kappa(t_n)))\}\end{aligned}$$

A feasible pattern $\theta_\oplus(N,SN)$ is globally language preserving *iff $\forall M,M'{\in}\mathcal{M}(P'')\left(\mathcal{L}^v_{\mathcal{N}}(\Lambda(N),M,M'){=}\mathcal{L}^v_{\mathcal{N}}(\Lambda(N''),M,M')\right)$*

It is important to note that the notion of globally language-preserving is defined on the *unfolding* of a net and its corresponding reduced net. For example, consider Fig. 6. In the WF-net in Fig. 6a, we observe concurrent behavior between a sequential construct between a and c, and, activity b. The fragment formed by p_1, p_3, p_5 and t_2 and t_4, is a feasible sequence pattern. In Fig. 6b, we depict the reduced counterpart of the net in Fig. 6a, in which transitions t_2 and t_4, and the place connecting them, i.e., p_3, are replaced by transition t_2' with label $\rightarrow(a,c)$. Observe that, the language of the original net (Fig. 6a) is $\{\langle t_1,t_2,t_3,t_4,t_5\rangle,\langle t_1,t_3,t_2,t_4,t_5\rangle,\langle t_1,t_2,t_4,t_3,t_5\rangle\}$, whereas the language of the corresponding reduced net (Fig. 6b) is $\{\langle t_1,t_2',t_3,t_5\rangle,\langle t_1,t_3,t_2',t_5\rangle\}$. Consequently, the corresponding labeled languages are $\{\langle a,b,c\rangle,\langle b,a,c\rangle,\langle a,c,b\rangle\}$ and $\{\langle\rightarrow(a,c),b\rangle,\langle b,\rightarrow(a,c)\rangle\}$ respectively. The labeled language of the reduced net, after evaluating the process tree fragments inside, yields $\{\langle a,c,b\rangle,\langle b,a,c\rangle\}$, i.e., the trace $\langle a,b,c\rangle$ is not in the corresponding language. However, if we first unfold the label of t_2' in Fig. 6b, i.e., yielding the model in Fig. 6a (modulo renaming of transitions), the labeled languages of the two nets are indeed equal, i.e., they both describe $\{\langle a,b,c\rangle,\langle b,a,c\rangle,\langle a,c,b\rangle\}$.



(a) A WF-net describing concurrent behavior between a sequential construct between a and c, and, activity b. Observe that, the fragment formed by p_1, p_3, p_5 and t_2 and t_4 is a feasible sequence pattern.



(b) The (PTree)WF-net after reduction of the sequential pattern between t_2 and t_4.

Figure 6. Example WF-net (and a corresponding reduction) in which we are able to detect the feasible pattern $\rightarrow(a,c)$. The language of the original net (Fig. 6a) is $\{\langle t_1,t_2,t_3,t_4,t_5\rangle, \langle t_1,t_3,t_2,t_4,t_5\rangle, \langle t_1,t_2,t_4,t_3,t_5\rangle\}$. The language of the reduced net (Fig. 6b) is $\{\langle t_1,t_2',t_3,t_5\rangle, \langle t_1,t_3,t_2',t_5\rangle\}$. Applying the label functions on the firing sequences yields different labeled languages.

Furthermore, observe that there exist feasible patterns that are locally language-preserving, yet, not globally language-preserving. For example, consider the WF-net in Fig. 7. The place-bordered fragment formed by the subnet consisting of places p_1 and p_2 and transitions t_2 and t_3 (with the arcs connecting them), form a feasible pattern corresponding to $\circlearrowleft(a,b)$ (with M_i=$[p_1]$ and M_f=$[p_2]$). However, note that after reduction, i.e., by inserting t_2', we obtain a WF-net that no longer has the same language as the original model. This is because, before reduction, executing transition t_5 allows us to enable transition t_3. After reduction, however, this is no longer possible. Hence, whereas the observed feasible pattern is locally language preserving, i.e., when considering the elements it is composed of, it is not globally language-preserving.

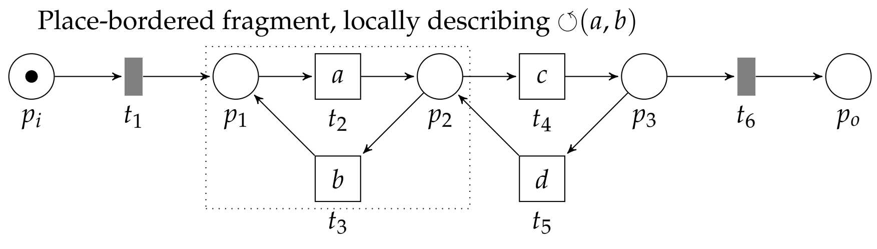
In the upcoming paragraphs, we characterize an instantiation of a global language preserving feasible pattern for each process tree operator considered in this paper. For each proposed pattern, we prove that it is both locally and globally language preserving.

3.3.1. Sequential Pattern

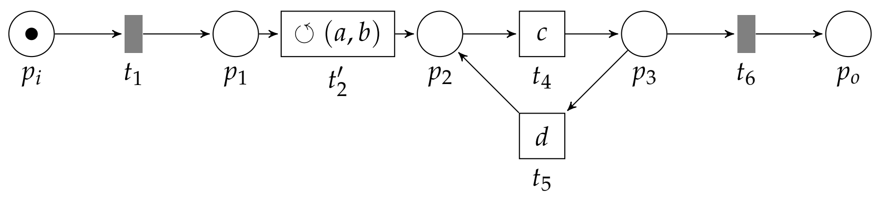
The $\rightarrow$-operator, i.e., $\rightarrow(Q_1, ..., Q_n)$, describes sequential behavior, hence, any subnet describing *strictly sequential behavior*, describes the same language. If a transition t_1 always, uniquely, enables transition t_2, which in turn enables transition $t_3, ..., t_n$, and, whenever t_1 has fired, the only way to consume all tokens from $t_1\bullet$, is by means of firing t_2, and similarly, the only way to consume all tokens from $t_2\bullet$, is by means of firing t_3, etc., then $t_1, ..., t_n$ are in a sequential relation. We visualize the $\rightarrow$-pattern in the left-hand side of Fig. 8.[5]

We formally define the notion of a sequential pattern as follows

[5] In the visualization, we omit P_i and P_o respectively, i.e., $\bullet t_1$ and $t_n\bullet$.


Place-bordered fragment, locally describing $\circlearrowleft(a,b)$
p_i
t_1
p_1
a
t_2
p_2
b
t_3
c
t_4
d
t_5
p_3
t_6
p_o

(a) The WF-net containing two local language equivalent feasible patterns.



(b) The (PTree)WF-net after reduction of the loop pattern between t_2 and t_3.

Figure 7. Example WF-net (and a corresponding reduction) in which we are able to detect feasible patterns ($\circlearrowleft(a,b)$ and $\circlearrowleft(c,d)$) that are not globally language preserving. In the exemplary reduced net (Fig. 7b), once we have executed t'_2, we are only able to execute the loop construct between t_4 and t_5.

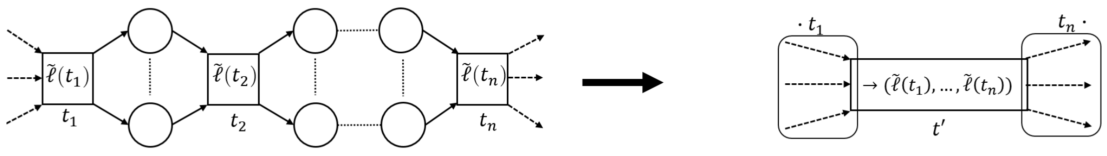


Figure 8. Schematic visualization of the $\rightarrow$-pattern reduction (dashed arcs are allowed to be part of the pattern, solid arcs are required). The post-set of each transition t_i acts as the pre-set of t_{i+1} ($1 \leq i < n$). The transition t' replacing the identified pattern inherits $\bullet t_1$ and $t_n \bullet$ (these corresponding places are not explicitly visualized in this figure). The label of t' is formed by the sequence operator defined on top of the labels of $t_1, \ldots, t_n$ respectively.

