The Relationship Between Workflow Graphs and Free-Choice Workflow Nets

Cédric Favre[a], Dirk Fahland[b], Hagen Völzer[a]

[a]*IBM Research - Zurich, Switzerland*
[b]*Eindhoven University of Technology, The Netherlands*

Abstract

Workflow graphs represent the main control-flow constructs of industrial process modeling languages such as BPMN, EPC and UML Activity diagrams, whereas free-choice workflow nets is a well understood class of Petri nets that possesses many efficient analysis techniques. In this paper, we provide new results on the translation between workflow graphs and free-choice workflow nets.

We distinguish workflow graphs with and without inclusive Or-logic. For workflow graphs without inclusive logic, we show that workflow graphs and free-choice workflow nets are essentially the same thing. More precisely, each workflow graph and each free-choice workflow net can be brought into an equivalent normal form such that the normal forms are, in some sense, isomorphic. This result gives rise to a translation from arbitrary free-choice workflow nets to workflow graphs.

For workflow graphs with inclusive logic, we provide various techniques to replace inclusive Or-joins by subgraphs without inclusive logic, thus giving rise to translations from workflow graphs to free-choice nets. Additionally, we characterize the applicability of these replacements. Finally, we also display a simple workflow graph with an inclusive Or-join, which, in some sense, cannot be replaced. This shows a limitation of translating inclusive logic into free-choice nets and illustrates also a difficulty of translating inclusive logic into general Petri nets.

Keywords: Workflow graphs, Petri nets, Free choice, Inclusive OR-join

1. Introduction

Different BPM tools, execution engines, and scientific analysis techniques are based on different modeling languages for business processes. This generates a general interest in translating models from one language into another. In particular, business processes are in practice often modeled in industrial languages

Email addresses: `ced@zurich.ibm.com` (Cédric Favre), `d.fahland@tue.nl` (Dirk Fahland), `hvo@zurich.ibm.com` (Hagen Völzer)

such as BPMN, EPCs, and UML activity diagrams whereas many analysis techniques, such as control-flow analysis, cost estimation, performance analysis, and process mining are based on Petri nets.

A particular appealing and well understood class of Petri nets are free-choice workflow nets. While they are expressive enough to model the most important control-flow patterns, they rule out some behavioral patterns that are often undesired, such as race conditions, where some choices in a system may become dependent on the ordering of concurrent events. These restrictions make free-choice Petri nets easier to understand and analyze (cf. discussion in [1]). In fact, for free-choice Petri nets, various analysis problems can be solved in polynomial time, which are NP-hard for general Petri nets [2–4].

The gap between industrial languages and Petri nets so far has been bridged in one direction, viz. by translating a model in an industrial language to a Petri net, e.g., [2, 5]. However, such a translation becomes less effective if the input model contains not only exclusive but also inclusive alternative branching. Existing translations to Petri nets make the inclusive logic explicit, which causes an exponential blow-up in the Petri net, which in turn affects running times of analysis techniques.

Furthermore, results of algorithms working on Petri nets currently cannot be translated easily to an industrial language for the lack of a well-understood translation mechanism from Petri nets to industrial languages.

By extending existing translation mechanisms to also translate inclusive branching without exponential blow-up, and to translate from Petri nets to industrial languages, a number of interesting use cases could be enabled. Results of various Petri-net based techniques such as process discovery or process model repair could be easier translated to industrial languages. Process analysis techniques such as verification, simulation, or performance analysis could be applicable to a larger class of industrial languages. Techniques for relating different process models to each other, e.g., process model comparison, alignment or querying from a repository, could become easier to apply to process models of different meta-models.

We formally study the problem as the relation between *workflow graphs* and Petri nets. The main control flow of a BPMN, EPC or UML activity diagram can be captured as a workflow graph. A workflow graph may contain exclusive or inclusive alternative branching as well as parallel branching of control flow. In this paper, we present new results on the translation between workflow graphs and Petri nets, in particular free-choice Petri nets.

The requirements of a translation between a workflow graph and a Petri net can vary for different use cases. To obtain general, yet useful results, we take the following requirements into account:

- A model and its translation must have *equivalent* behavior. Many notions of equivalence exist [6]. The adequacy of an equivalence for the translation depends on the use case. We will present the equivalences we use later in the paper. Note that this requirement may by itself not be challenging. For example, one can easily 'unfold' an acyclic workflow graph into its finite full behavior (i.e., computation tree) and then encode this 'unfolding' as a Petri net. Such a construction would preserve, depending on its precise execution, many popular behavioral equivalences, such as trace equivalence and bisimulation. However, the obtained translation is in general exponentially larger than the original workflow graph. Therefore,

- the size of the translated model must be manageable. An exponential blowup is usually not acceptable. We are not aware of any general translation from all workflow graphs with inclusive logic into Petri nets that preserves the behavior and does not incur an exponential blowup. Furthermore,
- the translation must preserve the structure of the original model as much as possible. This is important if we want to map analysis results between the original model and its translation. For example, in order to return to the user of an analysis technique the results in terms of the original process model or, when monitoring or administrating a process, to understand a trace or a state of the running process in terms of the original process model.

We present the following results. We first consider the simpler case of workflow graphs without inclusive logic. Although it is known that workflow graph without inclusive logic are tightly related to free-choice nets, only a translation from workflow graphs to free-choice nets, but not a reverse translation from free-choice nets to workflow graphs was published so far. We present such a reverse translation. Moreover, we show that workflow graphs and free-choice workflow nets are essentially the same thing. More precisely, each workflow graph and each free-choice workflow net can be brought into an equivalent normal form such that the normal forms are, in some sense, isomorphic. (The workflow graph is isomorphic to the graph of conflict clusters of its corresponding workflow net.) This means that, when being in normal form, the workflow graph representation and the free-choice net representation can be used completely interchangeably in every use case.

In the second part, we study workflow graphs with inclusive logic. The inclusive branching forks or joins a variable set of threads, thereby supporting various workflow patterns [7]. The inclusive Or-join (IOR-join), which has a *non-local* semantics, is difficult to translate to Petri nets because the semantics of a Petri net transition is *local*. That is, the enablement and effect of a transition in a Petri net relates only to its adjacent places—a small part of the state of the Petri net—whereas the enablement of an IOR-join may depend on the entire state of the process model.

We show that, in many cases, the IOR-join can be replaced with free-choice constructs, i.e., with a combination of exclusive and parallel joins. However, we will also display a simple workflow graph in which, in some formal sense, an IOR-join cannot be replaced. This will reveal an intrinsic limitation on the replaceability of IOR-joins and hence the translatability of the workflow graph of general process models into Petri nets. This also suggests that the expressiveness of IOR-joins extends beyond free-choice nets.

The remainder of this paper is structured as follows. In Sect. 2, we introduce the notions of a workflow net and workflow graph. In Sect. 3, we present the translation between workflow nets and workflow graphs without inclusive logic. In Sect. 4, we present our results on the translation of workflow graphs with inclusive logic.

2. Foundations

In this section, we define the necessary fundamental notions, which include workflow nets, workflow graphs, and their semantics.

2.1. Workflow nets

A *Petri net* $N = (P, T, F)$ consists of a finite set P of *places*, a finite set T of *transitions*, $P \cap T = \emptyset$, and *arcs* $F \subseteq (P \times T) \cup (T \times P)$. For any *node* $x \in P \cup T$, we write ${}^\bullet x = \{y \mid (y, x) \in F\}$ for the *predecessors* of x and $x^\bullet = \{y \mid (x, y) \in F\}$ for the *successors* of x; these notions canonically lift to sets of nodes. N is *connected* iff between any two nodes of N there is path along the arcs of N when ignoring arc directions.

A Petri net N is *(extended) free-choice* if for any two transitions $t_1, t_2 \in T$ with ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ holds ${}^\bullet t_1 = {}^\bullet t_2$. N is *simple free-choice* iff additionally ${}^\bullet t_1 = {}^\bullet t_2 = \{p\}$ for some place $p \in P$ [4].

A *conflict cluster* (P', T', F') of N is a connected subnet of N, $P' \subseteq P, T' \subseteq T, F' \subseteq F'$ such that $(P')^\bullet = T'$ and ${}^\bullet(T') = P'$ and $F' = \{(p, t) \in F \mid p \in P', t \in T'\}$. In a free-choice Petri net, a conflict cluster consists of all transitions that share a pre-place.

A workflow net $N = (P, T, F, \alpha, \Omega)$ is a Petri net (P, T, F) with a distinguished *initial* place $\alpha \in P$ and distinguished *final* transitions $\emptyset \neq \Omega \subseteq T$, such that

1. for all $p \in P$ holds ${}^\bullet p = \emptyset$ implies $p = \alpha$,
2. for each $t \in T$ holds $t^\bullet = \emptyset$ implies $t \in \Omega$, and
3. each node $x \in P \cup T$ is on a path (of edges from F) from initial place α to some final transition $\omega \in \Omega$.

Note that our definition of workflow net slightly deviates from the classical definition [1] which defines a final place instead of final transitions. Both definitions are equivalent. We chose final transitions to avoid some technicalities in the subsequent arguments.

A state of a workflow net N is expressed as a *marking* $m : P \to \mathbb{N}$, which assigns each place a non-negative number of tokens. The *initial marking* m_α of a workflow net is such that $m_\alpha(\alpha) = 1$ and $m_\alpha(p) = 0$ for all $p \in P \setminus \{\alpha\}$, and the *final marking* m_ω is such that $m_\omega(p) = 0$ for all $p \in P$. A transition $t \in T$ is *enabled* at marking m if for each $p \in {}^\bullet t$, $m(p) \geq 1$. If t is enabled, it can *occur*, which defines the *step* $m \xrightarrow{t} m_t$ of N to the *successor marking* m_t, where $m_t(p) = m(p) - 1$ if $p \in {}^\bullet t \setminus t^\bullet$, $m_t(p) = m(p) + 1$ if $p \in t^\bullet \setminus {}^\bullet t$, and $m_t(p) = m(p)$ otherwise.

2.2. Workflow graphs

A *directed multi-graph* $G = (N, E, c)$ consists of a set N of nodes, a set E of *edges* and a mapping $c : E \to (N \cup \{\text{null}\}) \times (N \cup \{\text{null}\})$ that maps each edge to an ordered pair of nodes or a null value. If $c(e) = (s, t)$, then s is called the *source* of e, t is called the *target* of e; e is an *outgoing* edge of s, and e is an *incoming* edge of t. If $s = \text{null}$, then we say that e is a *source* of the graph. If $t = \text{null}$, then we say that e is a *sink* of the graph. For a node $n \in N$, the set of incoming edges of n is denoted by $\circ n$. The set of outgoing edges of n is denoted by $n\circ$.

A *partial workflow graph (pwfg)* $W = (N, E, c, l)$ consists of a multi-graph $G = (N, E, c)$ and a mapping $l : N \to \{\text{AND}, \text{XOR}, \text{task}\}$ that associates a *logic* with every node $n \in N$. A *workflow graph* is a partial workflow graph $W = (N, E, c, l)$, such that: 1. W has exactly one source and at least one sink. 2. For each node $n \in N$, there exists a path from the source to one of the sinks that contains n. 3. A node with task logic always has a single incoming edge and a single outgoing edge.

Figure 1 depicts a workflow graph. A rectangle represents a task node. A diamond containing a plus symbol represents a node with AND logic and an empty diamond represents a node with XOR logic. A node with a single incoming edge and multiple outgoing edges is called a *split*. A node with multiple incoming edges and a single outgoing edge is called a *join*. A node with AND or XOR logic is called *gateway*.

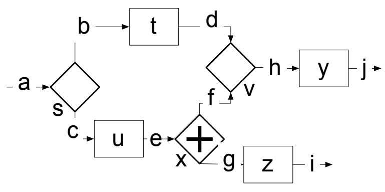


Figure 1: A workflow graph.

The semantics of workflow graphs is, similarly to workflow nets, defined as a token game. Let $W = (N, E, c, l)$ be a workflow graph. A *marking* of W is represented by tokens on the edges of W, i.e., a *marking* is a mapping $m : E \to \mathbb{N}$. We write $m[e]$ instead of $m(e)$. When $m[e] = k$, we say that the edge e *is marked with* k *tokens* in m. When $m[e] > 0$, we say that m *marks* e. The *initial marking* m_s of W is such that the source edge is marked with exactly one token in m_s and m_s does not mark any other edge. If a node n of a workflow graph has AND or task logic, executing n removes one token from each of the incoming edges of n and adds one token to each of the outgoing edges of n. If n has XOR logic, executing n removes one token from one of the incoming edges of n and adds one token to one of the outgoing edges of n.

The choice of the set of outgoing edges to which a token is added when executing a node with XOR logic is non-deterministic. In the following, this semantics is defined formally:

A triple (E_1, n, E_2) is called a *transition* if $n \in N$, and any of the following propositions:

- $l(n) = \text{AND}$ or $l(n) = \text{task}$, $E_1 = \circ n$, and $E_2 = n\circ$.
- $l(n) = \text{XOR}$, there exists an edge $e \in \circ n$ such that $E_1 = \{e\}$, and there exists an edge $e' \in n\circ$ such that $E_2 = \{e'\}$.

Let m and m' be two markings of W. A transition (E_1, n, E_2) is *enabled* in a marking m if, for each edge $e \in E_1$, we have $m[e] > 0$. A transition t can be *executed* in a marking m if t is enabled in m. When t is executed in m, a marking m' results such that: $m'[e] = m[e] - 1$ if $e \in E_1$, $m'[e] = m[e] + 1$ if $e \in E_2$, and $m'[e] = m[e]$ otherwise. We write $m_1 \xrightarrow{t} m_2$, when a transition t is enabled in a marking m_1 and its execution results in a marking m_2.

3. Translation between Workflow Graphs and Workflow Nets

We first focus on translating workflow graphs as defined in Sect. 2.2 to workflow nets, and back. IOR-logic will be introduced later. Translating workflow graphs just containing activities and AND/XOR-gateways to workflow nets has been studied earlier [8]. First, we recall this result and then extend it in two ways. We present a different set of translation rules than [8]; these new rules also allow to translate every free-choice workflow net into an equivalent workflow graph. The proof introduces a normal form for workflow graphs and a normal form for free-choice workflow nets under which both are structurally isomorphic. We position our results regarding related works at the end of this section.

3.1. Existing translation from workflow graphs to workflow nets

Aalst et al. [8] proposed a simple structural translation from workflow graphs to workflow nets in two steps. Firstly, each activity A and each AND-gateway A of the workflow graph is translated to a transition t_A, and each XOR-gateway X is translated to a place p_X. Secondly, each edge between two nodes x and y is translated to a pattern between the corresponding Petri net nodes according to the schema shown in Fig. 2.

