RZ 3824 (# Z1206-001) 06/01/2012
Computer Science 24 pages

Research Report

The Difficulty of Replacing an Inclusive OR-Join

Cédric Favre, Hagen Völzer

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Almaden • Austin • Brazil • Cambridge • China • Haifa • India • Tokyo • Watson • Zurich

The Difficulty of Replacing an Inclusive OR-Join

Cédric Favre and Hagen Völzer

IBM Research - Zurich, Switzerland,
{ced,hvo}@zurich.ibm.com

Abstract. Some popular modeling languages for business processes, e.g., BPMN, contain inclusive OR-joins (*IOR-joins*), but others, e.g., Petri nets, do not. Various scenarios in Business Process Management require, or benefit from, translating a process model from one language to another. This paper studies whether the control flow of a process containing IOR-joins can be translated into a control flow without IOR-joins.
First we characterize which IOR-joins can be replaced locally and define a local replacement for each replaceable IOR-join. Then, we present examples that cannot be locally replaced but have a more general translation. We give a non-local replacement technique, together with its condition of applicability, which runs in polynomial time. Finally, we show that there exist simple process models with an IOR-join that cannot be replaced – in the sense that its synchronization behavior cannot be obtained by any combination of AND and XOR gateways. The proof reveals an intrinsic limitation on the replaceability of IOR-joins and hence the translatability of BPMN-like control flow into Petri nets.
This technical report is the extended version of a conference publication [5].

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

One major obstacle to translate process models between those industrial languages and Petri nets is the presence of gateways with inclusive OR (IOR) logic in the industrial languages. An IOR gateway forks or joins a variable set of threads, thereby supporting various workflow patterns [11]. The 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.

In this paper, we study the question to what extent a process model with IOR-joins can be translated into a process model without IOR-joins. More precisely, we focus on the control flow of a business process, which is modeled by a workflow graph. We ask whether a workflow graph with IOR-joins can be translated into a workflow graph with only XOR and AND gateways. Workflow graphs with XOR and AND gateways can be easily translated into an isomorphic Petri net [10].

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 replacability of IOR-joins and hence the translatability of the workflow graph of general process models into Petri nets.

The requirements of a translation from a workflow graph with IOR-joins into a workflow graph without IOR-joins can vary for different use cases. To obtain general, yet useful results, we take the following requirements into account:

- The translated workflow graph must have *equivalent* behavior. Many notions of equivalence exist [12]. 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 is by itself not challenging as one can easily 'unfold' an acyclic workflow graph into its finite full behavior (i.e., computation tree) and then encode this 'unfolding' as a sequential workflow graph. 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, we require that
- the size of the obtained workflow graph must be manageable. An exponential blowup is usually not acceptable. We are not aware of any general translation from workflow graphs with IOR-joins into Petri nets that preserves the behavior and does not incur an exponential blowup.
- Furthermore, we are interested to preserve the structure of the workflow graph as much as possible. This is important if we want to map analysis results between the two workflow graphs. 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.

The paper is structured as follows. In Sect. 3, we will first 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, 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. 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. 6, we relate this result and its implications to the translations of workflow graphs containing IOR-joins into Petri nets.

2 Workflow graphs

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

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 $n\circ$.

Let $G = (N, E, c)$ be an acyclic multi-graph. If x_1, x_2 are two elements in $N \cup E$ such that there is a non trivial 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.

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 $n \in N$. 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.

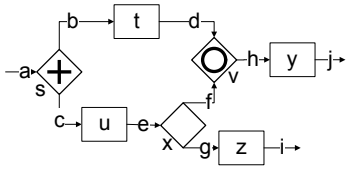

Fig. 1. A workflow graph.

Figure 1 depicts an acyclic workflow graph. A rectangle represents a task node. A diamond containing a plus symbol represents a node with AND logic, an empty diamond represents a node with XOR logic, and a diamond with a circle inside represents a node with IOR 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, IOR, or XOR logic is called *gateway*. For the sake of presentation simplicity, we do not use gateways with multiple incoming edges and multiple outgoing edges. It will become clear later that this restriction does not reduce the expressiveness of workflow graphs because such gateway could be represented by a split and a join of the same logic. Nodes with task logic always have a single incoming edge and a single outgoing

edge. In the rest of this paper, we only consider acyclic workflow graphs, which simplifies some of our definitions such as the semantics of the IOR-join.