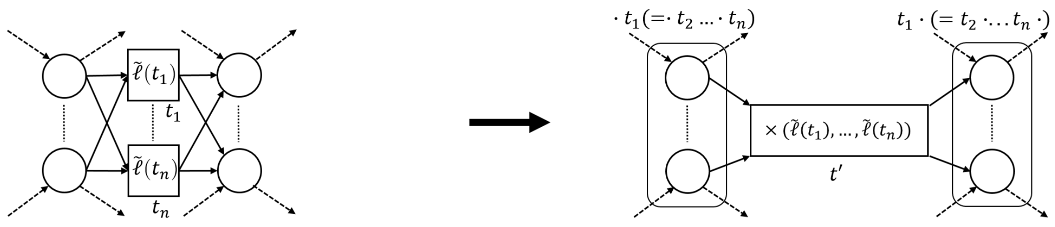
Proposition 1 ($\rightarrow$-Pattern). *Let $N=(P,T,F,\kappa) \in \mathcal{N}_Q$ and let $T'=\{t_1,\ldots,t_n\} \subseteq T$ ($n \geq 2$). If and only if:*

1. *$\forall 1 \leq i < n$ $(|t_i\bullet| \geq 1 \wedge t_i\bullet = \bullet t_{i+1})$; transition t_i enables t_{i+1},*
2. *$\forall 1 \leq i < n$ $(\forall p \in t_i\bullet\ (\bullet p = \{t_i\} \wedge p\bullet = \{t_{i+1}\}))$; enabling is unique,*

then, system net $SN=(N'=(P',T',F',\kappa|_{T'}), \bullet t_1, t_n\bullet)$ ($P_i = \bullet t_1$ and $P_o = t_n\bullet$), with $P' = \bullet t_1 \cup \bullet t_2 \cup \cdots \bullet t_n \cup t_n\bullet$, $F' = \{(x,y) \in F \mid y \in T' \vee x = t_n\}$, is a feasible $\rightarrow$-pattern.

Proof. Observe that t_1 is the only enabled transition in marking $M_i = P_i$. By definition of the proposed pattern, after firing t_i ($1 \leq i < n$), the only enabled transition is t_{i+1}. After firing t_n we reach the final marking M_f, which is a deadlock marking (the only deadlock marking) of the place-bordered subnet. Hence, any firing sequence of $\Lambda(N')$ can be written as $\sigma_1 \cdot \sigma_2 \cdots \sigma_n$, s.t., $(\Lambda(N'), M_i) \xrightarrow{\sigma_1}\!\!\!\!\rightarrow (\Lambda(N'), t_1\bullet) \xrightarrow{\sigma_2}\!\!\!\!\rightarrow \cdots \xrightarrow{\sigma_n}\!\!\!\!\rightarrow (\Lambda(N'), M_f)$. Observe that each element of σ_1 is a transition in $\lambda(\kappa(t_1))$, each element of σ_2 is a transition in $\lambda(\kappa(t_2))$, etc. Furthermore, by definition of λ, σ_1 is a firing sequence describing (when projected on its visible labels) a memmber of $\mathcal{L}_Q(\kappa(t_1))$, σ_2 describes as sequence in $\mathcal{L}_Q(\kappa(t_1))$, etc. Hence, the set of all firing sequences projected on their visible labels equals $\mathcal{L}_Q(\rightarrow(\kappa(t_1), \kappa(t_2), \ldots, \kappa(t_n)))$. $\square$

Lemma 1 ($\rightarrow$-Pattern (Proposition 1) is Globally Language-Preserving). *Let $N=(P,T,F,\kappa) \in \mathcal{N}_Q$ and let $SN=(N'=(P',T'=\{t_1,t_2,\ldots,t_n\},F',\kappa|_{T'}), \bullet t_1, t_n\bullet)$ s.t. $\theta_{\rightarrow}(N,SN)$ according to Proposition 1. The feasible pattern $\theta_{\rightarrow}(N,SN)$ is globally language-preserving.*



$\tilde{\ell}(t_1)$
t_1
$\tilde{\ell}(t_n)$
t_n
$\cdot t_1 (= \cdot t_2 \dots \cdot t_n)$
$t_1 \cdot (= t_2 \cdot \dots t_n \cdot)$
$\times(\tilde{\ell}(t_1), \dots, \tilde{\ell}(t_n))$
t'

Figure 9. Visualization of the $\times$-pattern reduction (dashed arcs are allowed to be part of the pattern, solid arcs are required). All transitions in the pattern share the same pre- and post-set. The replacing transition inherits said pre- and post-set.

Proof. Let N'' denote the net obtained after reduction (cf. Definition 9) and let $P''=(P\setminus P')\cup P_i \cup P_o$. We need to prove that $\forall M, M' \in \mathcal{M}(P'') \left(\mathcal{L}^v_{\mathcal{N}}(\Lambda(N), M, M') = \mathcal{L}^v_{\mathcal{N}}(\Lambda(N''), M, M')\right)$.

Observe that $\Lambda(N)$ and $\Lambda(N'')$ are identical, except for $\Lambda(N')$ in N and $\hat{\lambda}(t')$ (the transition-bordered unfolding of the newly added transition t') in N'' respectively. The only connections between N' in N and t' in N'' with the identical parts of the two nets are through P_i and P_o. Hence, if there exists a visible firing sequence in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N), M, M')$ that is not in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N''), M, M')$, this can only be due to different behavior described by $\Lambda(N')$ and $\lambda(t')$. However, this directly contradicts feasibility of the pattern. $\square$

3.3.2. Exclusive Choice Pattern

The $\times$-operator, i.e., $\times(Q_1, ..., Q_n)$, describes "execute either one of $Q_1, ..., Q_n$". In terms of a Petri net fragment, transitions $t_1, ..., t_n$ are in an exclusive choice pattern if their pre and post-sets are equal (yet non-overlapping). Consider Fig. 9, in which we schematically depict the $\times$-pattern. We formalize the $\times$-pattern as follows.

Proposition 2 ($\times$-Pattern). *Let $N=(P,T,F,\kappa)\in\mathcal{N}_Q$ and let $T'=\{t_1,...,t_n\}\subseteq T$ $(n\geq 2)$. If and only if:*

1. *$\bullet t_1 = \bullet t_2 = \cdots = \bullet t_n$; all pre-sets are shared among the members of the pattern,*
2. *$t_1\bullet = t_2\bullet = \cdots = t_n\bullet$; all post-sets are shared among the members of the pattern,*
3. *$\forall 1\leq i\leq n \; (\bullet t_i \neq t_i\bullet)$; self-loops are not allowed.*

then, system net $SN=(N'=(P',T',F',\kappa|_{T'}), \bullet t_1, t_1\bullet)$ $(P_i{=}\bullet t_1$ and $P_o{=}t_1\bullet)$, with $P'=\bullet t_1 \cup t_1\bullet$, $F'=\{(x,y)\in F \mid x\in T' \vee y\in T'\}$, is a feasible $\times$-pattern.

Proof. Observe that $t_1, t_2, ..., t_n$ are the only enabled transitions in N' in marking $M_i{=}P_i$. When we fire any one of these transitions, we immediately mark P_o, which is the final marking of the system net. Hence, the set of firing sequences of the system net is the union of the set of sequences $(\Lambda(N'), M_i) \xrightarrow{\sigma}\!\!\!\!\rightarrow (\Lambda(N'), M_i)$, where σ either corresponds to a labelled language of $\mathcal{L}_Q(\kappa(t_1))$, or $\mathcal{L}_Q(\kappa(t_2))$, ..., or $\mathcal{L}_Q(\kappa(t_n))$. Observe that, indeed, this set corresponds to $\mathcal{L}_Q(\times(\kappa(t_1), \kappa(t_2), \dots, \kappa(t_n)))$. $\square$

Lemma 2 ($\times$-Pattern (Proposition 2) is Globally Language-Preserving). *Let $N=(P,T,F,\kappa)\in\mathcal{N}_Q$ and let $SN=(N'=(P',T'=\{t_1,t_2,\dots,t_n\},F',\kappa|_{T'}), \bullet t_1, t_1\bullet)$ s.t. $\theta_\times(N,SN)$ according to Proposition 2. The feasible pattern $\theta_\times(N,SN)$ is globally language-preserving.*

Proof. Let N'' denote the net obtained after reduction (cf. Definition 9) and let $P''=(P\setminus P')\cup P_i\cup P_o$. Observe that, similar to the sequential pattern, $\Lambda(N)$ and $\Lambda(N'')$ are identical, except for $\Lambda(N')$ and $\hat{\lambda}(t')$ respectively. Again, the only connections between N' and t' with the identical parts of the two nets are through P_i ("entering") and P_o("exiting"). Hence, if there exists a visible firing sequence in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N), M, M')$ that is not in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N''), M, M')$, this can only be due to different behavior described by $\Lambda(N')$ and $\lambda(t')$, again contradicting the feasibility of the pattern. $\square$

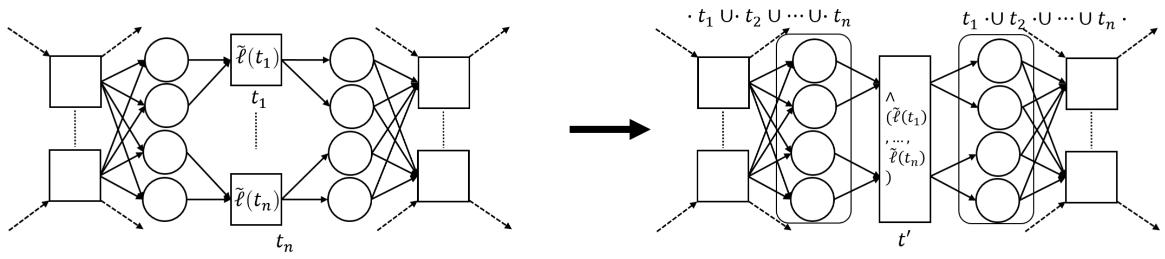

$\tilde{\ell}(t_1)$
t_1
$\tilde{\ell}(t_n)$
t_n
$\cdot t_1 \cup \cdot t_2 \cup \cdots \cup \cdot t_n$
$t_1 \cdot \cup t_2 \cdot \cup \cdots \cup t_n \cdot$
$\wedge(\tilde{\ell}(t_1), \ldots, \tilde{\ell}(t_n))$
t'

Figure 10. Visualization of the $\wedge$-pattern reduction. Transitions $t_1, ..., t_n$ have disjunct pre-sets, yet, their pre-sets have the exact same pre-sets. The same holds for the post-sets of transitions $t_1, .., t_n$. The replacing transition inherits all pre- and post-sets of $t_1, .., t_n$.