These rules translate every given workflow graph into a free-choice workflow net [8], but they cannot be directly applied to define a translation in the reverse direction. To apply the rules of Fig. 2 in reverse order, one has to identify which right-hand side of a rule matches which part of the workflow net. This is non-trivial as the right-hand sides of the rule of Fig. 2 overlap. For instance, the right-hand side of rule Fig. 2(a) occurs within the right-hand sides of the three other rules. Thus, for translating a workflow net to a workflow graph one first has to find a partitioning of the edges of the workflow net such that each partition corresponds to a right-hand side of one rule. Now, there are workflow nets where the edges cannot be partitioned so that each partition matches the right-hand side of some rule of Fig. 2. In such a case, the choice of rules for the reverse translation becomes ambiguous or the net cannot be translated at all.

Figure 3(a) shows an example of a free-choice workflow net that cannot be translated to a workflow graph by reversing the rules of Fig. 2. The largest possible partitioning of edges with the matching rule is highlighted in Fig. 3(b).

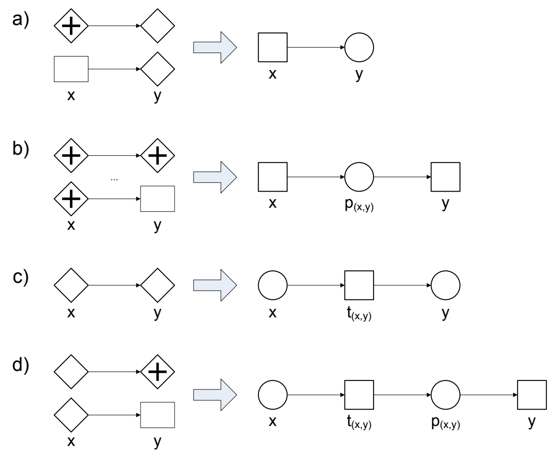


Figure 2: Translating edges of a workflow graph to corresponding Petri net patterns [8, Fig.6]

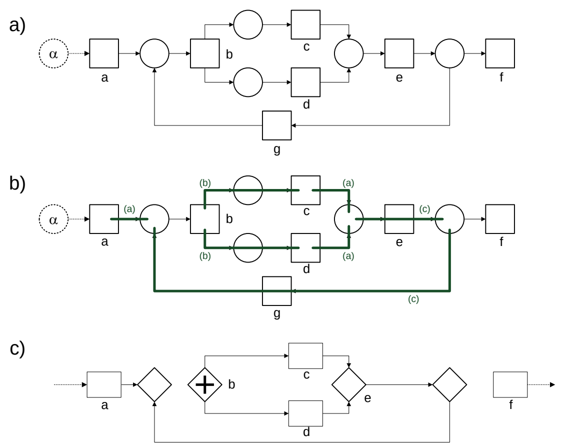


Figure 3: Translating the workflow net (a) to a workflow graph by reversing the rules of Fig. 2 fails.

In particular, Fig. 2 provides no rule to match the edges entering transitions b and f, respectively. This partitioning cannot be improved, i.e., there is no partitioning where more edges can be matched to a rule. By matching those right-hand sides of rules of Fig. 2 that can be matched and replacing them with the corresponding left-hand side, we obtain the graph of Fig. 3(c).[1] From an intuitive point of view, the graph of Fig. 3(c) does not represent the workflow net of Fig. 3(a): it lacks the edges entering b and f that were present in the

[1]Note that Aalst et al. [8] assume a workflow net to have an initial and a final transition (instead of an initial place), so the initial place is removed prior to the translation.

workflow net. Also, the graph of Fig. 3(c) is not a workflow graph as it has three source nodes a, b, and f, whereas a workflow graph has exactly one source node.

In the following, we present a different set of translation rules from workflow graphs to workflow nets that allows to translate every free-choice workflow net to a workflow graph.

3.2. Translating between workflow graphs and workflow nets

The approach of Sect. 3.1 translates a workflow graph to a workflow net by replacing an edge in the former by a pattern in the latter. In the following, we present a translation that replaces a node of the workflow graph (with its adjacent edges) by a Petri net pattern (where adjacent edges are preserved). The translation is based on the rules shown in Fig. 4.

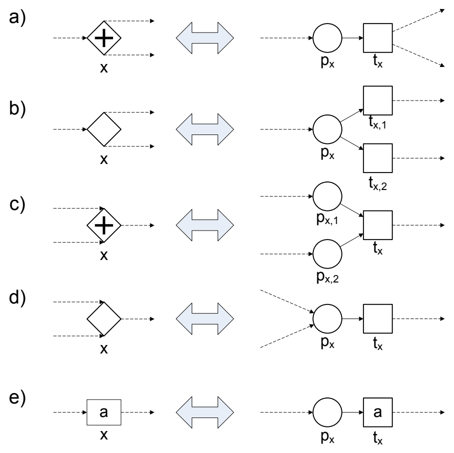


Figure 4: Translating nodes of a workflow graph to corresponding Petri net patterns

The rules of Fig. 4 only apply if the graph is in a *normal form.*

Definition 1 (Normal forms)**.** We say that a node x of a workflow graph or of a Petri net is *trivial* if it has at most one incoming and at most one outgoing edge; x is said to be *simple* if it has at most one incoming or at most one outgoing edge.

A workflow graph is *normalized* if each node is simple.

A Petri net N is *degree-normalized* if each node of N is simple. A conflict cluster (P', T', F') of N is *normalized* if there is at most one node $x \in P' \cup T'$ that is not trivial in N, i.e., x is the only node with more than one incoming or outgoing arc. N is *cluster-normalized* if each conflict cluster of N is normalized. N is *normalized* if N is degree-normalized and cluster-normalized.

Based on these normal forms, we will show a strong correspondence between workflow graphs and workflow nets, cf. Figure 5:

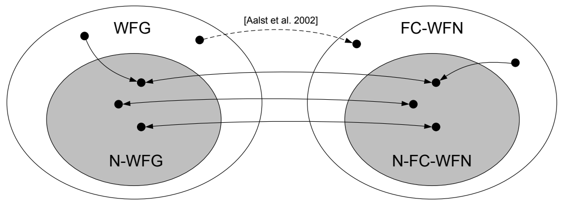


Figure 5: Relation between workflow graphs and free-choice workflow nets

1. The translation rules of Fig. 4 define a *bijection* between workflow graphs in normal form (N-WFG) and *corresponding* free-choice workflow nets in normal form (N-FC-WFN).
2. A workflow graph in normal form and its corresponding workflow net in normal form are *isomorphic.*
3. Each workflow graph (workflow net) can be transformed into one in normal form through a sequence of simple semantics-preserving transformation rules.

3.2.1. Translation from workflow graphs to workflow nets

We introduce a few notions to formalize the translation. Each left-hand side of a rule r in Fig. 4 consists of a single node x of the workflow graph. The adjacent edges (drawn dashed) indicate the environment of the rule, i.e., they distinguish split nodes from join nodes. Node x is a parameter of the rule. For a chosen node y of a workflow graph W, the rule instance $r[y]$ defines n incoming edges $\circ y$ of y in W and m outgoing edges $y\circ$ of y in W (see Sect. 2.1).

The right-hand side of r consists of a small Petri net pattern $pat(r) = (P, T, F)$ and adjacent arcs (drawn dashed) indicating the environment of the rule. Two implicit mappings relate edges of the workflow graph to nodes of $pat(r)$:

1. $tgt : \circ x \to P$ returns for each incoming edge $(y_i, x) \in \circ x$ the corresponding Petri net place $p_{x,i} = tgt_{r[x]}(y_i, x)$, and
2. $src : x\circ \to T$ returns for each outgoing edge $(x, z_j) \in x\circ$ the corresponding Petri net transition $t_{x,j} = src_{r[x]}(x, z_j)$.

In a workflow graph W, for each node x of W, there exists exactly one corresponding rule r_x of Fig. 4, which is instantiated to $r_x[x]$ according to incoming and outgoing edges. This allows us to translate an edge e of W from x to y into an edge (t, p) of a Petri net where $t = src_{r_x[x]}(e)$ and $p = tgt_{r_y[y]}(e)$.

We can now translate a normalized workflow graph W into a workflow net N by applying the following procedure:

1. For each node x of W, find the matching rule r_x of Fig. 4, and add $pat(r_x[x])$ to N.

2. For each edge e of W such that $c(e) = (x, y)$ and $x, y \neq$ null, add the edge $(src_{r_x[x]}(e), tgt_{r_y[y]}(e))$ to N.

Let $g2p(W) = N$ denote the resulting workflow net. Note that in particular nodes adjacent to a source (or sink) edge of W result in a Petri net pattern with initial place (or final transitions) of N, respectively.

3.2.2. Translation from workflow nets to workflow graphs

The rules of Fig. 4 can be applied in reverse direction. The key for identifying which rule applies on which part of a workflow net, is to decompose a normalized, free-choice workflow net N into its *conflict clusters*.

Every Petri net completely decomposes into conflict clusters and edges between conflict clusters, that is: there exists partitions $P_1 \cup \ldots \cup P_n = P$ of places, $T_1 \cup \ldots \cup T_n = T$ of transitions, $F_1 \cup \ldots \cup F_n \cup F_0$ of N such that each $(P_i, T_i, F_i), i = 1, \ldots, n$ is a conflict cluster and F_0 contains all edges from transitions to places. The following lemma shows that we have to distinguish just five cases of conflict clusters for the translation.

Lemma 1. *In a normalized free-choice workflow net, there are only the five types of conflict clusters (with adjacent arcs) which are shown as a right-hand side of the rules in Fig. 4.*

Proof. By Def. 1, a cluster in a normalized free-choice workflow net has at most one node x with more than one incoming edge or more than one outgoing edge. Moreover, the net is degree-normalized so x cannot have multiple incoming edges and multiple outgoing edges. This allows for the following cases. Node x has neither multiple incoming nor multiple outgoing edges (rule (e) of Fig. 4). Node x is a place with either multiple incoming edges or multiple outgoing edges (rules (b) and (d) in Fig. 4). Node x is a transition with either multiple incoming edges or multiple outgoing edges (rules (a) and (c) in Fig. 4). Any other conflict cluster would have more than one node with multiple incoming/outgoing edges, would not be degree-normalized, or not be free-choice. □

To translate a normalized free-choice workflow net N to a workflow graph W, apply the following procedure:

1. Decompose N into its conflict clusters $(P_1, T_1, F_1), \ldots, (P_n, T_n, F_n)$.

2. For each $i = 1, \ldots, n$ find the rule r where the right-hand side matches (P_i, T_i, F_i) and the adjacent arcs of (P_i, T_i, F_i) match the adjacent arcs of r. Add a new node x_i to W according to the left-hand side of r.

3. For each edge (u, v) of N between different conflict clusters, i.e., $(u, v) \in F_0$, $u \in T_i$ and $v \in P_j$ for $i \neq j$, $1 \leq i, j \leq n$, add to W an edge e from x_i to x_j.

4. For each cluster (P_i, T_i, F_i) where $p \in P_i \cap \{\alpha\}$, add a source edge targeting x_i to W, and for each cluster (P_i, T_i, F_i) where $t \in T_i \cap \Omega$, add to W a sink edge from x_i.

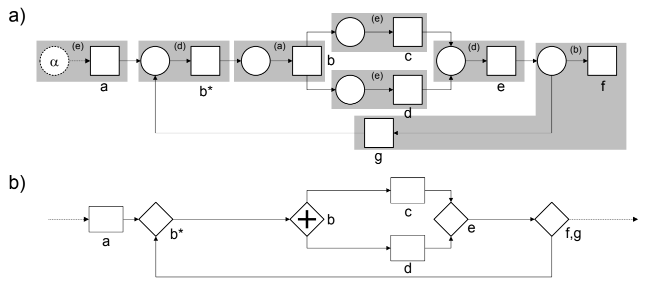


Figure 6: The workflow net (a) translates to a workflow graph (b) using the rules of Fig. 4.

Let $p2g(N)$ denote the resulting workflow graph. Figure 6 shows an example. The workflow net shown in (a) is in normal form and decomposes into the conflict clusters indicated by the grey rectangles. Each conflict cluster has a corresponding rule in Fig. 4. Applying the rule and reconnecting the edges yields the workflow graph of Fig. 6(b). Note that these rules cannot be applied on the the workflow net of Fig. 2(a) as it is not in normal form. Section 3.2.4 presents rules that allow to normalize the net of Fig. 2(a) into the net of Fig. 6(a).

3.2.3. Equivalence of normalized workflow graphs and -nets

The two translation procedures $g2p(.)$ and $p2g(.)$ already show that the rules of Fig. 4 allow to translate between workflow graphs and workflow nets. Next, we establish a stronger result about the relation between normalized workflow graphs and normalized free-choice workflow nets.

We will show that a normalized workflow graph is isomorphic to the *cluster graphs* of its corresponding normalized free-choice workflow net. The cluster graph of a free-choice workflow net N is defined as follows. Let $(P_1, T_1, F_1), \ldots,$ (P_n, T_n, F_n) be the conflict clusters of N and let F_0 be the remaining edges of N. The *cluster graph* of N is the labeled graph $G_N = (V, E, l)$ with

1. vertices $V = \{v_1, \ldots, v_n\}$ where v_i represents cluster $(P_i, T_i, F_i), 1 \leq i \leq n$,
2. edges $(v_i, v_j) \in E$ iff $(t, p) \in F_0, x \in T_i, p \in P_j, 1 \leq i, j, \leq n$, and
3. labeling l that assigns node v_i the label $l(v_i)$ according to the rules of Fig. 4, where $l(v_i)$ gets the label of a rule's left-hand side node if the cluster (P_i, T_i, F_i) and its environment match the rule's right hand side.

Theorem 2. *Normalized workflow graphs and normalized free-choice workflow nets are in bijection. Moreover, a normalized workflow graph and the cluster graph of its corresponding workflow net are isomorphic.*

Proof. First, *g2p*(.) is defined for all normalized workflow graphs, because the patterns on the left-hand side of the rules of Fig. 4 show all possible normalized nodes. Also, $p2g(.)$ is defined for all normalized, free-choice workflow nets, because every Petri net completely decomposes into its conflict clusters, and a normalized free-choice net only has the 5 shown types of conflict clusters. Further, it follows straight from the definition of $g2p(.)$ and $p2g(.)$ that $p2g(g2p(W)) = W$, for each normalized workflow graph W, and that $g2p(p2g(N)) = N$, for each normalized free-choice workflow net N. Thus, the translation is injective. Finally, by $g2p(.)$ and $p2g(.)$ being total functions, the translation is also surjective, and thus the rules of Fig. 4 define a bijection between normalized workflow graphs and normalized free-choice workflow nets.

Second, the two translations $p2g(.)$ and $g2p(.)$ both replace a workflow graph node x by a corresponding workflow net cluster, or vice versa. Thus, the cluster graph G_N is isomorphic to the workflow net W. □

Note that the isomorphism also preserves the behavior as the firing behavior of a workflow graph node is exactly the same as the firing behavior of its corresponding conflict cluster in the net.

3.2.4. Equivalence of workflow graphs and workflow nets

In this section, we show how we can translate between arbitrary workflow graphs and free-choice workflow nets. We do this by normalizing arbitrary workflow graphs and free-choice workflow nets and then using the translation above between the normal forms (cf. Fig. 5).