The semantics of workflow graphs is, similarly to Petri 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. If n has IOR logic, n can be executed 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. 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 [14], complies with the BPMN 2.0 standard and BPEL's dead path elimination [1]. The choice of the set of outgoing edges to which a token is added when executing a node with XOR or IOR 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:

- $l(n) =$ AND or $l(n) =$ task, $E_1 = \circ n$, and $E_2 = n\circ$.
- $l(n) =$ 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 $E_2 = \{e'\}$.
- $l(n) =$ IOR, $E_1 \subseteq \circ n$, $E_2 \subseteq n\circ$, and E_1 and E_2 are non-empty.

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 $m[e] > 0$ and, if $l(n) = IOR$, $E_1 = \{e \in \circ n \mid m(e) > 0\}$ and for every edge $e \in \circ n \setminus E_1$, there exists no edge e', marked in m, such that $e' < e$. 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.

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. [10]).

3 Local replacements

Fig. 2 and Fig. 3 illustrate 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 plain 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.

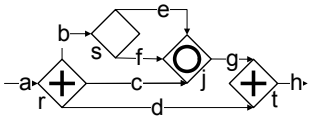

Fig. 2. A workflow graph.

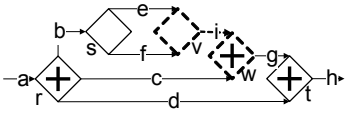

Fig. 3. Local replacement of j by the partial workflow graph composed of the nodes v and w, and the edge i.

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. Such a replacement is a very intuitive way to replace an IOR-join. 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 1 (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* $t \in N$ *iff* $e \in E$ *and* $c(e) = (j,t)$,
  $c'(e) = (s,t)$ *such that* $t \in N''$ *and* $s \in N$ *iff* $e \in E$ *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 iff R has the same "behaviour" as j. We formalize this as follows: 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. A transition (E_1, n, E_2) of W' is a *replacement transition* iff $n \in (N' \setminus N)$. An execution sequence σ of W' is a *replacement execution sequence* iff 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. W and W' are *equivalent* iff 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.

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 everytime it is executed, and by an XOR-join if only one of its incoming edge is marked everytime it is executed [15]. For an acyclic workflow graph, we have shown elsewhere [4] 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* [13] can also provide a local replacement for some IOR-joins.

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 [13] that completes a workflow graph with multiple sinks to obtain a workflow graph with a single sink.

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

Definition 2 (Mutually exclusive edges, test, and cover). *Let* $W = (N, E, c, l)$ *be a workflow graph.*

Two edges in E are mutually exclusive *iff 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 iff 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 $\mathcal{C}$.*

In Fig. 2, 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. 4 illustrates the structure of the replacement:

Definition 3 (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. 2 and Fig. 3 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.

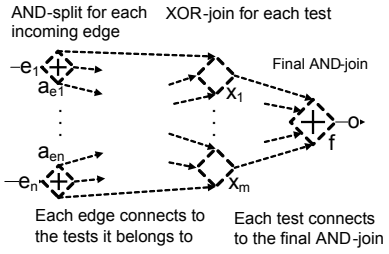

Fig. 4. Canvas of local replacement based on a cover.

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 compute in quadratic time whether a set of edges in an acyclic process is mutually exclusive [4]. 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 1 (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.*

Proof. We first proove the following lemma:

Lemma 1. *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^*. Let p be a path from e to e'.*

There exists a marking m', reachable from m, such that m' marks e' and e^*.

Proof. We proceed by induction on the length of p.

Base case: *length(p) = 1 (i.e., $e = e'$). The marking m marks e^* and e'.*

Induction step: *Assuming that Lemma 1 holds for any path of length n, we show that that it holds for any path of length $n + 1$. Let e_n be the n^{th} edge of p, e' be the next edge, and n be the node between e_n and e'. The prefix of p until e_n is a path of length n from e to e_n. Moreover, there is no path from e^* to e_n otherwise there would be a path from e^* to e'. Therefore, by induction assumption, there exists a marking 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 $m_n \xrightarrow{t} m'$ and m' marks e_* and e'. 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', reachable from m_n, which does not mark e_n , i.e., n is executed. By the workflow graph semantics, executing an IOR-join or an AND-join marks its single outgoing edge e'. Reaching m' 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.