3.3.3. Concurrent Pattern

The concurrent pattern is the most complicated pattern that we consider in this paper. In the concurrent pattern, interference between its transitions is possible. The interference is achieved by requiring that the pre-sets and post-sets of the transitions do not have any overlap. Furthermore, the pre-set of the transition's pre-set places needs to be shared by all of these places, and, symmetrically, the post-set of the transition's post-set places needs to be shared by all of these places. That is, the enabling of the transitions in the pattern needs to be the same, and their post-set should jointly block any further action (i.e., within its local scope) until all places in their joint post-set are marked. Consider the left-hand side of Fig. 10, in which we schematically depict the concurrent pattern, which we formalize in Proposition 3.

Proposition 3 ($\wedge$-Pattern)**.** *Let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$ and let $T'{=}\{t_1, ..., t_n\} \subseteq T$ $(n{\geq}2)$. If and only if:*

1. *$\forall 1{\leq}i{<}j{\leq}n\ \left(\bullet t_i{\cap}\bullet t_j{=}\emptyset\right)$; no interaction between the member's pre-sets,*
2. *$\forall 1{\leq}i{<}j{\leq}n\ \left(t_i\bullet{\cap}t_j\bullet{=}\emptyset\right)$; no interaction between the member's post-sets,*
3. *$\forall 1{\leq}i{\leq}n\ (\forall p{\in}\bullet t_i\ (p\bullet{=}\ \{t_i\}))$; pre-set places uniquely connect to a member,*
4. *$\forall 1{\leq}i{\leq}n\ (\forall p{\in}t_i\bullet\ (\bullet p{=}\ \{t_i\}))$; post-set places uniquely connect to a member,*
5. *$\forall p{\in}\bullet T\ (\bullet p{\cap}\{t_1, ..., t_n\}{=}\emptyset)$; members do not influence other members,*
6. *$\forall p, p'{\in}\bullet T\ (\bullet p{=}\bullet p')$; member's pre-sets share their pre-set,*
7. *$\forall p{\in}T\bullet\ (p\bullet{\cap}\{t_1, ..., t_n\}{=}\emptyset)$; member firing does not affect other members,*
8. *$\forall p, p'{\in}T\bullet\ (p\bullet{=}p'\bullet)$; member's post-sets share their post-set,*
9. *$\forall t, t'{\in}\bullet(\bullet T)\ (\bullet t{=}\bullet t')$; pre-sets of enablers are equal,*
10. *$\forall t, t'{\in}(T\bullet)\bullet\ (t\bullet{=}t'\bullet)$; post-sets of enablers are equal,*

then, system net $SN{=}(N'{=}(P', T', F', \kappa|_{T'}), \bullet T', T'\bullet)$ $(P_i{=}\bullet T',\ P_o{=}T'\bullet)$, with $P'{=}\bullet T{\cup}T\bullet$, $F'{=}\{(x,y){\in}F|(x{\in}P'{\wedge}y{\in}T'){\vee}(x{\in}T'{\wedge}y{\in}P')\}$ is a feasible $\wedge$-pattern.

Proof. Observe that, $t_1, t_2, ..., t_n$ are the only enabled transitions in N' in marking $M_i{=}P_i$. Furthermore, since none of the transitions share any of their presets, the transitions can be fired in any order. Note that, by definition of the pattern, after all transitions have fired, we reach final marking M_f (which is a deadlock in the place-bordered system net). Hence, the labeled language described by the pattern equals $\mathcal{L}_Q(\kappa(t_1)) \leftrightharpoons \mathcal{L}_Q(\kappa(t_2)) \cdots {\leftrightharpoons} \mathcal{L}_Q(\kappa(t_n))$, which equals $\wedge(\kappa(t_1), \kappa(t_2), ..., \kappa(t_n))$. □

Lemma 3 ($\wedge$-Pattern (Proposition 3) is Globally Language-Preserving)**.** *Let $N{=}(P, T, F, \kappa){\in}\mathcal{N}_Q$ and let $SN{=}(N'{=}(P', T'{=}\{t_1, t_2, \ldots, t_n\}, F', \kappa|_{T'}), \bullet t_1, t_1\bullet)$ s.t. $\theta_\wedge(N, SN)$ according to Proposition 3. The feasible pattern $\theta_\wedge(N, SN)$ is globally language-preserving.*

Proof. Observe that, $\Lambda(N)$ and $\Lambda(N'')$ are identical, except for $\Lambda(N')$ and $\hat{\lambda}(t')$ respectively. Again, the only connections between N' and t' with the identical parts of the two nets are through P_i ("entering") and P_o("exiting"). Hence, if there exists a visible firing sequence in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N), M, M')$ that is not in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N''), M, M')$, this can only be due to different behavior described by $\Lambda(N')$ and $\lambda(t')$, again contradicting the feasibility of the pattern. □

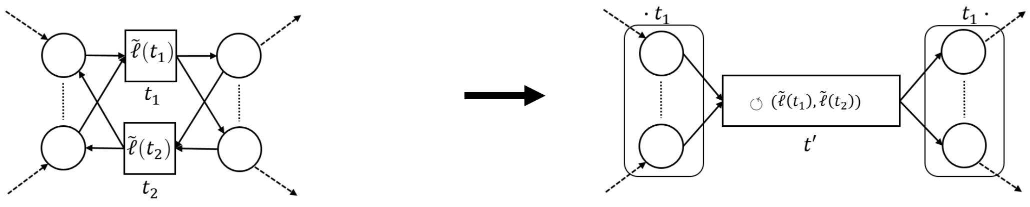


Figure 11. Visualization of the $\circlearrowright$-pattern reduction. The pre-set of transition t_1 equals the post-set of t_2 and vice-versa. The replacing transition inherits the pre- and post-set of transition t_1.

3.3.4. Loop Pattern

The final operator we consider is the $\circlearrowright$-operator, i.e., the only operator with just two children. Hence, the fragments representing a loop pattern in the Ptree-net consist of two transitions. Consider the left-hand side of Fig. 11, in which we schematically depict the loop pattern fragment. Conceptually, the loop operator requires us first to execute its leftmost child (the "do-part"). Secondly, its rightmost child is optionally executed (the "redo-part"). However, we always finish with the leftmost child. As such, the post-set of a transition corresponding to the do-part needs to be the pre-set of the transition that represents the redo-part. Furthermore, there should be no other way to enable the redo-part. Hence, the do-part needs to be the only transition that marks the pre-set of the redo-part. Similarly, to guarantee global language preservation, the do-part should be the only element in the post-set of its pre-set, i.e., no other transition may be enabled by the pre-set of the do-part. Reconsider Fig. 7, observe that p_2 contains multiple incoming and outgoing arcs, hence, it does not describe a loop-pattern for transitions t_2 and t_3, nor for transitions t_4 and t_5.

Proposition 4 ($\circlearrowright$-Pattern). *Let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$ and let $t_1{\neq}t_2{\in}T$. Iff:*

1. *$\bullet t_1{=}t_2\bullet$; pre-set of t_1 is the post-set of t_2,*
2. *$t_1\bullet{=}\bullet t_2$; pre-set of t_2 is the post-set of t_1,*
3. *$\forall p{\in}\bullet t_1\ (p\bullet{=}\{t_1\})$; t_1 is the only transition in the post-set of its pre-set,*
4. *$\forall p{\in}t_1\bullet\ (\bullet p{=}\{t_1\})$; t_1 is the only transition in the pre-set of its post-set*

then, system net $SN{=}\left(N'{=}\left(P',T',F',\kappa|_{\{t_1,t_2\}}\right),\bullet t_1,t_1\bullet\right)$ $(P_i{=}\bullet t_1,\ P_o{=}t_o\bullet)$, with $P'{=}\bullet t_1{\cup}t_1\bullet$, $T'{=}\{t_1,t_2\}$, $F'{=}\{(x,y){\in}F \mid x{\in}\{t_1,t_2\}{\vee}y{\in}\{t_1,t_2\}\}$ is a feasible $\circlearrowright$-pattern.