We normalize workflow graphs and free-choice workflow nets by using the well-known local transformation rules in Figs. 7,8,9,11 (cf. e.g., [9]). Such local transformation rules preserve semantics in a strong sense: They fully preserve the concurrency and the branching structure as they may only change the atomicity of state transitions. Thus, they preserve many strong semantic equivalences such as fully concurrent bisimulation and trace equivalence. In order that we do not have to pick a concrete such equivalence, we consider in the following two workflow graphs or workflow nets *equivalent* if one can be derived from the other by a sequence of applications of local transformation rules from Figs. 7,8,9,11.

Lemma 3. *Let W be a workflow graph. Then W can be transformed into an equivalent normalized workflow graph $norm(W)$.*

Let N be a free-choice workflow net. Then N can be transformed into an equivalent normalized free-choice workflow net $norm(N)$.

To show Lemma 3, we first consider workflow graphs. In an arbitrary workflow graph, only gateways can have a non-normal degree. The rule of Fig. 7 splits a gateway with non-normal degree into two gateways of the same type with a normal degree (there is a corresponding pattern for AND-gateways). It is easy to see that each workflow graph W can be transformed into a unique normalized workflow graph $norm(W)$.

In a similar fashion, each free-choice workflow net N can be transformed into a unique normalized free-choice workflow net. Though, N can have more involved structures that require more rules to achieve normal form. A first set

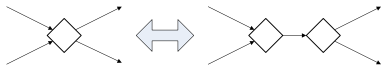


Figure 7: Rule for degree-normalizing a workflow graph.

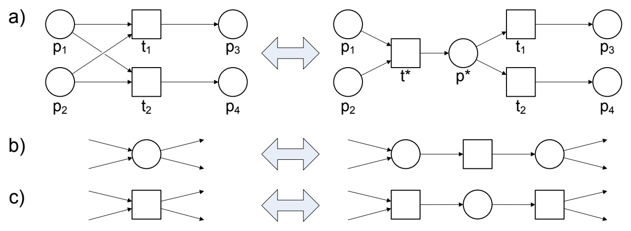


Figure 8: Rules for transforming nodes of a workflow net into simple nodes.

of rules establishes that each node in N simple, i.e., has no multiple incoming and multiple outgoing edges.

The rule of Fig. 8(a) turns an extended free-choice conflict cluster into a simple free-choice conflict cluster (see Sect. 2.1), and establishes degree-normalization of the involved nodes. The rule of Fig. 8(b) corresponds to the rule of Fig. 7 for XOR-gateways. The rule of Fig. 8(c) corresponds to the rule of Fig. 7 for AND-gateways. The rules of Fig. 8 are applied until reaching a fixed point. The resulting workflow net is degree-normalized.

To fully normalize a workflow net, also each conflict cluster has to be normal. This is achieved by the transformation rules of Fig. 9. The left hand side of each rule shows a conflict cluster that is not normal, each rule covers a possible case; the fourth case where p has multiple outgoing arcs and t has multiple incoming arcs cannot occur in a normal free-choice net. Each rule splits the non-normal conflict cluster into two simple conflict clusters. The rules of Fig. 9 are again applied until reaching a fixed point. The resulting workflow net $norm(N)$ is

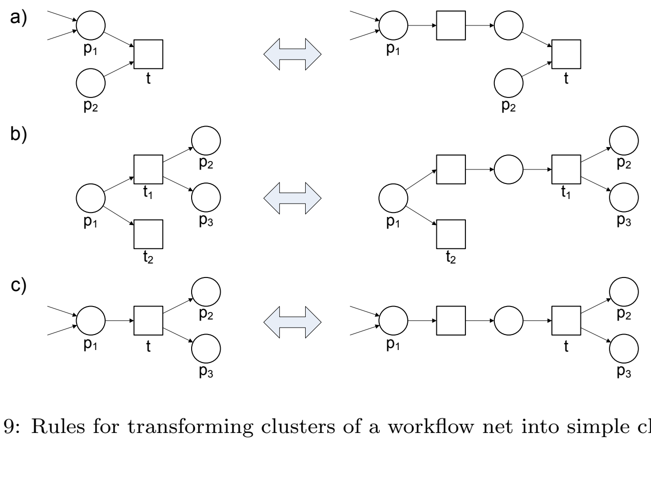


Figure 9: Rules for transforming clusters of a workflow net into simple clusters.

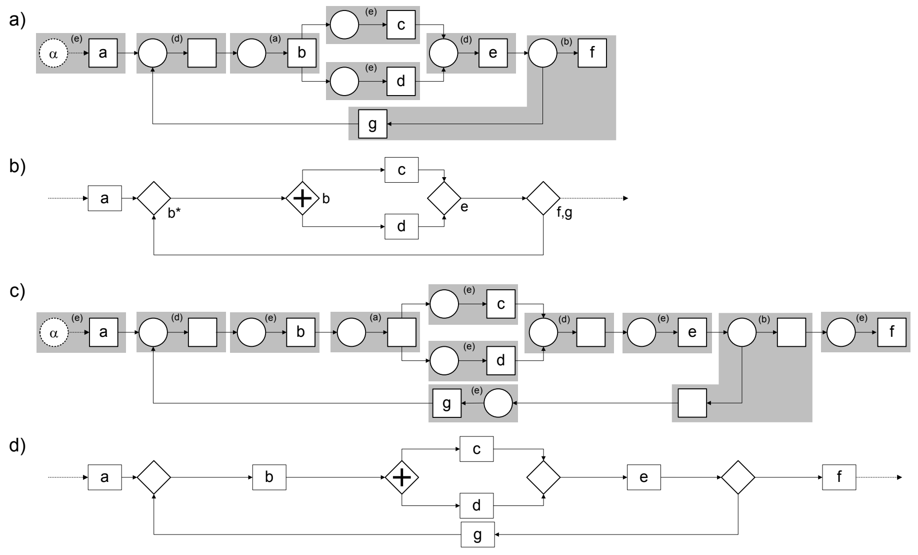


Figure 10: Translating workflow nets with task names to workflow graphs with corresponding tasks.

simple.

For example, the non-normal workflow net of Fig. 3(a) can be normalized by applying the rule of Fig. 9(c) on transition b; the resulting normalized workflow net is shown in Fig. 6(a).

3.3. Preserving tasks

Up to now, we have presented translations between workflow graphs and workflow nets that preserve their logical structure. The translation from workflow graphs to workflow nets also preserves tasks, that is, for each task A in a workflow graph, there will be a task A in the workflow net. The converse, however, is not true, which is primarily because a workflow net does not distinguish "task" transitions and "gateway" transitions. We solve this problem in the following.

A standard way of highlighting specific tasks in a workflow net is to *label* transitions with a task name $a \in \Sigma$ from some task name set Σ, or with τ (denoting a gateway transition). Figure 10(a) shows a workflow net where transitions are labeled with task names (no inscribed label means τ). Translating the workflow net of Fig. 10(a) using the rules of Fig. 4 where labels of transitions matching rule Fig. 4(e) are preserved, yields the workflow graph of Fig. 10(b). This workflow graph does not match the intuition of the workflow net of Fig. 10(a) having tasks a-g.

In order to translate each transition labeled with a task name a into a task with the same label, we have to separate transitions with task labels and transitions which correspond to gateway logic. Figure 11 shows further local structural

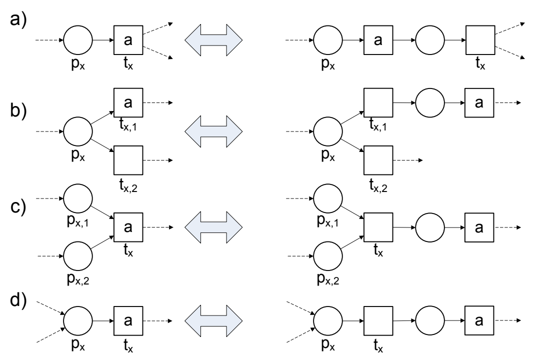


Figure 11: Transformation rules to separate labeled transitions from gateway logic.

transformation rules for achieving this separation. Each left-hand side of a rule depicts a situation in a normalized free-choice workflow net, where a conflict cluster that represents a workflow graph gateway contains a transition labeled with a task name. The corresponding right-hand side inserts a τ-labeled transition before or after this task-labeled transition such that we obtain two conflict clusters: one with the same logical structure as the given conflict cluster and with only τ-labeled transitions, and one containing just the task-labeled transition. Applying all rules on a labeled normalized, free-choice workflow net until reaching a fixed point produces the desired structure.

For example, the workflow net of Fig. 10(a) gets transformed into the net of Fig. 10(c). Translating this net by the rules of Fig. 4 produces the workflow graph of Fig. 10(d) which matches the intuition of the original workflow net. Alternatively, one could provide a richer set of translation rules which take the labeling of transitions into account to produce tasks and gateways together.

3.4. Equivalence of translations

With Theorem 2 we have proven that workflow graphs without IOR-gateways and free-choice workflow nets are identical, when considering their normal forms. Non-normal workflow graphs and workflow nets can be normalized while preserving their behavior.

As a side note, we show now that the existing translation of van der Aalst et al. [8], presented in Fig. 2 and our translation presented in Fig. 4 in fact coincide. We only sketch the proof, which is based on the observation that the rules of Fig. 2 and Fig. 4 are composed of finer-grained rules. Figure 12(a)-(d) shows four “atomic” rules for translating an incoming/outgoing edge of an XOR/AND-gateway into a corresponding Petri net pattern. The left-hand sides of the rules of Fig. 2 and of Fig. 4 are composed from the left-hand sides of the atomic rules (either by composing dangling edges or by fusing nodes of the same kind). When the corresponding right-hand sides of the the atomic rules are composed in the same way, we obtain the respective right-hand sides of the rules of Fig. 2 and of Fig. 4. This is shown exemplarily for the rule of Fig. 2(d)

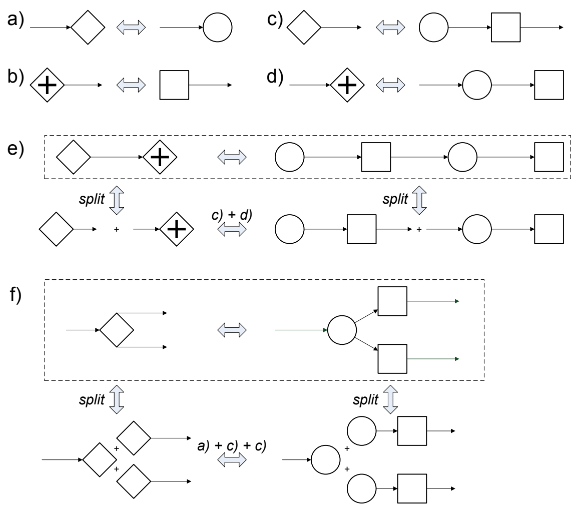


Figure 12: The "atomic" rules (a)-(d) allow to construct any of the translation rules of Fig. 2 and 4.

in Fig. 12(e), and for the rule of Fig. 4(b) in Fig. 12(f); for rule Fig. 4(e), treat the activity as an AND-gateway.

To conclude, the rules presented in this paper implement the same translation as the rules proposed in [8]. Yet, the different composition principles of the rules of this paper (Fig. 4) allowed us to identify normalized conflict clusters as the building blocks that correspond to workflow graph nodes and in turn allowed us to display a reverse translation from free-choice workflow nets to workflow graphs.

3.5. Related work

Besides [8], the problem of translating process models to Petri nets has been addressed also explicitly for individual industrial process modeling languages. Central subsets of EPC [10] and of BPMN [11] can be translated to Petri nets by translating nodes of the source model to Petri-net pattern that are then merged based on the connecting edges. The rules for translating UML Activity Diagrams to Petri nets proposed in [12] translate nodes and arcs individually, where the translation rules for arcs are context-sensitive with respect to their environment. In all cases, the resulting model is always a workflow net, but not necessarily free-choice. Also, not all constructs of industrial languages can be translated to workflow nets; for instance some EPC constructs feature non-local semantics that cannot be translated to Petri net patterns [13]. For workflow graphs with AND/XOR-gateways, the translation pattern in [10–12] yield similar results as the patterns in this paper.

None of the mentioned approaches considered the converse problem of translating workflow nets to workflow graphs (or a particular industrial language). In this context, the rules of [10] and [12] are problematic as the rules' right-hand sides are not mutually exclusive, and the problems described in Sect. 3.1 will arise for these rules as well. The rules of [11] essentially correspond to our rules and hence are reversible. Our paper provides the proof that they are indeed reversible for normalized free-choice workflow nets.

3.6. Conclusion: Taking the best of two worlds

With the strong translation between workflow graphs and free-choice workflow nets, a user may use techniques from either domain and apply it in the other. For example, process discovery algorithms such as [14, 15] can be configured to discover free-choice workflow nets from an event log; the presented translation then allows us to present the results as workflow graphs.

By the translation, any technique that transforms a free-choice workflow net N into a free-choice workflow net N' can be lifted to a technique that transforms a workflow graph G into workflow graph G'. For instance, one can repair a workflow graph G that does not conform to an event log L as follows. Translate the workflow graph G to a workflow net N; repair N through model extensions and transformation so that it conforms to L, for instance using [16]; if the repaired workflow net N' is free-choice, it can be translated to a workflow graph G' that is guaranteed to conform to L.

Workflow graphs and workflow nets can now be related to each in either representation which, in principle, allows model comparison, alignment, or querying of models of different meta-models. Furthermore, diagnostic information from Petri net-based analysis technique has a simple and direct correspondence to diagnostic information on workflow graphs. Using this correspondence, structural techniques for free-choice workflow nets such as structural reduction rules or transition invariants [4] can also be interpreted directly in terms of workflow graphs.

4. Translating Inclusive Or-Logic

In this section, we study whether a workflow graph with IOR gateways can be translated into a workflow graph with only XOR and AND gateways. We have shown in the previous section that such a workflow graph could in turn easily be translated into a free-choice workflow net.

There are two aspects of the IOR gateway: the split and the join. On the one hand, the IOR-split has a *local* semantics, i.e., the enablement and effect of a transition executing an IOR-split relates only to its adjacent places. We will present a natural translation of the IOR-split. On the other hand, the IOR-join, which has a *non-local* semantics, i.e., the enablement of the IOR-join depends on the marking of the whole workflow graph, is more difficult to translate. [2]

[2]A special case of this problem arises when we want to translate a free-choice workflow net with multiple sinks (or equivalently, a workflow graph without inclusive logic with multiple sinks) into a free-choice workflow net with a unique sink. This is because the termination semantics for multiple sinks is usually equivalent with the Or-join semantics. Note that a workflow net or -graph has a unique sink in the classical definition.

To study the IOR-join replacement, we focus on acyclic workflow graphs, for which the IOR-join semantics is simpler. This will allow us to provide replacement strategies for IOR-joins. Conversely, it will already suffice to display simple workflow graphs in which, in some formal sense, an IOR-join cannot be replaced. This will reveal an intrinsic limitation on the replaceability of IOR-joins and hence the translatability of workflow graphs into workflow nets.

Note that the replacement techniques for acyclic graphs can also be applied to cyclic graphs in case the IOR-joins can be separated in certain acyclic *single-entry-single-exit fragments* using graph parsing techniques [17, 18], cf. also [19].