Proof. Observe that t_1 is the only enabled transition in N' in marking $M_i{=}P_i$. When we fire it, we immediately mark P_o, which is the final marking of the place-bordered system net. In said marking, t_2 is the only enabled transition. Firing t_2 yields us with marking M_i again. We can repeat this infinitely. Hence, the labeled language described by the pattern is $\{\sigma_1{\cdot}\sigma_1'{\cdot}\sigma_2{\cdot}\sigma_2'\cdots\sigma_n \mid n \geq 1 \wedge \forall 1{\leq}i{\leq}n\ (\sigma_i{\in}\mathcal{L}_Q(\kappa(t_1))) \wedge \forall 1{\leq}i{<}n\ (\sigma_i'{\in}\mathcal{L}_Q(\kappa(t_2)))\}$, which equals $\circlearrowright(\mathcal{L}_Q(\kappa(t_1)),\mathcal{L}_Q(\kappa(t_2)))$ $\square$

Lemma 4 ($\circlearrowright$-Pattern (Proposition 4) is Globally Language-Preserving). *Let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$ and let $SN{=}(N'{=}(P',T'{=}\{t_1,t_2,\dots,t_n\},F',\kappa|_{T'}),\bullet t_1,t_1\bullet)$ s.t. $\theta_{\circlearrowright}(N,SN)$ according to Proposition 4. The feasible pattern $\theta_{\circlearrowright}(N,SN)$ is globally language-preserving.*

Proof. Let N'' denote the net obtained after reduction (cf. Definition 9) and let $P''{=}(P{\setminus}P'){\cup}P_i{\cup}P_o$. Observe that, $\Lambda(N)$ and $\Lambda(N'')$ are identical, except for $\Lambda(N')$ and $\hat{\lambda}(t')$ respectively. Again, the only connections between N' and t' with the identical parts of the two nets are through P_i ("entering") and P_o("exiting"). In particular, when P_i is marked, the only way to mark P_o is by firing t_1, followed by an arbitrary number of $\langle t_2,t_1\rangle$ repetitions. Hence, if there exists a visible firing sequence in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N),M,M')$ that is not in $\mathcal{L}^v_{\mathcal{N}}(\Lambda(N''),M,M')$, this can only be due to different behavior described by $\Lambda(N')$ and $\lambda(t')$, contradicting the feasibility of the pattern. $\square$

Algorithm 1: WF-net Reduction

```
    input  : $W{=}(P,T,F,p_i,p_o,\ell)\in\mathcal{W}$
    output: $W'{=}(P',T',F',p_i,p_o,\kappa)\in\mathcal{W}_Q$
1  $P',T',F',\kappa\leftarrow P,T,F,\ell$;
2  while $\exists\, SN{\in}\mathcal{SN}_Q$ s.t. $\theta_{\oplus}(N,SN)$ for $\oplus{\in}\{\rightarrow,\times,\wedge,\circlearrowleft\}$ do
3      $P',T',F',\kappa\leftarrow\Theta_{\oplus}(N,SN)$;
4  return $(P',T',F',p_i,p_o,\kappa)$;
```

3.4. Algorithm

In this section, we present an algorithm that iteratively applies the reductions defined in Section 3.3. By doing so, the algorithm is able to translate a WF-net into a process tree. We prove that, if the algorithm terminates correctly, i.e., it finds a process tree, the input WF-net is sound. Moreover, we show that the language of the input WF-net equals the language of the process tree found.

Consider Algorithm 1, in which we present an algorithmic description of the reduction algorithm, on the basis of the proposed patterns in Section 3.3. As an input, the algorithm needs any WF-net W, which, by definition, is also a PTWF-net. Initially, the elements of W, excluding the initial and final place, are copied into variables P', T', F', κ. In case any pattern of the form $\theta_{\oplus}(N,SN)$ is found in N, the corresponding reduction $\Theta_{\oplus}(N,SN)$ is applied (line 3). If no more pattern is found, the algorithm returns (N,p_i,p_o,κ). The algorithm returns the most recent reduction, if no more pattern is found. Observe that, intentionally, the order and size of the patterns to be reduced is not specified, i.e., it is of no relevance to any of the lemmas and the theorems regarding the algorithms properties and correctness.

In case the obtained PTree-WF-net consists of just one transition, i.e., connected to place p_i (incoming) and place p_o (outgoing), cf. Fig. 5e, the label of the transition represents a process tree, describing the same language as the original WF-net. Furthermore, we can conclude that the original WF-net is, in fact, a sound WF-net. We prove these observations in Theorem 1. However, before this, we first present two useful lemmas. In Lemma 5, we prove that the proposed reduction rules are bidirectionally soundness preserving, i.e., if a PTree-WF-net is sound, the reduced PTree-WF-net is sound (and vice versa). In Lemma 6, we prove that, if we are able, from the initial marking $[p_i]$, to enable the observed fragment (enabling differs per fragment), then the language of the original net and the reduced net is equal (and vice versa). Observe that, trivially, the reduction rules applied on a PTree-WF-net yield a PTree-WF-net, i.e., none of the requirements of Definition 1 are violated on the resulting net.

Lemma 5 (Pattern Reduction is Soundness Preserving). *Let* $\oplus{\in}\{\rightarrow,\times,\wedge,\circlearrowleft\}$, *let* $W{=}(P,T,F,p_i,p_o,\kappa){\in}\mathcal{W}_Q$, *let* $SN{\in}\mathcal{SN}_Q$, *s.t.,* $\theta_{\oplus}(W,SN)$, *and, let* $W'{=}(P',T',F',p_i,p_o,\kappa'){=}\Theta_{\oplus}(W,SN){\in}\mathcal{W}_Q$. *$W'$ is sound iff W is sound.*

Proof. ($\Rightarrow$) Let $t'{\in}T'{\setminus}T$. Assume that W is sound, yet, W' is not sound. By definition of any reduction $\Theta_{\oplus}(W,SN)$, if W' is not safe, then W is not safe. For any $t{\in}T{\cap}T'$, if $\nexists M{\in}\mathcal{R}(W',[p_i])\ ((W',M)[t\rangle)$, then also, $\nexists M{\in}\mathcal{R}(W,[p_i])\ ((W,M)[t\rangle)$. Similarly, if $\nexists M{\in}\mathcal{R}(W',[p_i])\ ((W',M)[t'\rangle)$, then this is also holds for the transitions in SN. In case $\exists M{\in}\mathcal{R}(W',[p_i])$ s.t. $\nexists\sigma{\in}T'^{*}\left((W',M)\xrightarrow{\sigma}\!\!\!\rightarrow(W',[p_o])\right)$, then, again by definition of the reductions, also $M{\in}\mathcal{R}(W',[p_i])$ and $\nexists\sigma{\in}T'^{*}\left((W,M)\xrightarrow{\sigma}\!\!\!\rightarrow(W,[p_o])\right)$, contradicting soundness of W.

($\Leftarrow$) Assume that W' is sound, yet, W is not sound. Given that W' and W only differ on t' and SN respectively, the "non-sound" part of W needs to be part of SN. However, it is easy to see that none of the patterns defined in Section 3.3 do not describe any non-sound construct. Hence, replacing SN, implies that W' needs to be unsound, which contradicts the assumption. □

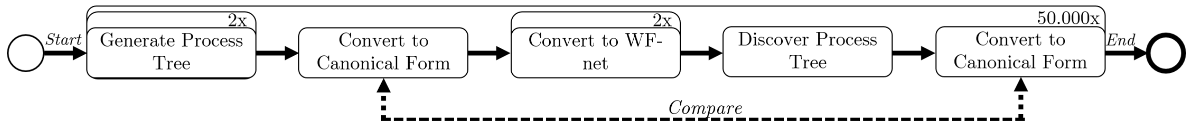


Figure 12. Overview of the experimental setup of the conducted experiments.

Lemma 6 (Pattern Reduction is Language Preserving in Λ)**.** *Let* $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$, *let* $W=(P,T,F,p_i,p_o,\kappa) \in \mathcal{W}_{\mathcal{Q}}$, *let* $SN \in \mathcal{SN}_{\mathcal{Q}}$, *s.t.,* $\theta_{\oplus}(W,SN)$, *and, let* $W'=(P',T',F',p_i,p_o,\kappa')=\Theta_{\oplus}(W,SN) \in \mathcal{W}_{\mathcal{Q}}$. $\mathcal{L}^{v}_{\mathcal{N}}(\Lambda(W))=\mathcal{L}^{v}_{\mathcal{N}}(\Lambda(W'))$.