We start by introducing acyclic workflow graphs with IOR logic in Sect. 4.1. In Sect. 4.2, we present two approaches to translate IOR-splits. In Sect. 4.3, we consider *local replacements* of IOR-joins. This translation strategy consists in replacing an IOR-join by a *partial workflow graph* that connects to the edges left dangling by the removal of the IOR-join. Such a local replacement fully maintains, apart from the IOR-join, the original workflow graph and thus makes the mapping to the original workflow graph trivial. Moreover, it leads to a very intuitive notion of equivalence: the partial workflow graph must have the same behavior as the IOR-join. We characterize under which conditions a local replacement is possible in an acyclic workflow graph and define a replacement for these cases. In Sect. 4.4, we consider a non-local translation strategy and characterize its condition of application. A non-local replacement essentially still retains the structure of the original workflow graph and allows us to replace some IOR-joins that have no local replacement. In Sect. 4.5, we relax our notion of replacement even more, and we show that even then, there exist simple acyclic workflow graphs that have IOR-joins that cannot be replaced. In Sect. 4.6, we relate this result and its implications to the translations of workflow graphs containing IOR-joins into Petri nets.

4.1. Acyclic workflow graphs with inclusive logic

We extend our definition of workflow graphs (See Sect. 2.1) to add the IOR logic: A *partial workflow graph (pwfg)* $W = (N, E, c, l)$ consists of a multi-graph $G = (N, E, c)$ and a mapping $l : N \to \{\text{AND}, \text{XOR}, \text{IOR}, \text{task}\}$ that associates a *logic* with every node $n \in N$. The structural restriction on a partial workflow graph to be a workflow graph remains the same. For the sake of presentation simplicity, we will only use workflow graphs in normal form, i.e., we do not use gateways with multiple incoming edges and multiple outgoing edges.

Figure 13 depicts an acyclic workflow graph. A gateway with IOR logic is represented by a diamond with a circle inside.

The elements of an acyclic workflow graphs are in a partial order defined by the flow of the graph: Let $G = (N, E, c)$ be an acyclic multi-graph. If x_1, x_2 are two distinct elements in $N \cup E$ such that there is a path from x_1 to x_2, then we say that x_1 *precedes* x_2, denoted $x_1 < x_2$, and x_2 *follows* x_1.

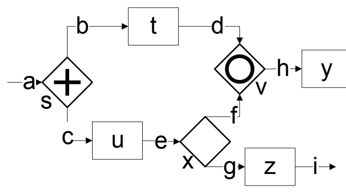


Figure 13: A workflow graph.

If a gateway n has IOR logic, n is enabled if and only if at least one of its incoming edges is marked and there is no marked edge that precedes a non-marked incoming edge of n. In Fig. 13, the marking $[d, e]$ would not enable v, because the marked edge e precedes the non-marked incoming edge f of v. The marking $[d, g]$, would enable v.

When n executes, it removes one token from each of its marked incoming edges and adds one token to a non-empty subset of its outgoing edges. This IOR semantics, which is explained in detail elsewhere [19], complies with the BPMN 2.0 standard and BPEL's dead path elimination [20]. As for XOR-splits, the choice of the set of outgoing edges to which a token is added when executing an IOR-spit is non-deterministic.

More formally: Let $W = (N, E, c, l)$ be an acyclic workflow graph. In addition to the definition of transition in Sect. 2.2, a triple (E_1, n, E_2) is also called a *transition* if $n \in N$, $l(n) =$ IOR, $E_1 \subseteq \circ n$, $E_2 \subseteq n\circ$, and E_1 and E_2 are non-empty. A transition (E_1, n, E_2) such that $l(n) = IOR$ is *enabled* in a marking m if, for each edge $e \in E_1$, we have $m[e] > 0$, $E_1 = \{e \in \circ n \mid m(e) > 0\}$ and for every edge $e \in \circ n \setminus E_1$, there exists no edge e', marked in m, such that $e' < e$.

In the rest of the paper, we will work with the notions of workflow graphs such as execution sequence, reachable marking, and soundness which we introduce now:

An *execution sequence* of W is a finite alternating sequence $\sigma = \langle m_0,\ t_0,\ m_1,\ ...,\ m_n\rangle$ of markings m_i of W and transitions $t_i = (E_i, n_i, E'_i)$ such that, for each $i \geq 0$, t_i is enabled in m_i and m_{i+1} results from the execution of t_i in m_i. We write $m_1 \xrightarrow{\sigma} m_2$ when m_1 is the first marking and m_2 is the last marking of σ. An execution sequence σ *marks* an edge e if there exists a marking of σ that marks e.

An *execution* of W is a (finite) execution sequence $\sigma = \langle m_0, ..., m_n\rangle$ of W such that $m_0 = m_s$ and there is no transition enabled in m_n. As the transition between two markings can be easily deduced, we often omit the transitions when representing an execution or an execution sequence, i.e., we write them as sequence of markings. We only consider finite executions and finite execution sequences because we only discuss acyclic workflow graphs.

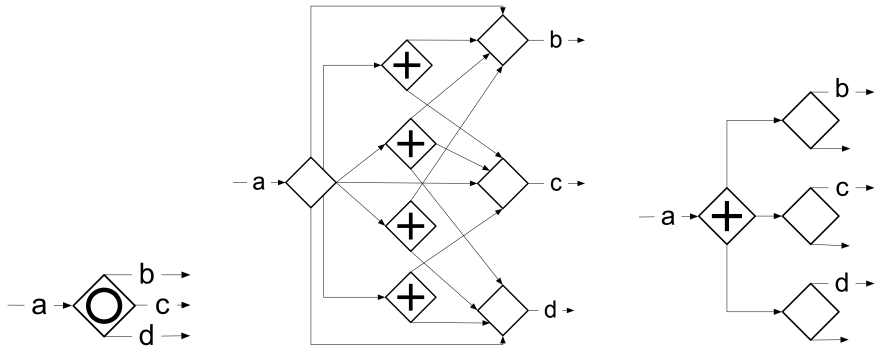
Let m be a marking of W, m is *reachable from* a marking m' of W if there exists an execution sequence $\sigma = \langle m_0, ..., m_n\rangle$ of W such that $m_0 = m'$ and $m = m_n$. The marking m is a *reachable marking* of W if m is reachable from m_s.

The marking m is a *(local) deadlock* if there exists a non-sink edge $e \in E$ that is marked in m and e is marked in all the markings reachable from m. We say that W is *deadlock-free* if there exists no reachable marking m of W such that m is a deadlock. The marking m is a *lack of synchronization* (or *unsafe*) if there exists an edge $e \in E$ that is marked by more than one token in m. We say that a workflow graph W *contains* a lack of synchronization if there exists a reachable marking m of W such that m is a lack of synchronization. A workflow graph is *sound* if it is deadlock-free and does not contain a lack of synchronization. Note that this notion of soundness is equivalent to the usual notion of soundness used for workflow nets (see for e.g. [8]).

4.2. Translating IOR-splits

As described above, an IOR-split consumes a token from its single incoming edge, and it produces a token on each of a non-empty subset of its outgoing edges. This semantics is local and it can be translated to XOR- and AND-logic as shown in Fig. 14(b).

The replacement illustrated by Fig. 14(b), replacing the IOR-split from Fig. 14(a), creates an XOR-split and a branch for each possible assignment of the outgoing edges of the IOR-split. This replacement faithfully mimics the semantics of the IOR-split and fully preserves the structure of the original workflow graph. Unfortunately, the resulting workflow graph grows exponentially with respect to the number of outgoing edges of the IOR-join.



(a) An IOR-split. (b) A faithful translation. (c) A simpler translation.

Figure 14: Two possible ways to replace an IOR-split.

Another translation for an IOR-split is shown in Fig. 14(c). This replacement only adds two edges and one gateway per outgoing edge. However, this replacement has two drawbacks: 1. It creates an additional sink per outgoing edge, which can be a problem for use cases requiring a single sink. 2. It does not enforce that at least one outgoing edge carries a token. In some use cases, this can be enforced by the user through the specification of data-related conditions specifying when each outgoing edge carries a token. Therefore, for a use case like translating a BPMN 2.0 process to be executed on an execution engine that does not support IOR-logic directly, this replacement can be adequate. For use cases where the data-related conditions are abstracted, like most control-flow analysis, this replacement is not adequate.

However, both ideas, from Fig. 14(b) and from Fig. 14(c), can be combined into a third translation, which is shown in Fig. 15. There, a first decision makes sure that at least one of the outcomes b, c, or d takes place, and afterwards, a set of pairwise concurrent decisions chooses additional optional outcomes. This is a faithful translation where the number of gateways is linear and the number

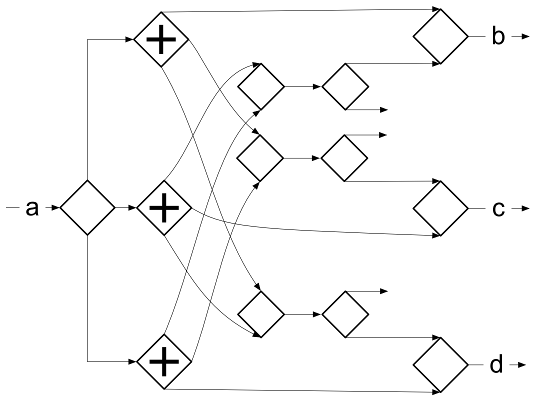
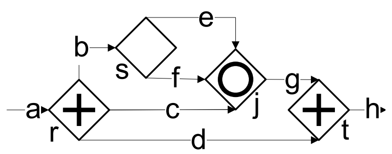


Figure 15: A faithful quadratic translation of an IOR-split creating multiple sinks.

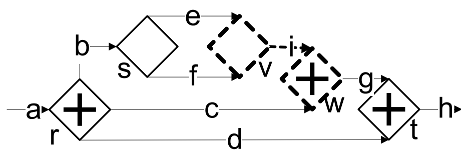
of edges is quadratic in the number of outgoing edges of the original IOR-split. It creates a linear number of additional sinks. Note that multiple sinks can be merged into a a single sink by using an IOR-join. The IOR-join can then in turn be replaced by XOR- and AND-logic as we describe later in Sections 4.3 and 4.4. Note that such a replacement is not always local, i.e., it does not necessarily fully preserve the structure, cf. Section 4.4.

4.3. Local replacements for IOR-joins

Fig. 16 illustrates an example of a *local replacement* of an IOR-join. In this example, the IOR-join j is replaced by the partial workflow graph composed of the nodes v and w, and the edge i. As a graphical convention, we represent the elements of the original workflow graph using solid lines and the elements introduced to replace an IOR-join using dashed lines. In the following, we omit tasks in the workflow graphs when they are not relevant for our discussion.



(a) A workflow graph.



(b) Local replacement of j.

Figure 16: An example of local replacement where the IOR-join j is replaced by the partial workflow graph composed of the nodes v and w, and the edge i.

4.3.1. Local replacement and equivalence

A local replacement of an IOR-join j is a partial workflow graph R that connects to the edges left dangling by the removal of j. Note that j and R have exactly the same set of incoming and outgoing edges. Apart from the IOR-join, a local replacement preserves the original workflow graph, which makes it very easy to relate the original and the translated workflow graph. We formalize a local replacement as follows:

Definition 2 (Local replacement)**.** Let $W = (N, E, c, l)$ be a workflow graph and j be a node in N. Let $R = (N'', E'', c'', l'')$ be a partial workflow graph such that for each node $n \in N''$, $l''(n) = \text{XOR}$ or $l''(n) = \text{AND}$, $N \cap N'' = \emptyset$, and $E \cap E'' = \emptyset$.

A *local replacement* of j in W by R results in a workflow graph $W' = (N', E', c', l')$ such that:

- $N' = N \setminus \{j\} \cup N''$,
- $E' = E \cup E''$,
- $c'(e) = c''(e)$ when $e \in E''$,
  $c'(e) = c(e) = (s, t)$ when $e \in E$ and $s \neq j \neq t$,
  $c'(e) = (s, t)$ such that $s \in N''$ and $t \in N$ if $e \in E$ and $c(e) = (j, t)$,
  $c'(e) = (s, t)$ such that $t \in N''$ and $s \in N$ if $e \in E$ and $c(e) = (s, j)$,
- $l'(n) = l(n)$ when $n \in N \setminus j$, $l'(n) = l''(n)$ when $n \in N''$, and
- each element $x \in N'' \cup E''$ is on a path in W from an edge $e_{in} \in E$ such that $c(e_{in}) = (s, j)$ to the edge $e_{out} \in E$ such that $c(e_{out}) = (j, t)$.

Intuitively, the workflow graph W' resulting from the local replacement of an IOR-join j in a workflow graph W by a partial workflow graph R is *equivalent* to W if R has the same "behavior" as j. We will now formalize this intuition.

Let, in the rest of this section, $W = (N, E, c, l)$ be a workflow graph containing an IOR-join j and $W' = (N', E', c', l')$ be a workflow graph obtained by local-replacement of j by a partial workflow graph R. We first introduce the notions of replacement transition and replacement execution sequence:

Definition 3 (Replacement transition and replacement sequence)**.** A transition (E_1, n, E_2) of W' is a *replacement transition* if $n \in (N' \setminus N)$. An execution sequence σ of W' is a *replacement execution sequence* if each transition of σ is a replacement transition and after σ no replacement transition is enabled and no edge $e \in E' \setminus E$ is marked.

We can now define a notion of equivalence between W and W':

Definition 4 (Equivalence of a local replacement)**.** W and W' are *equivalent* if the following two conditions are met:

1. Let m_1 and m_2 be two reachable markings of W. For any transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W, there exists a replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W'.

2. Let m_1 and m_2 be two reachable markings of W' such that m_1 and m_2 only mark edges in E. For any replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W', there exists a transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W.

4.3.2. Characterization of locally replaceable IOR-joins and replacement technique

In the following, we give a local replacement technique which, as we will see later, can provide a local replacement for any IOR-join that can be replaced locally. Then, we characterize under which conditions an IOR-join can be replaced locally, i.e., regardless of the replacement technique. This result is an extension of a technique [21] that completes a workflow graph with multiple sinks to obtain a workflow graph with a single sink.

Some IOR-joins can easily be replaced locally: It is clear that we can replace an IOR-join by an AND-join if all its incoming edges are marked every time it is executed, and by an XOR-join if only one of its incoming edge is marked every time it is executed [22]. For an acyclic workflow graph, we have shown elsewhere [23] how to compute these properties in quadratic time with respect to the size of the workflow graph. Furthermore, a workflow graph completion heuristic based on the *refined process structure tree* [21] can also provide a local replacement for some IOR-joins.

The idea behind the local replacement that we propose is to merge groups of edges which are never marked together with an XOR-join using enough edges to ensure that an edge of the group is marked during any execution where the IOR-join is executed. Then we join the outgoing edges of the created XOR-joins by an AND-join. This strategy also supports the simple cases described earlier. After defining this strategy, we will present examples of this replacement and discuss the limitations of local replacements in Sect. 4.3.3.

First, we define the notions of test and cover which will allow us to formulate the replacement based on covers:

Definition 5 (Mutually exclusive edges, test, and cover)**.** Let $W = (N, E, c, l)$ be a workflow graph.

Two edges in E are *mutually exclusive* if there exists no execution of W which marks both edges.

A *test of* an edge $e \in E$ is a set $T_e \subseteq E$ of pairwise mutually exclusive edges such that an execution σ of W marks e if σ marks one of the edges in T_e. If $X \subseteq E$ is a subset of edges such that $T_e \subseteq X$, we say that T_e is a test *over* X.

Let $X \subseteq E$ and $e \in E$. A *cover of* X *with respect to* e is a set $\mathcal{C}$ of tests of e over X such that each edge in X belongs to a test in $\mathcal{C}$.

Note this definition allows a test to contain a single edge. In Fig. 16(a), the tests $T_1 = \{e, f\}$ and $T_2 = \{c\}$ of g form a cover $\mathcal{C} = \{T_1, T_2\}$ of $\circ j$ with respect to g. We now describe how to obtain a local replacement of an IOR-join using a cover of its incoming edges with respect to its outgoing edge. We shall see later that such cover does not always exist. Fig. 17 illustrates the structure of the replacement:

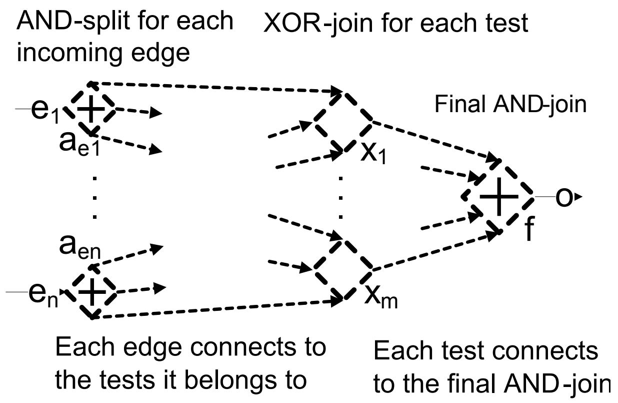

AND-split for each incoming edge
XOR-join for each test
Final AND-join
e_1
a_{e1}
x_1
a_{en}
e_n
x_m
o
f
Each edge connects to the tests it belongs to
Each test connects to the final AND-join

Figure 17: Canvas of cover-based replacement.

Definition 6 (Cover-based replacement)**.** Let j be an IOR-join in a workflow graph W and o be the outgoing edge of j. Let $\mathcal{C}$ be a cover of $\circ j$ with respect to o.

Let the $R = (N'', E'', c'', l'')$ be a partial workflow graph defined as follows:

- N'' contains: an AND-split a_e for each edge $e \in \circ j$, an XOR-join x_i for each test $T_i \in \mathcal{C}$, and one AND-join f.
- For each test $T_i \in \mathcal{C}$, for each edge $e \in T_i$, E'' contains an edge from the AND-split a_e to the XOR-join x_i. For each XOR-join x_i, E'' contains an edge from x_i to the AND-join f.

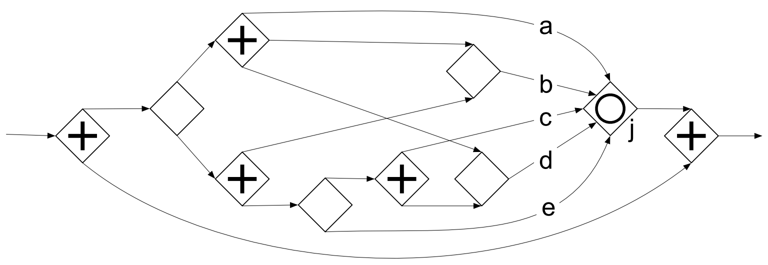
The *Cover-based replacement* (*C-replacement* for short) of j replaces j by R as follows: The target of each (previously) incoming edge e of j is set to be a_e. The source of the (previously) outgoing edge o of j is set to f.

Note that when an edge e in $\circ j$ belongs to only one test, the AND-split a_e has a single outgoing edge and can be removed. When a test T_i contains only one edge, the XOR-join x_i has a single incoming edge and can be removed.

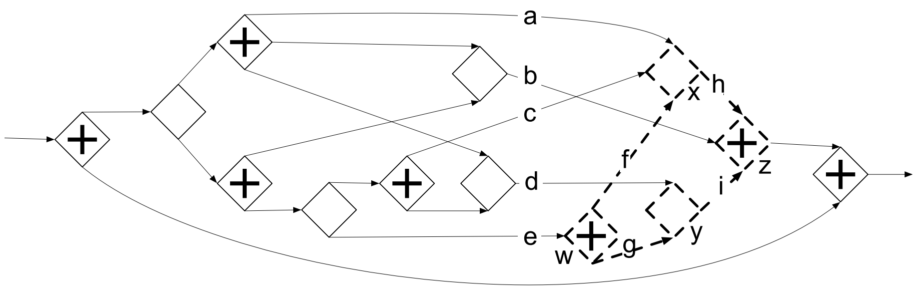
The local replacement illustrated by Fig. 16(a) and Fig. 16(b) is the result of a C-replacement using the cover $\mathcal{C} = \{\{e, f\}, \{c\}\}$ of $\circ j$ with respect to the outgoing edge g of j. Because each edge is used only in one test, there is no AND-split necessary. Moreover, the test $\{c\}$ contains only one edge, therefore it does not require an XOR-join.

Fig. 18 illustrates a more complex C-replacement of the IOR-join labeled j of Fig. 18(a) by the partial workflow graph containing the nodes w, x, y, and z and the edges f, g, h, and i in Fig. 18(b). This replacement is based on the cover $\mathcal{C} = \{\{a, c, e\}, \{b\}, \{d, e\}\}$.

Computing tests, including checking that edges are mutually exclusive can be done using state space exploration, which can take exponential time. More efficient heuristics exist for some special cases. For example, it is possible to



(a) A workflow graph.



(b) Local replacement of j.

Figure 18: An example of local replacement where the IOR-join j is replaced by the partial workflow graph composed of the nodes w, x, y, and z and the edges f, g, h, and i.

compute in quadratic time whether a set of edges in an acyclic process is mutually exclusive [23]. Efficient computation of the tests is out of scope of this paper.

We can now characterize the conditions under which an IOR-join has an equivalent local replacement:

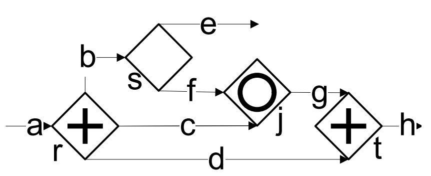
Theorem 4 (Equivalent local replacement existence)**.** *Let W be a sound workflow graph containing an* IOR*-join j.*

An IOR*-join j has an equivalent local replacement iff there exists a cover of $\circ j$ with respect to the outgoing edge of j. (See Appendix B for a proof.)*

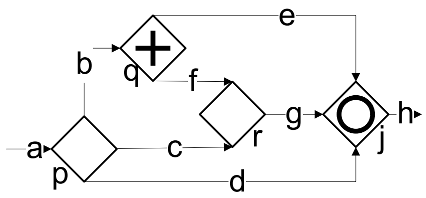
While Thm. 4 applies to any local replacement technique, the proof of the 'if' direction shows that, whenever there exists a cover of $\circ j$ with respect to the outgoing edge of j, the C-replacement of j produces an equivalent workflow graph.

4.3.3. The limitation of local replacements

The local replacement replaces an IOR-join by gateways which have local semantics and, as the replacement retains the same incoming edges as the IOR-join, only receives local "information". In some cases, this local information is not enough to differentiate between some reachable markings that can be differentiated by the non-local semantics of the IOR-join. The two workflow



(a) A local replacement of j cannot differentiate the markings $[b, c]$ and $[e, c]$.



(b) A local replacement of j cannot differentiate the markings $[e, g]$ and $[g]$.

Figure 19: Two workflow graphs containing an IOR-join that cannot be replaced locally

graphs illustrated by Fig. 19 are workflow graphs that cannot be locally replaced for this reason. The 'only if' direction of Thm. 4 allows us to capture this limitation.

Fig. 19(a) is a slight variation of Fig. 16(a) where changing the target of the edge e makes it impossible replace the IOR-join locally. By changing the target of e, e does not belong to $\circ j$ anymore. As in Fig. 16(a), the marking $m_1 = [b, c]$ does not enable j because there is a path from the token on b to the empty slot f and the marking $m_2 = [e, c]$ enables j. In contrast to Fig. 16(a), in Fig. 19(a), the markings m_1 and m_2 cannot be differentiated based only on the local "information", i.e., based on the marking of the edges f and c. In terms of Thm. 4, the test $T_1 = \{e, f\}$ cannot be used to build a cover of $\circ j$ anymore and there is no other test of g that contains f.

In Fig. 19(b), a local replacement would not be able the to differentiate the two reachable marking $[e, g]$ and $[g]$. In terms of Thm. 4, the IOR-join cannot be replaced locally because e does not belong to any test of h contained in $\circ j$, i.e., there exists no cover of h.

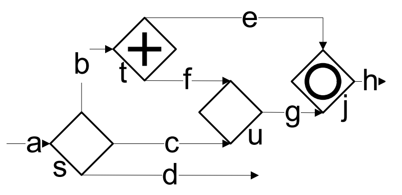
We will see in the next section how the IOR-joins of these two examples can be replaced using a non-local replacement which allows us to transfer the "missing information".

4.4. Non-local replacements for IOR-joins

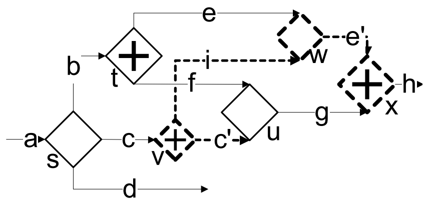
Fig. 20(b) shows an example of a non-local replacement where the IOR-join j of Fig. 20(a) is replaced by the partial workflow graph composed of the nodes w, x and the edges i, e' where, additionally, the AND-split v was inserted on the edge c which delivers additional (non-local) information to the IOR-join replacement via the edge i.

So, in addition to a local replacement, we allow non-local replacements to insert additional AND-splits in the graph, which can only be connected to the IOR-join replacement. These AND-splits only "copy" tokens to route them to the replacement and do not alter the original behavior of the process. This also preserves the graph structure to a large extent.

Kiepuszewski et al. [24, Proof of Theorem 5.1] give a completion approach to transform a Petri net with multiple sinks into a Petri net with a single sink. In this section, we will show that one can use a variation of that approach, which



(a) A workflow graph.



(b) Non-local replacement of j in Fig. 20(a).

Figure 20: An example of non-local replacement.

we call *K-replacement*, to replace an IOR-join in an acyclic workflow graph. We give a condition that characterizes the IOR-joins for which this replacement produces an equivalent workflow graph. Finally, we show that checking whether replacing the IOR-join produces an equivalent workflow graph can be done in polynomial time and that the replacement itself requires polynomial time.

4.4.1. Non-local replacement and equivalence:

First, we formalize the concept of non-local replacement. When *inserting* a gateway g on an edge e, we create an additional edge e' such that the source of e' is g and the target of e' is the target of e and the target of e becomes g. We say that the edge e' is the *resulting edge* from the insertion of g on e.

Definition 7 (Non-local replacement)**.** Let $W = (N, E, c, l)$ be a workflow graph and j be a node in N. Let $R = (N'', E'', c'', l'')$ be a partial workflow graph such that for each node $n \in N''$, $l''(n) = \text{XOR}$ or $l''(n) = \text{AND}$, $N \cap N'' = \emptyset$, and $E \cap E'' = \emptyset$. Let $l_g =< a_0, \ldots, a_n >$ be a list of AND-splits such that $l_g \cap (N \cup R) = \emptyset$.

A *non-local replacement* of j in W *by* R and l_g results in a workflow graph $W' = (N', E', c', l')$ obtained by the *insertion* of l_g on some edges $< e_0, \ldots, e_n >$ of W resulting in a list l_e of edges and the *insertion* of R such that:

- $N' = N \setminus \{j\} \cup N'' \cup l_g$,
- $E' = E \cup E'' \cup l_e$,
- $c'(e) = c(e) = (s, t)$ when $e \in E$, $s \in N$, $t \in N$, and $s \neq j \neq t$,

  $c'(e) = c''(e) = (s, t)$ when $e \in E''$, $s \in N''$, and $t \in N''$,

  $c'(e) = (s, t)$ such that $s \in N$ and $t \in N''$ if $e \in E$ and $c(e) = (s, j)$ or $e \in E''$ and $s \in l_g$,

  $c'(e) = (s, t)$ such that $s \in N''$ and $t \in N$ if $e \in E$ and $c(e) = (j, t)$,
- $l'(n) = l(n)$ when $n \in N \setminus j$, $l'(n) = l''(n)$ when $n \in N''$, and
- each element $x \in N'' \cup E''$ is on a path in W from an edge $e_{in} \in E$ such that $c(e_{in}) = (s, j)$ or a node $a_i \in l_g$ to the edge $e_{out} \in E$ such that $c(e_{out}) = (j, t)$.

We now define when a non-local replacement is semantically correct through a notion of equivalence of the two workflow graphs. Let, in the rest of this section, $W = (N, E, c, l)$ be a workflow graph containing an IOR-join j and $W' = (N', E', c', l')$ be a workflow graph obtained by non-local replacement of j. To define equivalence we need to map the markings of W and W'. We first define a mapping $\psi : E' \to E \cup \{null\}$ such that for any edge e' in E':

$$\psi(e') = \begin{cases} e' & \text{if } e' \in E, \\ e & \text{if } e' \text{ is the resulting edge of the insertion of an AND-join on } e, \\ null & \text{otherwise.} \end{cases}$$

We define a mapping ϕ from a marking of W' to a marking of W such that $\phi(m)[e] = \sum\limits_{\psi(e')=e} m[e']$.

We reuse the notion of *replacement execution sequence* defined in Sect. 4.3.

Definition 8 (Equivalence of non-local replacement)**.** W and W' are *equivalent* if for any pair of reachable markings m_1 of W, m'_1 of W' such that $m_1 = \phi(m'_1)$, we have:

1. for any transition $t = (E_1, j, E_2)$ and any marking m_2 such that $m_1 \xrightarrow{t} m_2$ in W, there exists a replacement execution sequence σ and a marking m'_2 such that $m'_1 \xrightarrow{\sigma} m'_2$ in W' and $m_2 = \phi(m'_2)$, and

2. for any replacement execution sequence σ and any marking m'_2 such that $m'_1 \xrightarrow{\sigma} m'_2$ in W', there exists a transition $t = (E_1, j, E_2)$ and a marking m_2 such that $m_1 \xrightarrow{t} m_2$ in W and $m_2 = \phi(m'_2)$.