Proof. Trivially follows from Lemmas 1-4. □

Theorem 1 (Algorithm 1 is able to find Language-Equal Process Trees)**.** *Let* $W= (P,T,F,p_i,p_o,\ell) \in \mathcal{W}$ *and let* $W'=(P',T',F',{p'}_i,{p'}_o,\kappa) \in \mathcal{W}_{\mathcal{Q}}$ *be the resulting WF-net of Algorithm 1 on W. If* $P'=\{P_i,P_o\}$, $T'=\{t\}$, *and,* $F=\{(p_i,t),(t,p_o)\}$, *then,* $\mathcal{L}_{\mathcal{Q}}(\kappa(t))=\mathcal{L}^{v}_{\mathcal{N}}(W)$.

Proof. Observe that, W is sound. Lemma 5 implies that if we (continuously) revert the reductions applied by Algorithm 1, i.e., corresponding to all intermediate assignments of W in Algorithm 1, all reverted nets are sound.[6] Observe that, Lemma 6 proves that the language of the unfoldings of all the intermediate WF-nets found is the same as well. Since the labels of the initial WF-net are all members of $\Sigma \cup \{\tau\}$, their unfolding remains the same. Hence, we deduce $\mathcal{L}_{\mathcal{Q}}(\kappa(t))=\mathcal{L}^{v}_{\mathcal{N}}(W)$. □

It is important to note that the algorithm nor the supporting lemmas and proofs specify any condition on order and the size of the pattern(s) to be reduced. In fact, the size of the pattern reduced is not of influence w.r.t. any of the correctness proofs. Note that, indeed, $\rightarrow(Q_1, \rightarrow(Q_2, \rightarrow(Q_3, \tau)))$ corresponds to $\rightarrow(Q_1, Q_2, Q_3)$, and hence, whether we iteratively find the first pattern, or apply some form of pattern maximization strategy to instantly find the latter pattern is not at all of influence w.r.t. correctness of the proposed algorithm.

4. Evaluation

In this section, we evaluate the proposed algorithm. We briefly present the implementation, after which we discuss the experimental setup and the results.

4.1. Implementation

An implementation of Algorithm 1 is available[7], i.e., built on top of the process mining framework PM4Py [10]. As indicated, the size of the patterns identified has no influence on the correctness of the algorithm. Hence, the implementation searches for *binary patterns*, yielding binary trees. Such a tree can be further reduced, e.g., $\rightarrow(Q_1, \rightarrow(Q_2, \rightarrow(Q_3, \tau)))$ corresponds to $\rightarrow(Q_1, Q_2, Q_3)$.

4.2. Experimental Setup

Here, we briefly discuss the experimental setup of our experiments. Consider Fig. 12, in which we present a graphical overview. Using an implementation of *PTandLogGenerator* [14,15], we generate process trees, using two triangular distributions for the number of activities, i.e., $\{10,20,30\}$ and $\{40,50,60\}$. The process trees are translated to WF-nets, using two different translations. One translation creates invisible *start* and *end* transitions for each operator; the other translation only does so when required (similar to Fig. 4). The first translation generates larger nets in terms of transitions/places/arcs. For each tirangular distribution/translation combination, we generate 50.000

[6] As a corollary of this fact, it follows that W is sound.
[7] https://github.com/s-j-v-zelst/pm4py-source/blob/pn_to_pt/scripts/pn_to_pt.py

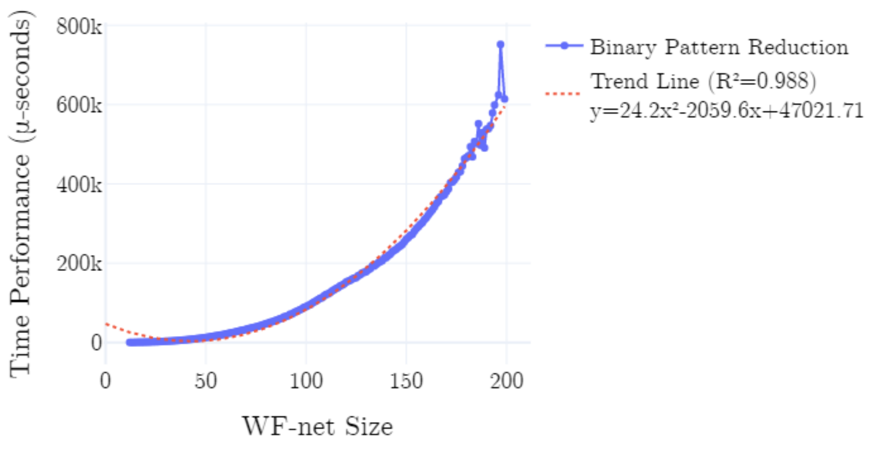


Figure 13. Average time performance of the implementation. A quadratic relation, in computation time measured in micro-seconds (μ-seconds), w.r.t. the size of the WF-net, is observable.

process trees (yielding 200.000 experiments). Finally, we compare the generated process tree in *canonical form*[16, Section 5.1], to the resulting process tree in canonical form.

4.3. Results

Here, we briefly discuss the results of the conducted experiments. Consider Fig. 13, in which we present the average time performance of the implementation on the data as generated according to the described experimental setup. We plot the time performance, conditional to the size of the input WF-net. Additionally, we plot a polynomial trend-line, computed using polynomial least squares. As is clearly observable in Fig. 13, the time performance is quadratic in the size of the net ($|P| + |T|$). This is confirmed by the R^2-score of the trend-line, i.e,. ~ 0.988. In all experiments, the canonical form of the generated process tree equals the canonical form of the (re)discovered process tree.

5. Related Work

Process trees are often used in the domain of process mining. However, a complete overview of the field is outside of scope, i.e., we refer to [1] for a gentle introduction. Similarly, we refer to [17] for an in-depth overview of process discovery algorithms, and, we refer to [18] for an overview of the sub-field of conformance checking.

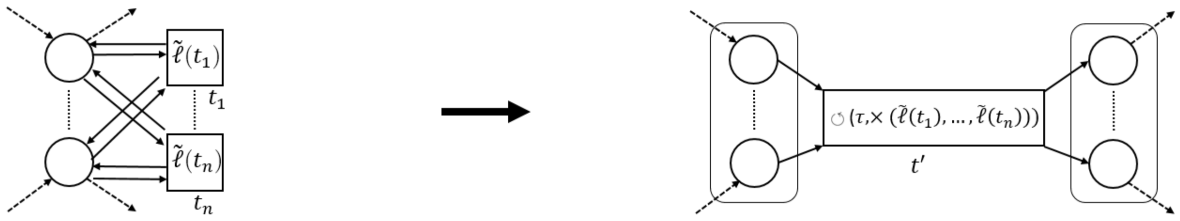
The conceptual idea of transforming a given process model in a certain formalism F into an alternative process modeling formalism F' is well-studied. Transformations of *graph-oriented* modeling formalisms, e.g., Petri nets, and *block-oriented* modeling formalisms, e.g., Process Trees, are often studied. In [19], the authors generalize work that transforms (both ways) graph-based process modeling formalisms into *Business Process Execution Language for Webservices (BPEL)* (an XML-based format). The authors characterize several strategies for such translations. In this context, the work presented in this paper belongs to the *structure-identification* category.

Of particular interest is the work of van der Aalst and Lassen [20,21], i.e., on translating of WF-nets to BPEL. In the work, XML fragments of BPEL are generated on the basis of a given WF-net. Since XML, by definition, is a tree-like data representation, BPEL and process trees are conceptually close. The algorithm replaces *components*, i.e., connected, complete subnets with a unique start and end element, i.e., such a start/end element can be either a place or a transition. The authors prove that "folding" a WF-net based on an identified component, under certain conditions, yields a sound WF-net. The folding operator for components as defined in [20] can be regarded as being very similar to the reduction (cf. Definition 9) presented in this paper. However, note that the proofs in [20] only hold for compoonents, i.e., subnets with a single entry and exit. The algorithm described in [20] is similar to the algorithm presented in this paper, i.e., searching and replacing patterns until either a WF-net with a single transition is found, or no more pattern is found. However, within the algorithm, a specific ordering for the patterns and a maximization of the pattern-size is applied. Observe that, whereas [20,21] is conceptually close to the work presented here, there are several differences as well. For example, in this work, we use system nets to represent patterns, i.e., lifting the single source/sink requirement for pattern recognition. As such, the algorithm presented in this paper is able to detect process tree fragments in WF-nets, in which the algorithm reported on in [20,21] would not find any fragments. Similarly, the algorithm presented in this paper does not impose any order on the reduction of the patterns identified, nor on their size. However, there is no clear motivation provided in [20] on the underlying reasons for imposing an order and maximization w.r.t. pattern detection.