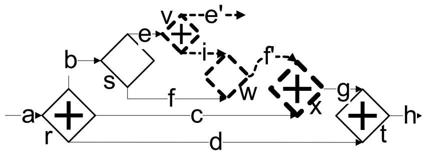
4.4.2. A simple non-local replacement:

As an intermediary step, we discuss informally a simple version of a local replacement technique, which we call *simple K-replacement.* We will then point out a shortcoming of simple K-replacement and modify it to obtain the *K-replacement.*

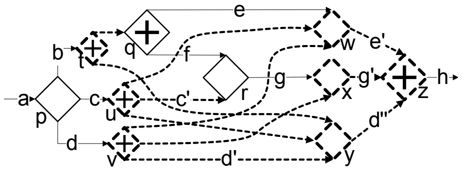
Fig. 21(a) and Fig. 21(b) show equivalent non-local replacements for the workflow graphs in Fig. 19(a) and Fig. 19(b), respectively. These are two examples of simple K-replacement.

The simple K-replacement uses the notion of a *bridge*: A *bridge* from an edge e to an edge e' is a path from e to e' such that each split on the path is an AND-split and each join on the path is an XOR-join. For example, in Fig. 21(a), the path $< e, v, j, w, f' >$ is a bridge from e to f'. The existence of a bridge from e to e' implies that every execution which marks e also marks e'.

The key idea of simple K-replacement is to replace an IOR-join by an AND-join and to ensure that every incoming edge of the new AND-join carries a token in every execution by adding suitable bridges. More precisely, for each incoming edge e' of the AND-join and each outgoing edge e of any XOR-split such that e is not on a path to e', we create a bridge from e to e'. Intuitively, for each



(a) Non-local replacement of j in Fig. 19(a).



(b) Non-local replacement of j in Fig. 19(b).

Figure 21: Non-local replacements of the IOR-joins in Fig. 19 that do not have a local replacement

outgoing edge e of an XOR-split that removes a token from a path to e', we add a bridge that brings an additional token to e' on a different path. This leads to equivalent workflow graphs for the examples in Fig. 19(a) and Fig. 19(b).

However, consider the workflow graph in Fig. 22 obtained by the same technique for the IOR-join v in the workflow graph shown by Fig. 20(a). When an execution σ marks the edge d of the workflow graph in Fig. 22, the edge h is also marked by σ which is not the case when an execution marks the edge d in the workflow graph in Fig. 20(a). Thus, simple K-replacement does not lead to an equivalent workflow graph when applied to an IOR-join that is not executed by every execution of the original workflow graph.

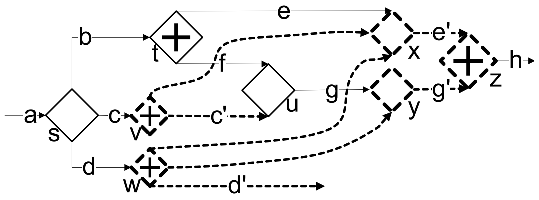


Figure 22: Non-local replacement of j in Fig. 20(a).

4.4.3. K-replacement:

We now describe our generalized technique, called *K-replacement*. Applying K-replacement to f in Fig. 20(a) results in the workflow graph in Fig. 20(b). K-replacement uses the notion of *dominator frontier* to apply the same replacement strategy as the simple K-replacement to a sub-graph of the workflow instead of the complete graph. Applying the K-replacement to a sub-graph of the workflow graph implies that the AND-join replacing the IOR-join only executes during the execution that marks an edge of this sub-graph and thus allows us to produce an equivalent replacement in more cases than with simple K-replacement.

To this end, we use the notions of dominator and dominator frontier.

Definition 9 (Dominator and dominator frontier)**.** A node x_1 *dominates* another node x_2 if each path from the source edge of the workflow graph to x_2 contains x_1.

A dominator x_1 of a node x_2 is the *minimal dominator* of x_2 if there exists no node x'_1 such that x'_1 dominates x_2, x_1 dominates x'_1, and $x_1 \neq x'_1 \neq x_2$.

A set E_d of edges is the *dominator frontier* of a node x_2 if a node x_1 is the minimal dominator of x_2, $E_d \subseteq x_1\circ$, for each edge $e \in E_d$, $e < x_2$, and for each edge $e' \in ((x_1\circ) \setminus E_d)$, we have $e' \not< x_2$.

For example, in Fig. 16(b), the nodes r and s dominate the node v and the node s is the minimal dominator of v. In Fig. 20(a), the dominator frontier of j is the set of edges $\{b, c\}$.

K-replacement replaces an IOR-join j by an AND-join. Furthermore, a bridge from e to e' is created for each incoming edge e' of j and each edge e such that e is the outgoing edge of an XOR-split on a path from an edge of the dominator frontier of j to j, and there is no path from e to e'. K-replacement is detailed further by Algorithm 1. As mentioned earlier, Fig. 20(b) shows the workflow graph resulting from the application of Algorithm 1 to the IOR-join j in Fig. 20(a).

Algorithm 1 K-replace(j, W).
Input: A workflow graph $W = (N, E, c, l)$ and and IOR-join $j \in N$.
Output: A workflow graph $W' = (N', E', c', l')$ where j has been K-replaced.

```
Create an AND-join $a$ and set the source of the outgoing edge of $j$ to be $a$.
Let $E_I$ be the set of incoming edges of $j$ in $W$.
for each edge $e \in E_i$ do
    Create an XOR-join $x_e$.
    Set the target of $e$ to be $x_e$.
    Create and edge from $x_e$ to $a$.
end for
Let $preset(j)$ be the set of all elements in $E \cup N$ from the minimal dominator
of $j$ having a path to $j$
for each edge $e' \in \circ j$ do
    for each decision $d \in preset(j)$ do
        Let $preset(e')$ be the set of all elements in $E \cup N$ from the dominator
frontier of $j$ having a path to $e'$,
        for each edge $e \in d\circ$ such that $e \notin preset(e')$) do
            if $d \neq$ minimal dominator of $j$ OR $e \in$ dominator frontier of $j$ then
                if The target of $e$ is not an and split then
                    Insert an AND-split $s$ on $e$
                else
                    Let $s$ be the target of $e$
                end if
                Add an edge from $s$ to $x_{e'}$
            end if
        end for
    end for
end for
```

The K-replacement implies that the AND-join replacing the IOR-join executes in every execution where an edge of the dominator frontier is marked. Thus, intuitively, the K-replacement produces an equivalent workflow graph if each execution that marks one edge of the dominator frontier also executes the IOR-join.

In the following, we formalize this intuition as a necessary and sufficient condition for the K-replacement to produce an equivalent workflow graph.

Theorem 5 (K-replacement applicability). *K-replacement of an* IOR*-join j in a workflow graph W produces a workflow graph that is equivalent with W iff j is executed in each execution of W in which an edge of the dominator frontier of j is marked. (See Appendix C for a proof.)*

Thus, K-replacement of the IOR-join j in Fig. 20(a) produces an equivalent workflow graph shown in Fig. 20(b) because j executes in an execution σ if and only if an edge of its dominator frontier $\{b, c\}$ is marked by σ. This is not the case for the workflow graph in Fig. 23. The dominator frontier of y in Fig. 23 is the set $\{b, c\}$. The edges b and c are marked by every execution. Thus, applying the K-replacement algorithm to y would thus produce a workflow graph in which the task z is executed during every execution. It is not the case for the workflow graph in Fig. 23 in which the execution where the edges g and d are marked does not execute z.

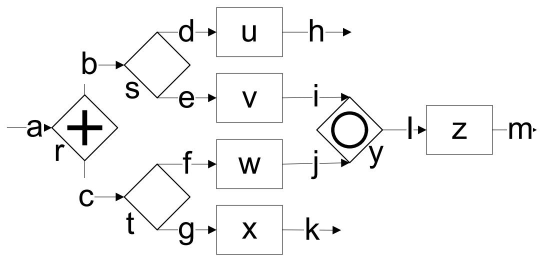


Figure 23: The IOR-join y cannot be replaced.

4.4.4. A note on complexity

We now argue that the condition expressed by Thm. 5 can be computed efficiently, i.e., in polynomial time with respect to the size of the workflow graph. Algorithm 1 also runs in polynomial time. This allows us to conclude that we can identify IOR-joins that can be replaced by K-replacement and replace them efficiently:

Theorem 6 (Polynomial time complexity of K-replacement). *Let j be an IOR-join, and W be the workflow graph containing j.*

1. *Computing whether the K-replacement of j produces a workflow graph that is equivalent to W can be done in polynomial time with respect to the size of W.*

2. *The K-replacement of an* IOR*-join j can be computed in polynomial time with respect to the size of W.*

Proof. We show the polynomial complexity of the check and of the replacement separately:

1. In previous work [23], we have shown how to perform a *symbolic execution* of an acyclic workflow graph. The symbolic execution assigns to each edge of the workflow graph a symbol. These symbols allow us to check whether two edges e_1, e_2 of the workflow graph are marked by the exact same set of executions by computing whether the symbol assigned to e_1 and e_2 satisfy a certain notion of equivalence.

   To check the condition of applicability ok the K-replacement of an IOR-join j with denominator frontier D_f, we must check that, the outgoing edge of j and at the dominator frontier of j are marked by the same set of executions. We achieve this by checking that the symbol assigned to the outgoing edge of j is equivalent to the symbol obtained by summing the symbols assigned to the edges of D_f. The symbolic execution, summing the symbols assigned to the edges in D_f, and computing the equivalence, requires a quadratic amount of time with respect to the size of the workflow graph.

2. The domination relationship can be expressed using a tree data structure. Computing the dominator tree of W can be done in $O(|E|log(|N|))$ [25]. Checking if an outgoing edge of the minimal dominator is part of the dominator frontier is a simple reachability test which requires linear time. Applying Algorithm 1 clearly requires a polynomial amount of time.

□

4.5. The difficulty of replacing an IOR-join

As discussed above, the IOR-join y in Fig. 23 cannot be replaced correctly with K-replacement. In this section, we provide an argument why it is difficult to implement the IOR-join in Fig. 23 with any combination of AND and XOR gateways.

Recalling the discussion in Sect. 1 we have to specify an equivalence and we require some structure to be preserved in order to rule out some 'simple' implementations that incur an exponential blowup.

In this section, we take the view that the IOR-join synchronizes its incoming branches, where these incoming branches have a certain 'forking' logic, depending on the gateway structure 'before' the IOR-join. The forking logic for the example in Fig. 23 is represented by the workflow graph in Fig. 24. We will show that the workflow graph in Fig. 24 cannot be completed with any combination of AND and XOR gateways to produce a behavior equivalent to the behavior of the workflow graph in Fig. 23. In this sense, no combination of AND and XOR gateways can produce the synchronization behavior of the IOR-join in Fig. 23.

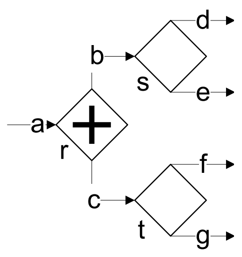


Figure 24: Prefix that cannot be completed.

Rather than picking a concrete behavioral equivalence (cf. discussion in Sect. 1), we formalize properties that a workflow graph must have to be equivalent to the workflow graph in Fig. 23. We allow multiple tasks of the implementing workflow graph to be labeled with z and therefore to correspond to the task z in Fig. 23.

Definition 10 (Equivalent workflow graph properties)**.** Let W' be a workflow graph which has the prefix illustrated by Fig. 24. Let $t_1 = (\{b\}, s, \{d\})$ and $t_2 = (\{c\}, t, \{g\}$. The workflow graph W' satisfies the following properties:

P1 There exists no execution where t_1, t_2, and a task labeled with z are executed.

P2 There exists an execution during which t_1 and a transition t_z executing a task labeled with z are executed and, for each execution where t_1 and t_z are executed, t_1 is executed before t_z.

It is easy to see that the workflow graph in Fig. 23 satisfies these properties and that notions of equivalence such as the ones that we defined for local and non-local replacements would ensure that any equivalent workflow graph satisfies them too.

We now present two lemmas that will be useful to prove the workflow graph in Fig. 24 cannot be completed:

Lemma 7. *Let W be a deadlock-free workflow graph, e, e' be two edges of W, m be a reachable marking of W which marks e.*

If there exists a path p from e to e' in W, then there exists a marking m', reachable from m, such that m' marks e'. (Can be proved by a straightforward induction on p. Moreover, the lemma presented in Appendix A.1 implies Lemma 7)

Lemma 8. *Let W be a deadlock-free acyclic workflow which does not contain IOR-joins. Let $t = (E_1, n, E_2)$ and $t' = (E'_1, n', E'_2)$ be two transitions of W.*

If, in each execution σ of W where t and t' occur, we have that t occurs before t' during σ, then there exists a path from an edge $e \in E_2$ to an edge $e' \in E'_1$. (See Appendix A.2 for a proof.)

We can now enunciate our result:

Theorem 9 (The synchronization role of IOR-joins cannot be implemented using only AND and XOR-logic)**.** *There exists no deadlock-free workflow graph W' such that W' has the workflow graph P illustrated by Fig. 24 as prefix, W' does not contain any IOR-join, and W' satisfies the properties of Def. 10.*

Proof. We prove this theorem by contradiction: Suppose that there exists a workflow graph W' such that W' has P (illustrated by Fig. 24) as prefix, does not contain an IOR-join, and satisfies P1 and P2.

By P2, we have that t_1 and a transition $t_z = (E_1, n, E_2)$ such that n is a task labeled z occur together in some execution and that t_1 always occur before t_z when both occur. By Lemma 8, there exists a path p from d to the incoming edge of n.

Consider an execution sequence $\sigma = \langle [a], (\{a\}, r, \{b, c\}), [b, c], t_1, [d, c], t_2, [d, g] \rangle$. We can complete σ to obtain the execution σ^* by following p and thus executing t_z (Lemma 7). This contradicts P1 because t_1, t_2, and a task labeled z are executed during σ^*. $\square$

4.6. On the translation to (non-free choice) Petri nets

We have shown that, in some sense, the IOR-join cannot always be replaced by a combination of AND and XOR gateways. We have shown in Sect. 3 that workflow graphs without IOR gateways correspond to free-choice workflow nets. Hence, there is no free-choice workflow net implementing the IOR-join in Fig. 23 in the sense discussed above.

To translate an acyclic workflow graph into a non-free-choice Petri-net, one can use *dead path elimination* [19, 20] to implement the IOR-join with gateways that have a local semantics. Dead path elimination uses workflow graphs with individual tokens, where a token can have a value that is either *true* or *false*. Such a workflow graph can be easily translated into a high-level Petri net, which in turn can be unfolded into a non-free-choice Petri net.

A more direct approach to implement the IOR-join from Fig. 23 as a Petri net is shown in Fig. 25, where the IOR-join replacement is delimited by the dashed box. The behavior of this Petri net is equivalent to the workflow graph in Fig. 23. Note that, similar to K-replacement, we provide additional inputs to the IOR-replacement and that this construction preserves most of the structure of the original workflow graph. These additional inputs give the IOR-join replacement information on the edges that have been marked by the execution.