In [22,23] the notion of the *Refined Process Structure Tree* (RPST) and its computation is introduced. The RPST is a hierarchical grouping of the edges present in a process model (defined as a *workflow graph*). As such, the given process model can be decomposed, i.e., on the basis of the hierarchy described by the RPST. Similar to [20,21], the identified model-fragments need a single source and a single sink element. The works [22,23] exemplify computing the RPST of a given BPMN model (which complies with the definition of the workflow graphs mentioned earlier), yet, indicate that the concepts can be generalized to WF-nets. However, as we show in Section 6.2, the fact that the RPST, by definition, ignores the semantics of the model provided as an input, leads the algorithm to find tree structures in unsound Workflow nets. The RPST decomposition has been exploited in various noteworthy other studies. In [24,25], the authors use the RPST to "structure acyclic process models". The core idea is to compute an RPST decomposition of a given acyclic model, which is possibly unstructured. An unstructured part of the model is recognized as a *rigid component* in the RPST decomposition. Subsequently, the behavioral ordering relations of the rigid component are computed, and a corresponding structured process model is synthesized. Since there exist process models that do not have an equivalent well-structured representation, the aforementioned work is further extended in [26], in which the authors exploit the RPST decomposition to compute a maximally structured version of the input process model. In [27], the authors extend the notion of RPSTs for *sound free-choice Wf-nets*, i.e., referred to as a WF-tree. Within a WF-Tree, certain internal vertices can be labeled as being either place-bordered, transition-bordered, or as a loop construct. As such, the authors partially annotate the RPST with behavioral information. Whereas the RPST is computed on the basis of a tree of *triconnected components*, other similar tree-based abstractions of process models have been considered as well. In [28], the authors propose to exploit the tree of *biconnected components* to check whether a given workflow net is sound on the basis of its structure. In particular, the authors show that it is sufficient to show that one biconnected subnet of the workflow net is not safe and sound, to conclude that the WF-net as a whole is not sound.

The reductions presented in this paper, alternatively to the different works on translating process modeling formalisms into each other, bear similarity to various reduction rules established on general Petri nets. The general idea of Petri net reduction (or the opposite, expansion) is a substitution of elements of a Petri net, i.e., either by a smaller or larger newly added subnet, while preserving the behavioral properties of the net. For example, in [29], the authors propose step-wise refinements of both transitions and places in Petri nets, while preserving liveness and boundedness properties of the Petri net. Similarly, in [30], the author propose a set of reduction rules for Colored WF-nets, i.e., WF-nets with additional data flow semantics. In [31], the authors present a set of reduction rules for free-choice probabilistic WF-nets, i.e., WF-nets in which transitions have an associated probability and reward. In particular, the reduction rules are proposed in order to preserve the expected reward of the workflow.

Clearly, the work presented in this paper bears similarity with the different works mentioned. However, the works in the area of process model transformation are typically not defined for WF-nets,



$\tilde{\ell}(t_1)$
t_1
$\tilde{\ell}(t_n)$
t_n
$\circlearrowleft(\tau,\times(\tilde{\ell}(t_1),\ldots,\tilde{\ell}(t_n)))$
t'

Figure 14. Schematic visualization of the $\circlearrowleft_s$-pattern reduction. The pre and post-set of transitions $t_1, t_2, ..., t_n$ are the same. In the reduction, the places are "split" into two groups, one copying all dashed incoming arcs, one copying all dashed outgoing arcs. The newly added transition t' is placed inbetween with label $\circlearrowleft(\tau, \times(\kappa(t_1), ..., \kappa(t_n)))$.

e.g., the RPST decomposition, or are more restrictive on the patterns to be replaced, e.g., the maximized unique source/sink patterns of [20]. Similarly, whereas the reduction patterns defined here do preserve the soundness of the WF-net, i.e., if the given WF-net is sound, the core of the related work in the field focuses more on the preservation of various behavioral properties.

6. Discussion

In this section, we discuss various aspects of the algorithm proposed in this paper. Firstly, i.e., in Section 6.1, we discuss the degree of extensibility of the framework, e.g., we discuss the detection of self-loops. Secondly, i.e., in Section 6.2, we provide an in-depth discussion of the relation of the proposed algorithm w.r.t. computation of the RPST decomposition. Finally, in **??**, we discuss the reducibility of arbitrary WF-nets in the context of our proposed algorithm, i.e., we show an example of simple sound WF-nets for which the algorithm cannot find a corresponding process tree.

6.1. Extensibility

Since correctness of the proposed algorithm (cf. Theorem 1) holds for any feasible pattern (cf. Definition 8) that is globally language preserving, the algorithm presented in this paper is easily extended with additional reduction rules. Hence, any system net that describes a language that is equal to the language of a process tree can be reduced (conditional to the aforementioned global language preservation). For example, consider the *self-loop reduction* visualized in Fig. 14 and defined in Proposition 5.

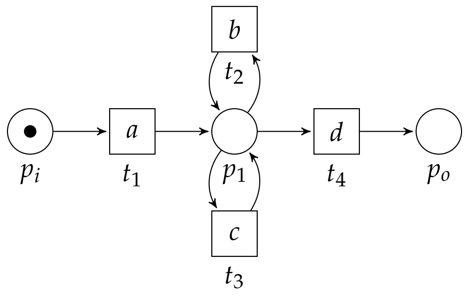
Proposition 5 ($\circlearrowleft_s$-Pattern (Self-Loop))**.** *Let $N{=}(P,T,F,\kappa){\in}\mathcal{N}_Q$ and let $T'{\subseteq}T$ ($|T|{\geq}1$). If and only if:*

1. *$\forall t,t'{\in}T'$ ($\bullet t{=}t\bullet{=}\bullet t'{=}t'\bullet$); all transitions are self-loops on the same set of places,*

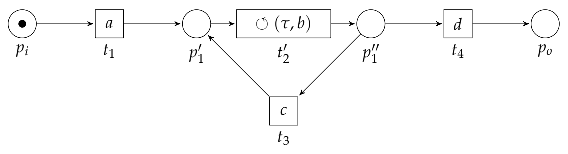
then, system net $SN{=}(N'{=}(P',T',F',\kappa|_{T'}),\bullet T,T\bullet)$, with $P{=}\bullet T$, $F{=}\{(x,y){\in}F \mid x{\in}T'{\vee}y{\in}T'\}$, is a feasible $\circlearrowleft_s$-pattern.

Observer that, applying the reduction as defined in Definition 9, again yields a self-loop. Hence, in the reduction, we split-up the places, i.e., one group of places, forming the preset of the newly generated transition t' copies all incoming arcs of the places of the pattern (excluding the connections to $t_1, ..., t_2$). The other "freshly" added place copies all outgoing arcs of the places of the pattern (again excluding the connections to $t_1, ..., t_2$).

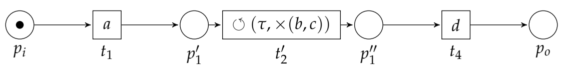
Consider Fig. 15, in which we depict a simple example of the application of the self-loop reduction as described. The transitions t_2 and t_3 in the WF-net depicted in Fig. 15a are self-loops on place p_1. In Fig. 15b, the reduction is applied only on t_2. Note that, in this model, a loop reduction can be applied yielding $\circlearrowleft(\circlearrowleft(\tau,b),c)$. Note that, first reducing t_3 is symmetrical, i.e., eventually yielding $\circlearrowleft(\circlearrowleft(\tau,c),b)$. Note that both process trees, indeed, describe the language $(b^*c^*)^*$ (i.e., when described as a regular expression). In Fig. 15c, we show the application of the self-loop reduction when it is applied directly using $T{=}\{t_2,t_3\}$. In this case, the reduction yields a new transition t_2' with label



(a) Simple WF-net, having self loop transitions t_2 and t_3.



(b) Self-loop reduction where $T'{=}\{t_2\}$.



(c) Self-loop reduction where $T'{=}\{t_2, t_3\}$

Figure 15. Example application of the self-loop pattern reduction.

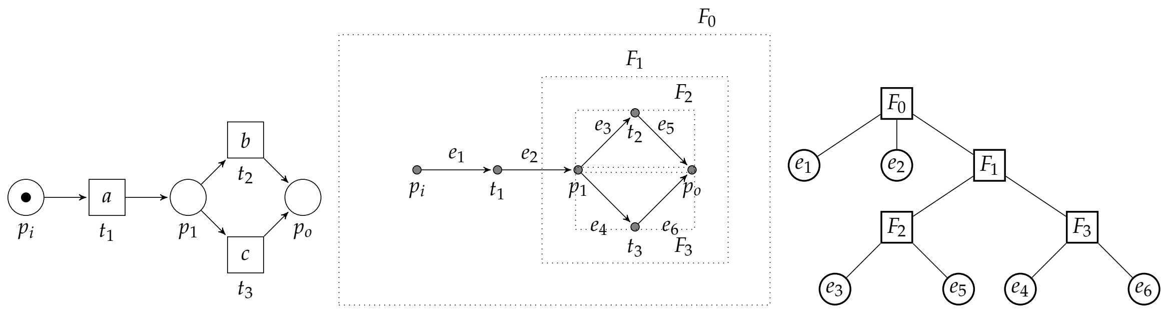
$\circlearrowleft(\tau, \times(b,c))$. Again, the language described by the process tree discovered can be described by $(b^*c^*)^*$. Finally, let X denote any of the three process tree fragments discoverable in the example describing $(b^*c^*)^*$. Observe that the reduction algorithm discovers $\rightarrow(a, X, d)$ (or any binary form thereof), which corresponds to the regular expression $a(b^*c^*)d$, which is indeed the language of the WF-net depicted in Fig. 15a.