We can then think of the IOR-join as two boolean expressions over these marked edges that fully characterize its executions: the first expression characterizes the executions that lead to a token on the outgoing edge of the IOR-join, the second characterizing all other executions. Both expressions can be easily represented by a non-free-choice Petri net as in the example in Fig. 25, where the transitions y_1, y_2, and y_3 implement the first expression $(d' \wedge j) \vee (i \wedge j) \vee (i \wedge g')$ The transition y^* implements the second expression. The role of y^* is to "purge" the place d' and g' in the second case.

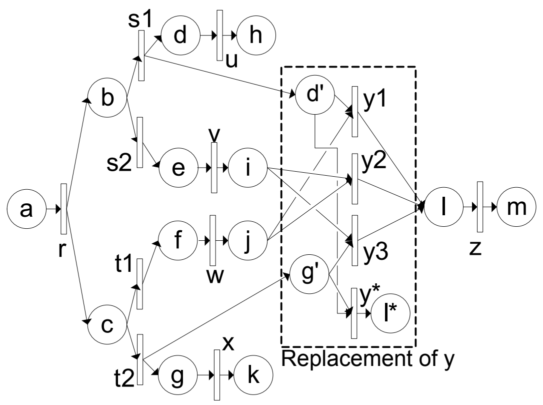


Figure 25: A non-free-choice Petri net that is equivalent to the one in Fig. 23.

This construction can be defined to implement an IOR-join in any acyclic workflow graph because the full execution history can be provided to the replacement subgraph. However, the Petri net representations of the boolean expressions are exponential in the size of the workflow graph. We leave it as future work to evaluate whether this exponential blowup can be mitigated using simplification techniques for boolean formulas.

To sum up, Thm. 9 gives a strong argument why IOR-joins cannot be easily implemented by a free-choice workflow net, it points to some difficulty when trying to translate to general workflow nets.

4.7. Related work

To our knowledge, no existing general translation satisfies our three requirements of equivalence, efficiency, and structure preservation simultaneously. But translations for subsets of workflow graphs or translations satisfying a subset of the requirements exist. In the following, we discuss the qualities of these translations and their shortcomings.

Wynn et al. [22] use a full state space exploration to find each IOR-join that has only mutually exclusive incoming edges, and thus can be replaced by an XOR-join. Likewise, an IOR-join that has only pairwise always-concurrent edges is replaced by an AND-join. This requires an exponential time and only works for the rare IOR-joins whose incoming edges meet one of these two conditions. Note that these conditions can also be computed in quadratic time for acyclic workflow graphs [23].

Vanhatalo et al. [21] use the *refined process structure tree* to decompose a workflow graph. The decomposition provides local replacement heuristics for some, but not all, IOR-joins. Although this approach is incomplete, the decomposition and heuristics can be computed in linear time and, because they result in local replacements, preserve the structure of the original model.

Mendling et al. [26] use the theory of regions to synthesize a Petri net from the reachability graph of the process model. The resulting net loses the structure of the original process and its size grows exponentially with respect to the size of the original process. While the paper does not prove any notion of equivalence that is preserved, it further points to the work of Cortadella et al. [27], which guarantees that the synthesized net is bisimilar to the process model reachability graph.

As mentioned in the previous section, one could use the dead path elimination mechanism of BPEL [19, 20] to implement IOR-join in acyclic processes. There are two approaches in the literature to translating dead path elimination to high-level Petri nets [28, 29]. A high-level Petri-net can be further translated into a low-level Petri net by a well-known unfolding construction, which in general incurs an exponential blowup.

Ouyang et al. [30] also give a translation from BPEL processes to Petri nets. In contrast to the work of Lohmann [29] and Stahl [28], they model dead path elimination directly using low-level nets. The evaluation of an IOR-join is modeled using a Petri-net encoding a boolean function similarly to the idea we presented in Sect. 4.6.

In all three approaches, the resulting Petri net is exponentially larger than the original process model. It can be argued that all three approaches fully preserve the behavior of the original process model and preserve its structure to a large extent. Note that these three translations in general result in a non-free-choice Petri net.

5. Conclusion

In the first part of this paper, we showed that a workflow graph without inclusive logic and a free-choice workflow net can be considered to be the same object. This gave rise to a general translation from free-choice workflow nets to workflow graphs.

In the second part of this paper, we gave new results on the translation from acyclic workflow graphs with inclusive logic to free-choice workflow nets. We characterized when a fully local replacement of an IOR-join exists and we presented a more general, non-local replacement technique that can be executed in polynomial time.

While these results have been presented for the replacement of a single IOR-join in a sound acyclic workflow graph, they can be applied to replace multiple IOR-joins in an acyclic workflow graph. Moreover, it can be shown that both, the local and the non-local, replacement techniques do not alter the soundness of the workflow graph, i.e., they cannot introduce or fix a control-flow error. This makes these replacement techniques applicable to replace IOR-joins in workflow graphs of which the soundness is unknown, for example, when performing a control-flow analysis. We have used elsewhere [5] *process structure trees* to decompose the workflow graph into fragments. Such a decomposition allows us to factor out cycles and therefore to apply the replacements also in many workflow graphs containing cycles.

Furthermore, we have shown that the synchronization provided by the IOR-join cannot, in general, be implemented by free-choice constructs. Translations of a workflow graph containing an IOR-join into a non-free-choice Petri nets exist, however, known translations have an exponential blowup and, usually, do not preserve the structure of the original process. These results show a difficulty to translate the non-local semantics of the IOR-join into a modeling language that only contains local gateways and therefore a difficulty to fully leverage Petri net based techniques for process models containing IOR-joins. For example, verification techniques for free-choice Petri nets, of which some run in polynomial time, may not be applicable to all workflow graphs with IOR-joins. For them, less efficient verification techniques for general Petri nets have to be used, which will additionally suffer from the exponential blowup created by using a fallback translation from IOR-joins to Petri nets.

Acknowledgement. We would like to thank the anonymous reviewers for their helpful comments. In particular, we thank the third reviewer who suggested the final translation for the IOR-split. Furthermore, the third author would like to thank Jussi Vanhatalo for fruitful discussions about the subject of this paper.

References

[1] W. van der Aalst, Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques, in: Business Process Management: Models, Techniques, and Empirical Studies, Vol. 1806 of LNCS, Springer-Verlag, 2000, pp. 161–183.

[2] W. van der Aalst, A. Hirnschall, H. Verbeek, An Alternative Way to Analyze Workflow Graphs, in: CAiSE02, volume 2348 of LNCS, 2002, pp. 535–552.

[3] J. Esparza, Decidability and complexity of petri net problems - an introduction, in: W. Reisig, G. Rozenberg (Eds.), Petri Nets, Vol. 1491 of Lecture Notes in Computer Science, Springer, 1996, pp. 374–428.

[4] J. Desel, J. Esparza, Free choice Petri nets, Cambridge University Press, New York, NY, USA, 1995.

[5] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Analysis on demand: Instantaneous soundness checking of industrial business process models, Data Knowl. Eng. 70 (5) (2011) 448–466.

[6] R. J. van Glabbeek, The linear time-branching time spectrum (extended abstract), in: CONCUR, Vol. 458 of LNCS, Springer, 1990, pp. 278–297.

[7] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, Distributed and parallel databases 14 (1) (2003) 5–51.

[8] W. M. P. van der Aalst, A. Hirnschall, H. M. W. E. Verbeek, An alternative way to analyze workflow graphs, in: CAiSE 2002, Vol. 2348 of Lecture Notes in Computer Science, Springer, 2002, pp. 535–552.

[9] T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580. doi:http://dx.doi.org/10.1109/5.24143. URL `http://dx.doi.org/10.1109/5.24143`

[10] J. Dehnert, P. Rittgen, Relaxed soundness of business processes, in: K. R. Dittrich, A. Geppert, M. C. Norrie (Eds.), CAiSE, Vol. 2068 of Lecture Notes in Computer Science, Springer, 2001, pp. 157–170.

[11] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in bpmn, Information & Software Technology 50 (12) (2008) 1281–1294.

[12] H. Störrle, J. H. Hausmann, Towards a formal semantics of uml 2.0 activities, in: P. Liggesmeyer, K. Pohl, M. Goedicke (Eds.), Software Engineering, Vol. 64 of LNI, GI, 2005, pp. 117–128.

[13] E. Kindler, On the semantics of epcs: Resolving the vicious circle, Data Knowl. Eng. 56 (1) (2006) 23–40.

[14] S. J. Leemans, D. Fahland, W. M. van der Aalst, Discovering block-structured process models from event logs - a constructive approach, in: S. Rinderle-Ma, F. Toumani, K. Wolf (Eds.), Petri Nets 2013, Vol. 6896 of Lecture Notes in Computer Science, Springer, 2013, pp. 148–165.

[15] J. van der Werf, B. van Dongen, C. Hurkens, A. Serebrenik, Process Discovery using Integer Linear Programming, Fundamenta Informaticae 94 (2010) 387–412.

[16] D. Fahland, W. M. P. van der Aalst, Repairing process models to reflect reality, in: BPM'12, Vol. 7481 of Lecture Notes in Computer Science, Springer, 2012, pp. 229–245.

[17] J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through sese decomposition, in: B. J. Krämer, K.-J. Lin, P. Narasimhan (Eds.), ICSOC, Vol. 4749 of Lecture Notes in Computer Science, Springer, 2007, pp. 43–55.

[18] A. Polyvyanyy, J. Vanhatalo, H. Völzer, Simplified computation and generalization of the refined process structure tree, in: M. Bravetti, T. Bultan (Eds.), WS-FM, Vol. 6551 of Lecture Notes in Computer Science, Springer, 2010, pp. 25–41.

[19] H. Völzer, A New Semantics for the Inclusive Converging Gateway in Safe Processes, in: BPM, Vol. 6336 of LNCS, Springer, 2010, pp. 294–309.

[20] A. Alves, et al., Web services business process execution language version 2.0, OASIS Standard 11.

[21] J. Vanhatalo, H. Völzer, F. Leymann, S. Moser, Automatic Workflow Graph Refactoring and Completion, in: ICSOC, Vol. 5364 of LNCS, 2008, pp. 100–115.

[22] M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, D. Edmond, Business Process Verification–Finally a Reality!, Business Process Management Journal 15 (1) (2009) 74–92.

[23] C. Favre, H. Völzer, Symbolic Execution of Acyclic Workflow Graphs, in: BPM, Vol. 6336 of LNCS, Springer, 2010, pp. 260–275.

[24] B. Kiepuszewski, A. ter Hofstede, W. van der Aalst, Fundamentals of control flow in workflows, Acta Informatica 39 (3) (2003) 143–209.

[25] T. Lengauer, R. Tarjan, A fast algorithm for finding dominators in a flowgraph, ACM Transactions on Programming Languages and Systems 1 (1) (1979) 121–141.

[26] J. Mendling, B. van Dongen, W. van der Aalst, Getting rid of or-joins and multiple start events in business process models, Enterprise Information Systems 2 (4) (2008) 403–419.

[27] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving petri nets for finite transition systems, IEEE Trans. Computers 47 (8) (1998) 859–882.

[28] C. Stahl, A petri net semantics for bpel, Tech. rep., Humboldt-Universität zu Berlin (2005).

[29] N. Lohmann, A Feature-Complete Petri Net Semantics for WS-BPEL 2.0, in: WS-FM, Vol. 4937 of LNCS, Springer-Verlag, 2008, pp. 77–91.

[30] C. Ouyang, E. Verbeek, W. Van Der Aalst, S. Breutel, M. Dumas, A. Ter Hofstede, Formal semantics and analysis of control flow in ws-bpel, Science of Computer Programming 67 (2) (2007) 162–198.

Appendix — Proofs

We include as appendix, the proofs related to the translation of IOR-joins which would disturb, due their size, the reading flow if they where inlined to the paper.

Appendix A. Lemmas

We first prove two lemmas:

Appendix A.1.

Lemma 10. *Let W be a deadlock-free workflow graph. Let e, e^*, e' be three edges of W such that there exists no path from e^* to e'. Let m be a reachable marking of W which marks e and e^*.*

If there exists a path p from e to e' in W, then there exists a marking m', reachable from m, such that m' marks e' and e^*.

Proof. We proceed by structural induction on p.

Base case: $e = e'$. The marking m marks e^* and e'.

Induction step: Let e_n be n-th edge of p, e_{n+1} be the next edge, and n be the node between e_n and e_{n+1}. By induction assumption there exists a making m_n, reachable from m, such that m_n marks e_n and e^*. By the workflow graph semantics, if n is an XOR-join, an XOR-split, an AND-split, or an IOR-split, there exists a transition t such that $m_n \xrightarrow{t} m_{n+1}$ and m_{n+1} marks e_* and e_{n+1}. We are left with the cases where n is an IOR-join or an AND-join. As W is deadlock-free, there exist a marking m_{n+1}, reachable from m_n, in which e_n is not marked, i.e., n was executed. By the workflow graph semantics, executing an IOR-join or an AND-join marks its single outgoing edge e_{n+1}. Reaching m_{n+1} from m_n does not require to execute the target of e^*, i.e., consume the token on e^*, otherwise there would exist a path from e^* to n and thus a path from e^* to e' would exist.

□

Appendix A.2. Proof of Lemma 8

Let W be a deadlock-free acyclic workflow which does not contain IOR-joins. Let $t = (E_1, n, E_2)$ and $t' = (E'_1, n', E'_2)$ be two transitions of W.

We prove that if in each execution σ of W where t and t' occur, we have that t occurs before t' during σ, then there exists a path from an edge in E_2 to an edge in E'_1.

We proceed by contradiction: suppose in each execution σ of W where t and t' occur we have that t occurs before t' during σ, that there is no path between any edge in E_2 and any edge in E'_1. We show a contradiction by showing the existence of an execution where t' executes before t. We consider the case where

n and n' have a single entry and a single exit edge, a similar reasoning can be applied for the other cases.

Let σ be an execution in which t and t' are executed. Let m be the last marking before executing t in σ. The marking m marks the incoming edge e^* of n. Let σ' be the prefix of σ until reaching the marking m. Because t' executes in σ, there exists an edge e, that carries a token in m, and there exists a path p to the outgoing edge e' of n'. By Lemma 10, there exist an execution sequence σ'' from m to a marking m'' such that m'' marks e^* and e', i.e, t' was executed during σ'' and t was not. As e^* is marked by m'', the transition t is enabled. Therefore there exists an execution in which t' occurs before t.

Appendix B. Proof of Thm. 4

Let W be a deadlock-free workflow graph containing an IOR-join j. We show that an IOR-join j has an equivalent local replacement iff there exists a cover of $\circ j$ with respect to j°.

We show the two directions of the theorem separately:

$\Leftarrow$ Let C be a cover of $\circ j$ with respect to the outgoing edge of j. We show that there exists a local replacement by showing that the C-replacement R of j results in a workflow graph W' that is equivalent to W. We use the usual labeling for the nodes of R as illustrated by Fig. 17. Let m_1, m_2 be two markings of W (and thus two markings of W' because $E \subseteq E'$):

1. For any transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W, there exists a replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W':

   By the IOR-join semantics, we have $m_2 = m_1 - E_1 + e_o$ and $E_1 \neq \emptyset$. Note that, because j is enabled in m_1 and W is sound, there is no token upstream of an edge in $\circ j$. We now build the execution σ of R that changes m_1 to m_i by executing the AND-splits a_e that have a marked incoming edge and then the XOR-joins of R. By the test definition and the definition of C-replacement, each XOR-join has one marked incoming edge and, by the the property of mutual exclusion of the edges in a test, only one of incoming edge is marked. Then the final and join f of R is executed. The AND-join f is enabled in m_i because each XOR-join was executed and thus, by C-replacement definition and XOR-join semantics, each incoming edge of f carries a token. Executing f changes m_i into m_2.