Observe that the self-loop pattern shows that the algorithm proposed is extensible. However, in this case, the reduction step needs to be altered to avoid iteratively adding self-loop places (i.e., indefinitely). Any pattern can be reduced, i.e., as long as it is a feasible pattern that is globally language-preerving. For example, in some cases, the *inclusive or* operator is considered in the context of process trees, i.e., $\vee(Q_1, ..., Q_n)$. An inclusive or structure dictates that at least one of its children $Q_1, ..., Q_n$ is executed, yet, possibly, all are executed. The order in which the children are executed is irrelevant. Similarly, the *interleaved* operator is sometimes considered, i.e., $\leftrightarrow(Q_1, ..., Q_n)$. This operator requires that all its children are executed in any order, however, the behavior of the respective children cannot be shuffled, i.e., this is allowed by $\wedge(Q_1, ..., Q_n)$. Since both operators have a translation to a Petri net structure, these patterns can serve as a basis for reduction (potentially in a generalized form). However, note that these patterns are more involved w.r.t. the four basic patterns (and the self-loop pattern) presented in this paper.

6.2. Relation to Refined Process Structure Tree

One of the works that is conceptually very close to the work presented in this paper, is the work on the Refined Process Structure Tree (RPST) [22,23]. An RPST describes a hierarchy of sub-workflows of a *workflow graph*, such that each sub-workflow represents a connected subgraph with a single entry and single exit of control. In this context, a workflow graph is simply a two-terminal graph (TTG), i.e., a directed graph without self-loops with a unique source (s) and sink node ($t{\neq}s$), s.t. each node in the TTG is on a path from s to t. Note that a WF-net, i.e., from a graph-theoretical perspective, is a workflow graph. However, various other process modeling formalisms, e.g., BPMN, are also considered a workflow graph. Hence, an RPST can be computed on a much wider variety of process modeling formalisms, i.e., compared to the approach presented in this paper.

Formally, given a workflow graph $G_W{=}(V, E, w)$ (a multi-graph in which w assigns each edge in E to an ordered pair of nodes) an RPST is a hierarchy of *fragments*. A fragment is a subset $E'{\subseteq}E$ of arcs, s.t., the subgraph formed by E' (including their incident vertices) is connected. Furthermore, the fragment should only contain one unique *entry vertex* and one unique *exit vertex*. A vertex is an entry vertex iff none of its incoming arcs are part of E' or all of its outgoing arcs are part of E'. A vertex is an exit vertex iff none of its outgoing arcs are part of E' or all of its incoming arcs are part of E'. The RPST of a workflow graph is the set of canonical fragments, i.e., those fragments that completely contain



(a) Simple WF-net W_2. **(b)** The workflow graph of W_2, including its canonical fragments. **(c)** The RPST of W_2

Figure 16. Example of an RPST decomposition (Fig. 16c) based on the workflow graph (Fig. 16b) of a simple sound WF-net W_2 (Fig. 16a)
.

other fragments or are completely contained by other fragments, i.e., any overlap between fragments is not allowed. In [23], the authors show that computation of an RPST is equivalent to computing the tree of triconnected components on a *normalized* variant of G_W[8].

Clearly, each individual edge of a workflow graph is a fragment. Similarly, the complete set of arcs E defines a fragment. As an example, consider Fig. 16, in which we present a simple example WF-net (Fig. 16a) and its corresponding RPST decomposition (Fig. 16b and Fig. 16c). The edges (p_i, t_1) and (t_1, p_1), visualized as e_1 and e_2 are part of the root fragment, i.e. F_0, which is the complete edge set of the WF-net/workflow graph. The choice construct, i.e., connecting place p_1 and p_o to transitions t_2 and t_3 respectively, comprises fragment F_1, which is further subdivided into fragments F_2 and F_3. Note that, the process tree corresponding to W_2 is $\rightarrow(a, \times(b, c))$, i.e., consisting of 5 vertices. Hence, to translate the RPST to the corresponding process tree, we need to "collapse" F_2 and F_3 into b and c respectively. Similarly, F_1 needs to be transformed to $\times$, and, F_0 needs to be transformed to $\rightarrow$. Finally, e_1 and e_2 need to be merged into a.

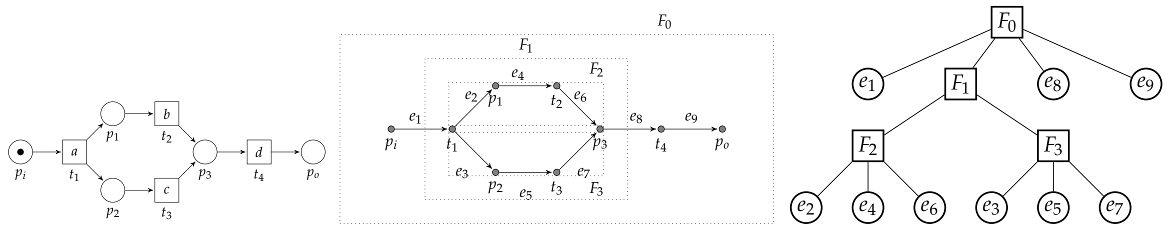
The previous example illustrates that there is no general direct correspondence between the RPST of a WF-net and a corresponding process tree that describes the same language. Furthermore, it shows that translation of the RPST to a process tree is a non-trivial operation. As the RPST decomposition ignores the semantics of a WF-net, i.e., contributing to its more general applicability w.r.t. the algorithm presented in this paper (only applicable to process models that can be transformed into a WF-net), it also exists for WF-nets that are *unsound*.

Due to the generic nature of the RPST, i.e., it defines a graph-theoretical property of a workflow graph, it (largely) ignores the semantics and graph-theoretical properties of Petri nets. In particular, as an activity in a BPMN model only consists of a single entry and exit arc, within the underlying workflow graphs these activities are simply presented as a single edge. Since transitions in Petri nets represent process activities, and, transitions are able to have multiple incoming and outgoing arcs, transitions in a WF-net cannot be represented as a single arc in the corresponding WF graph.

For example, consider Fig. 16, in which we show the RPST of a WF-net that is not *well-handled*, i.e., t_1 generates concurrent behavior, place p_3 merges both branches of concurrent behavior. Since the RPST decomposition is not aware of the conceptual difference between places and transitions, the RPST subdivides the graph into several fragments. However, this is rather inconvenient, since the WF-net is not sound at all. Hence, from the RPST decomposition itself, one cannot judge whether a corresponding process tree exists.

The reverse is also possible, i.e., given a WF-net with a corresponding process tree representation, finding an RPST can be challenging. For example, when computing the RPST of the running example

[8] Normalization is performed by splitting vertices with multiple different incoming and outgoing edges into two nodes



(a) Simple *unsound* (not *well-handled*) WF-net W_3. **(b)** The workflow graph of W_3, including its canonical fragments. **(c)** The RPST of W_3.

Figure 17. Example of an RPST decomposition (Fig. 16c) based on the workflow graph (Fig. 16b) of a simple sound WF-net W_2 (Fig. 16a)

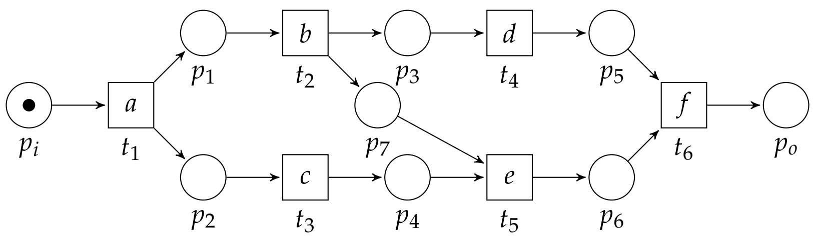


Figure 18. A Free-Choice WF-net for which the algorithm cannot find a corresponding process tree.

used in this paper, i.e., Fig. 2, the behavior formed by the loop transition t_6 cannot be decomposed into smaller chuncks[9] Observe that the algorithm proposed in this paper is able to find the loop behavior due to the simplification of the reduction steps executed prior to the loop detection in Fig. 5c.

Note that the aforementioned examples are not intended to disqualify the application of the RPST decomposition for the purpose of transforming WF-nets to process trees. However, they merely indicate that using the RPST decomposition as a basis for such translation is not a trivial adoption.

6.3. Reducibility of WF-Nets

Thus far, we have considered various system net based patterns that we reduce into a corresponding process tree notation. We have shown that if the algorithm returns a WF-net with a specific structure, its label captures a process tree describing the same language as the original WF-net. However, it remains an open question what class of WF-nets are guaranteed to result in a process tree.

Since the basic operators considered in this paper all correspond to *free-choice WF-nets*, i.e., WF-nets s.t., $\forall p{\in}P\,(|p\bullet|{=}1{\vee}\bullet(p\bullet){=}\{p\})$ (a place either has one outgoing arc, or it is the sole incoming arc of all transitions it connects to). Hence, intuitively, we suspect that the class of free choice nets always yields a corresponding process tree.

However, consider Fig. 18, in which we depict a free-choice WF-net, which the proposed algorithm is not able to reduce. Observe that the model consists of two concurrent branches, i.e., enabled by transition t_1. However, execution of transition t_5 is conditional to execution of t_2, i.e., firing both t_3 and t_2 enables transition t_5. One can look at this type of condition, i.e., induced by place p_7, as a *non-local* behavioral relation. There is interaction between members of the "upper part" and the "lower part" of the concurrent construct. Such a type of interaction cannot be modeled using a tree-based process modeling formalism.

Based on the previous example, we conclude that any class extending free-choice Petri nets might represent various WF-nets that cannot be reduced by the algorithm proposed. The notion of *block-structured* WF-nets seems to be an adequate subclass of free-choice WF-nets that can always be reduced by the model, i.e., they are often used interchangeable with process trees. However, an exact

[9] In terms of RPST, the loop structure is generating a *rigid fragment*.

structural definition of said class of WF-nets does not exist in literature (yet). For example, in [16], an informal description of block-structured WF-nets is proposed: *"A workflow net is block structured if for every place or transitions with multiple outgoing arcs, there is a corresponding place or transition with multiple incoming arcs. The parts of the net between the outgoing and incoming arcs form regions, and no arcs can exist between regions, i.e. the regions have a single entry and a single exit."* However, transforming said description into a formal, graph-theoretical property, is not trivial for cyclic models.

7. Conclusion

In this paper, we presented an algorithm to construct a process tree on the basis of a Workflow net (WF-net). The proposed algorithm replaces fragments of the WF-net that correspond to a process tree operator, i.e., by means of reduction rules. If the algorithm reduces the WF-net into a net, containing just one transition, there exists a corresponding process tree for the given WF-net, with the same language. The reduction rules proposed are bidirectionally soundness preserving. Hence, in case a process tree is found, the original WF-net is sound. We have conducted experiments using a prototypical implementation, indicating quadratic time complexity in the net and process tree rediscoverability.

Future Work We aim to extend the work presented in this paper in the following directions. We aim to provide diagnostics w.r.t. the reason why a given WF-net cannot be reduced further, e.g., by assessing if removal of certain elements of the WF-net allows for further reduction. Alternatively, it is interesting to "wrap" certain fragments of the net into an encapsulating transition, after which the search to process tree fragments is continued. Another interesting direction, as briefly discussed in Section 6, is the search for structural properties of WF-nets that directly indicate whether a given WF-net corresponds to a process tree.

1. van der Aalst, W.M.P. *Process Mining - Data Science in Action, Second Edition*; Springer, 2016.
2. Dijkman, R.M.; Dumas, M.; Ouyang, C. Semantics and analysis of business process models in BPMN. *Information & Software Technology* **2008**, *50*, 1281–1294.
3. van der Aalst, W.M.P. Formalization and verification of event-driven process chains. *Information & Software Technology* **1999**, *41*, 639–650.
4. van der Aalst, W.M.P.; Buijs, J.C.A.M.; van Dongen, B.F. Towards Improving the Representational Bias of Process Mining. SIMPDA 2011, Campione d'Italia, Italy, June 29 - July 1, 2011, Revised Selected Papers, 2011, pp. 39–54.
5. Leemans, S.J.J.; Fahland, D.; van der Aalst, W.M.P. Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings, 2013, pp. 311–329.
6. Verbeek, E.; Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. ProM 6: The Process Mining Toolkit. Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010, 2010.
7. van Dongen, B.F. BPI Challenge 2012, 2012. doi:10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F.
8. Lee, W.L.J.; Verbeek, H.M.W.; Munoz-Gama, J.; van der Aalst, W.M.P.; Sepúlveda, M. Recomposing conformance: Closing the circle on decomposed alignment-based conformance checking in process mining. *Inf. Sci.* **2018**, *466*, 55–91.
9. van Zelst, S.J.; Bolt, A.; van Dongen, B.F. Computing Alignments of Event Data and Process Models. *ToPNoC* **2018**, *13*, 1–26.
10. Berti, A.; van Zelst, S.J.; van der Aalst, W.M.P. Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science. ICPM Demo Track 2019, Aachen, Germany, June 24-26, 2019., 2019, p. 13–16.
11. van der Aalst, W.M.P. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers* **1998**, *8*, 21–66.
12. Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* **1989**, *77*, 541–580.

13. van der Aalst, W.M.P. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. Business Process Management, Models, Techniques, and Empirical Studies, 2000, pp. 161–183.
14. Jouck, T.; Depaire, B. PTandLogGenerator: A Generator for Artificial Event Data. BPM Demo Track 2016, Rio de Janeiro, Brazil, September 21, 2016, 2016, pp. 23–27.
15. Jouck, T.; Depaire, B. Generating Artificial Data for Empirical Analysis of Control-flow Discovery Algorithms - A Process Tree and Log Generator. *Bus. Inf. Syst. Eng.*, *61*, 695–712.
16. Leemans, S. Robust process mining with guarantees. PhD thesis, Department of Mathematics and Computer Science, 2017. Proefschrift.
17. Augusto, A.; Conforti, R.; Dumas, M.; Rosa, M.L.; Maggi, F.M.; Marrella, A.; Mecella, M.; Soo, A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Trans. Knowl. Data Eng.* **2019**, *31*, 686–705.
18. Carmona, J.; van Dongen, B.F.; Solti, A.; Weidlich, M. *Conformance Checking - Relating Processes and Models*; Springer, 2018.
19. Mendling, J.; Lassen, K.B.; Zdun, U. On the transformation of control flow between block-oriented and graph-oriented process modelling languages. *IJBPIM* **2008**, *3*, 96–108.
20. van der Aalst, W.M.P.; Lassen, K.B. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Information & Software Technology* **2008**, *50*, 131–159.
21. Lassen, K.B.; van der Aalst, W.M.P. WorkflowNet2BPEL4WS: A Tool for Translating Unstructured Workflow Processes to Readable BPEL. CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I, 2006, pp. 127–144.
22. Vanhatalo, J.; Völzer, H.; Koehler, J. The refined process structure tree. *Data Knowl. Eng.* **2009**, *68*, 793–818.
23. Polyvyanyy, A.; Vanhatalo, J.; Völzer, H. Simplified Computation and Generalization of the Refined Process Structure Tree. WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Papers, 2010, pp. 25–41.
24. Polyvyanyy, A.; García-Bañuelos, L.; Dumas, M. Structuring Acyclic Process Models. Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings, 2010, pp. 276–293. doi:10.1007/978-3-642-15618-2_20.
25. Polyvyanyy, A.; García-Bañuelos, L.; Dumas, M. Structuring acyclic process models. *Inf. Syst.* **2012**, *37*, 518–538. doi:10.1016/j.is.2011.10.005.
26. Polyvyanyy, A.; García-Bañuelos, L.; Fahland, D.; Weske, M. Maximal Structuring of Acyclic Process Models. *Comput. J.* **2014**, *57*, 12–35. doi:10.1093/comjnl/bxs126.
27. Weidlich, M.; Polyvyanyy, A.; Mendling, J.; Weske, M. Causal Behavioural Profiles - Efficient Computation, Applications, and Evaluation. *Fundam. Inform.* **2011**, *113*, 399–435. doi:10.3233/FI-2011-614.
28. Polyvyanyy, A.; Weidlich, M.; Weske, M. The Biconnected Verification of Workflow Nets. On the Move to Meaningful Internet Systems: OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part I, 2010, pp. 410–418. doi:10.1007/978-3-642-16934-2_29.
29. Suzuki, I.; Murata, T. A Method for Stepwise Refinement and Abstraction of Petri Nets. *J. Comput. Syst. Sci.* **1983**, *27*, 51–76. doi:10.1016/0022-0000(83)90029-6.
30. Esparza, J.; Hoffmann, P. Reduction Rules for Colored Workflow Nets. Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, 2016, pp. 342–358. doi:10.1007/978-3-662-49665-7_20.
31. Esparza, J.; Hoffmann, P.; Saha, R. Polynomial analysis algorithms for free choice Probabilistic Workflow Nets. *Perform. Evaluation* **2017**, *117*, 104–129. doi:10.1016/j.peva.2017.09.006.