2. For any replacement execution sequence σ such that $m_1 \xrightarrow{\sigma} m_2$ in W', there exists a transition $t = (E_1, j, E_2)$ such that $m_1 \xrightarrow{t} m_2$ in W: We must show that:

   (a) t is enabled in m_1: Suppose that t is not enabled in m_1. By IOR-join semantics either no incoming edge of j is marked in m_1 or there is a marked edge preceding a non-marked incoming edge of j. Because σ is a replacement transition sequence, all transitions of σ execute a node in R. Thus, because m_1 only marks edges in E and by the definition of local replacement, an edge of $\circ j$ is marked. Thus, there must be an edge e that precedes an edge $e' \in \circ j$ which does not carry a token in m_1. By the cover definition, e' belongs to a test T, and by C-replacement and the definition of replacement execution sequence, there exists an edge e'', such that $e'' \neq e'$, which belongs to T and is marked by m_1. As W is deadlock-free by assumption, there is a path from e to e', and e'' is not on a path to or from e', by Lemma 10, there exists an execution sequence σ' and a marking m'_1 such that $m_1 \xrightarrow{\sigma'} m'_1$ and the edges e', e'' carry a token in m'_1. This is in contradiction with the definition of a test as all the edges in a test are mutually exclusive.

(b) executing t results in m_2: like t, σ consumes one token of each incoming edge marked in m_1 because if an edge of $\circ j$ was marked in m_2, then a replacement transition would be enabled which would contradict the definition of replacement execution sequence. It is also clear that, like executing σ, executing t marks the (formerly) outgoing edge of j by the IOR-join semantics.

$\Rightarrow$ We show that when an IOR-join j in a workflow graph $W = (N, E, c, l)$ can be replaced locally by a partial workflow graph R (not necessarily using a C-replacement) to result in an equivalent workflow graph $W' = (N', E', c', l')$, then there exists a cover of $\circ j$ with respect to the outgoing edge e of j.

A set X of edges is an *independent set* iff for each pair of edge $e_1, e_2 \in X$ there exists no path from e_1 to e_2. An independent set X_1 is *maximal with respect to* a graph G iff there exist no independent set X_2 of edge in G such that $X_1 \subset X_2$. Let $G = R \cup \circ j \cup \{e\}$. In the following, whenever we mention a maximum independent set we omit to precise that it is with respect to G. It is clear that $\{e\}$ and $\circ j$ are both maximum independent sets. Let δ be a function which for each ordered pair of maximum independent sets (X_1, X_2) returns the number of edges, not comprised in $X_1 \cup X_2$ on a path from an edge in X_1 to an edge in X_2.

We proceed by structural induction on the G, starting from e and following the edges backward. We show that at each step, given a maximum independent set X s.t. $X \neq \circ j$ and a cover C of X with respect to e, we can build a maximal independent set X' such that $\delta(\circ j, X') < \delta(\circ j, X)$ and a cover C' of X' with respect to e. Which shows that there is a cover C^* of $\circ j$ with respect to e in W'. By definition of equivalence of the local replacement, it is clear that a test $T \subseteq E$ of an edge $e \in E$ in W' is also a test of e in W. Thus C^* is a cover of $\circ j$ with respect to the outgoing edge of e in W.

Base case: $\{\{e\}\}$ is a cover of $\{e\}$ with respect to e.

Induction step: Assume C to be a cover of $X \subseteq G$ with respect to e such that X is a maximum independent set. We show that for each edge u with the node n as target such that $n\circ \subseteq X$, there exists a cover C' of a set $X' \subseteq G$ with respect to $\{e\}$ such that $u \in X'$, X' is a maximum independent set, and $\delta(\circ j, X') < \delta(\circ j, X)$. We distinguish three cases:

1. n is an XOR-join: Without loss of generality, we can assume that n has two incoming edges u, v and one outgoing edge w. Let $X' = X \cup \{u, v\} \setminus \{w\}$. Because X is a maximum independent set, it is clear that X' is a maximum independent set such that $\delta(\circ j, X') < \delta(\circ j, X)$. We transform C into a cover C' of X' with respect to e by replacing w by u and v in all test of C.
   We first show that if a test $T \in C$ of e contains w, then $T' = T \setminus \{w\} \cup \{u, v\}$ is a test of e:
   We show that the edges in T' are pairwise mutually exclusive: Because T is a test, all edges in $T \setminus \{w\}$ are mutually exclusive. By the soundness assumption of W, the edges u and v are mutually exclusive. Moreover, the edges in $T \setminus \{w\}$ are pairwise mutually exclusive with u and v because otherwise they would not be mutually exclusive with w by the XOR-join semantics. Thus, all edges of T' are pairwise mutually exclusive.
   To prove that T' is a test, we are left to show that, for any execution σ, σ marks e iff σ marks an edge of T': from the construction of T' and the XOR-join semantics which ensures that σ marks w iff σ marks an edge in $\circ n$, we have that σ marks an edge in T' iff σ marks an edge in T. Because T is a test, it implies that σ marks T' iff σ marks e.
2. n is an XOR-split: Without loss of generality, we can assume that n has two outgoing edges v, w.
   We first show that a test $T \in C$ containing v must contain w (and vice et versa): Suppose that a test $T \in C$ contains v but not w. Let σ be an execution such that σ marks w. By the test definition, there exist an edge $x \in T$ that is marked by σ. Note that, by definition of C, we also have $x \in X$. Let σ' be a prefix of σ such that the last marking m_1 of σ' is followed in σ by m_2 and m_2 is the first marking in σ which marks w or x. We can assume, without loss of generality, that m_2 marks w and thus m_1 marks the incoming edge u of v. Because x is marked in a state following m_1 during σ, there exists an edge x' and a path p from x' to x such that m_1 marks x'. As X is a maximum independent set, there exists no path between x and w, which implies that there exists no path between u and x.ar By Lemma 10, there exists a marking m' reachable from m_1 such that m' marks u and x. The transition $t = (\{u\}, n, \{v\})$ is enabled in m' by XOR-split semantics and because m' marks u.

Executing t in m' results in a marking m'' such that m'' marks v and x. As there exists a marking which marks v and x, v and x are not mutually exclusive, so T is not test. We have shown that a test containing v must contain w. (The same reasoning can be applied to show that a test containing w must contain v.) Let $X' = X \cup \{u\} \setminus \{v, w\}$. Because X is a maximum independent set, it is clear that X' is a maximum independent set such that $\delta(\circ j, X') < \delta(\circ j, X)$. We transform C into a cover C' of X' with respect to e by replacing v and w by u in all test of C. Using a similar reasoning as done the previous case n is an XOR-join, it can be shown that if a test $T \in C$ of e contains v and w, then $T' = T \setminus \{v, w\} \cup \{u\}$ is a test of e.

3. $l(n) = AND$: Let $X' = X \cup \circ n \setminus j\circ$. Because X is a maximum independent set, it is clear that X' is a maximum independent set such that $\delta(\circ j, X') < \delta(\circ j, X)$. We transform C into a cover C' of X' with respect to e by as follows: For each test $T \in C$ such that T contains an edge $v \in j\circ$, we create a test T_u for each edge $u \in \circ j$ such that $T_u = T \cup \{u\} \setminus \{v\}$. We replace T in C by the tests $\displaystyle\bigcup_{u \in \circ j} T_u$ to obtain C'. It is easy to see that each T_u is a test of e and that C' is a cover of X' with respect to e.

Appendix C. Proof of Thm. 5

Let $W = (N, E, c, l)$ be a sound workflow graph and j an IOR-join in W.

We start by proving Lemma 11 which will be useful to prove the condition of applicability given by Thm. 5.

Lemma 11. *Let W' be the workflow graph obtained by K-replacment of j. Let F be the dominator frontier of j in W. Let e' be an edge in $\circ j$ in W. Let x be the XOR-join targeted by e' in W' and e'' be the outgoing edge of x.*

For any execution σ of W' that marks an edge e in F, there exists a path p in W' from e to e'' such that each edge of p is marked during σ.

Proof. We build p inductively showing that each edge on p is 1. marked during σ and 2. has a path to e'':

Base case: It is clear that e satisfies both 1 and 2.

Induction step: $e_1 \in \circ n$ is satisfies 1 and 2. We show that there exists an edge $e_2 \in n\circ$ that satisfies 1 and 2.

If n is an XOR-join, AND-split, or AND-join by induction hypothesis and workflow graphs semantics it is clear that e_2 exists and satisfies 1 and 2.

When n is an XOR-split, then, by the XOR-split semantics, there exists an edge e_2 that satisfies 1. We are left to show that e_2 satisfy 2. Suppose that e_2 is not on a path to e', then by the K-replacement procedure, the target of e_2 is an AND-split with a path to e''. □

We can now prove that j has an equivalent K-replacement iff j is executed in each execution where an edge of the dominator frontier of j carries a token.

We show the two directions of the theorem separately:

$\Rightarrow$ Assume that there exists some execution σ such that dominator of j is executed in σ but j is not executed in σ.

We show by contradiction that K-replacement does not lead to an equivalent workflow graph: Suppose that there exists a valid K-replacement of j resulting into a workflow graph W' such that $W \equiv W'$. Consider the first marking m_2 in σ such that there is no edge preceding j that is marked in m_2 and the previous marking m_1 in σ. It is clear that there exists an edge e_2 that is marked in m_2 and not in m_1 such that e_2 does not precede j. The edge e_1 preceding e_2 is marked in m_1 and precedes j. By the completion definition (Algorithm 1), in W', an AND-split a is inserted on e_2 to obtain W' and therefore there is a path from e_2 (which is the incoming edge of a) to the previously outgoing edge of j. There exists a marking m'_2 of W' such that $m_2 \equiv m'_2$ and e_2 is marked in m'_2. By Lemma 11 and because there is no deadlock, there exists an execution sequence in W' from m'_2 to a marking in which the (previously) outgoing edge of j carries a token. There exists no such execution sequence from m_2 in W.

$\Leftarrow$ Assume that j is executed in each execution where an edge of the dominator frontier F of j carries a token.

Let $W' = (N', E', c', l')$ be the workflow graph obtained by K-replacement of j. Let m_1 be a reachable marking of W and m'_1 be a reachable marking of W' such that $m_1 = \phi(m'_1)$

We show that W and W' are equivalent:

1. Let $t = (E_1, j, E_2)$ be a transition executing j such that $m_1 \xrightarrow{t} m_2$ in W, we show that there exists a replacement execution sequence σ and a marking m'_2 such that $m'_1 \xrightarrow{\sigma} m'_2$ in W' and $m_2 = \phi(m'_2)$:

   Because j is enabled in m_1, either some XOR-joins of the replacement are enabled in m'_1, or some incoming edge of the AND-join a of the replacement are marked, or both. The execution σ starts by executing the XOR-joins enabled in m'_1.

   After executing the enabled XOR-join we end up in a state m'_a in which some edges in $\circ a$ are marked. We aim to reach a state such that each edge in $\circ a$ is marked: Let's consider an edge $e'' \in \circ a$ such that e'' is not marked in m'_a. By Lemma 11 there exists a path p in W' from the incoming edge e of the minimal dominator of j in W to e'' such that all edges of p are marked during any execution where an edge in F carries a token. By assumption, it is the case for the execution we discuss.

   Nodes execute at most once during sound execution of an acyclic workflow graph, thus the token is on the path p to e'. The token cannot be on an edge that belongs to E because otherwise j would not be enabled in m'_1 because there would be a path from a token to an empty incoming edge of j. There exists a path from e to e' for which only executes consumes token marking edges of the replacement, i.e, edges in $E' \setminus E$. By Lemma 10, we can reach a making which marks e'.

This approach can be repeated for all empty incoming edges of a allowing to reach a state where each edge in $\circ a$ are marked and thus a is enabled. σ last transition executes a resulting in m'_2.

We have $m_2 \equiv m'_2$ because: σ consumes a token on all edges that where (in W) incoming to j. As mentioned earlier, other transitions only consume tokens marking edges of the replacement. Executing a produces a token on its outgoing edge, which was (in W) and consumes a token on each of its incoming edges.

2. Let σ^* be a replacement execution sequence such that $m'_1 \xrightarrow{\sigma^*} m'_2$ in W', we show that there exists a transition $t = (E_1, j, E_2)$ and a marking m_2 such that $m_1 \xrightarrow{t} m_2$ in W and $m_2 = \phi(m'_2)$:

   By definition of replacement execution we have that the (previously) outgoing edge of j is marked by m'_2. By definition of K-replacement, we have that for any execution σ' of W' containing m'_2, σ' contains a reachable marking m'_0, preceding m'_1 in σ', such that m'_0 marks and edge of F.

   We show that j executes after m_1 in W: Let σ be the execution sequence of W which contains the sequence of marking defined by applying the function ϕ to the markings of $exec'$ until reaching the marking m_1. It is easy to see that the σ contains a marking m_0 of W such that m_1 is reachable from m_0 and $m_0 = \phi(m'_0)$. By assumption, as m_0 is marked by σ, the outgoing edge o of j is marked by any execution sequence containing m_0. The edge o is marked by any marking obtained by executing j in W and any marking obtained by executing the final AND-join f of R in W'. Thus j must be executed after m_0. By replacement execution sequence definition, the final AND-join f of R is executed during σ^*. The IOR-join j executes after reaching m_1 during σ because if it was executed earlier then there would be a marking in σ' marking preceding m'_1 that marks o and f would execute before σ^* in W' and σ^* would execute f a second time which would contradict the soundness assumption.

   We show that a transition t executing j is enabled by m_1: There exists no edge preceding f that is marked in m_2, otherwise the AND-join f could execute a second time by Lemma 7 which would contradict the soundness assumption. Let $A_j = \circ j \cup j\circ$ be the set of edges adjacent to j in W. By definition of replacement execution sequence and of the the function ϕ, $\phi(m'_1) \setminus A_j = \phi(m'_2) \setminus A_j$, i.e., the edges marked by m'_2 which are not adjacent to j in W are marked in m'_1. Thus, no edge preceding an edge of A_j is marked by m_1'. By definition of ϕ, no edge preceding an edge of A_j is marked by m_1. As we have shown that j executes after m_1, there exists an edge e preceding j marked by m_1. As e does not precede an edge of A_j, we have $e \in \circ j$. Therefore a transition t executing j is enabled by m_1 because m_1 marks at least one edge in $\circ j$ and m_1 does not mark an edge preceding a non-marked edge of $\circ j$.

Executing j in m_1 results in a marking m_2. We have shown previously how a replacement execution results in a marking m_2' such that $m_2 = \phi(m_2')$.