Proof of Thm. 1: *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. 4. 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 1, 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 indepen-

dent 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 1, 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 $\bigcup_{u \in \circ j} T_u$ 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.*

While Thm. 1 applies to any local replacement technique, the proof of the 'if' direction [6] 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.

Fig. 5 is a slight variation of Fig. 2 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. Therefore, 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. 6, 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.

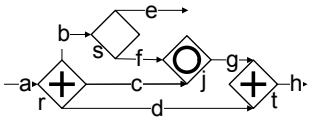

Fig. 5. The edge f does not belong to any test of g that is a subset of $\circ j$.

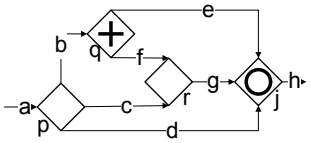

Fig. 6. The edge e does not belong to any test of h that is a subset of $\circ j$.

4 Non-local replacements

Fig. 8 shows an example of a non-local replacement where the IOR-join j of Fig. 7 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.

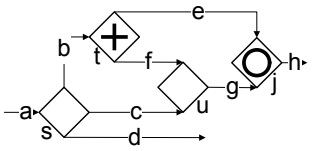

Fig. 7. A workflow graph.

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.

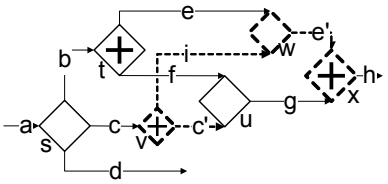

Fig. 8. Non-local replacement of j in Fig. 7.

Kiepuszewski et al. [7, 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 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.

4.1 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 4 (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 $n \in N''$, $l''(n) =$ XOR or $l''(n) =$ 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* $t \in N''$ *iff* $e \in E$ *and* $c(e) = (s,j)$ *or* $e \in E''$ *and* $s \in l_g$,
  $c'(e) = (s,t)$ *such that* $s \in N''$ *and* $t \in N$ *iff* $e \in E$ *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 $\phi(m)[e] = \sum_{\psi(e')=e} m[e']$.

We reuse the notion of *replacement execution sequence* defined in Sect. 3.

W and W' are *equivalent* iff 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 $m_2 = \phi(m'_2)$.

4.2 A simple non-local replacement:

As intermediary step, we discuss informally a simple version of the 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*.

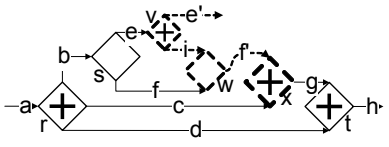

Fig. 9. Non-local replacement of j in Fig. 5.

As intermediary step, we discuss informally a simple version of the 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. 9 and Fig. 10 show equivalent non-local replacements for the workflow graphs in Fig. 5 and Fig. 6, respectively. These are two examples of simple K-replacement.

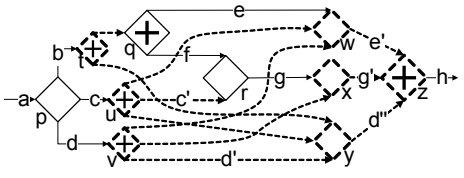

Fig. 10. Non-local replacement of j in Fig. 6.

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. 9, 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 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. 5 and Fig. 6.

However, consider the workflow graph in Fig. 11 obtained by the same technique for the IOR-join v in the workflow graph shown by Fig. 7. When an execution σ marks the edge d of the workflow graph in Fig. 11, 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. 7. 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.

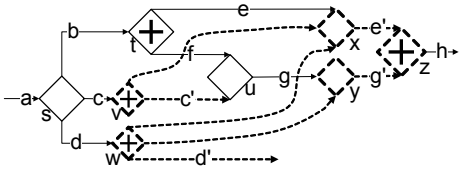

Fig. 11. Non-local replacement of j in Fig. 7.

4.3 K-replacement:

We now describe our generalized technique, called *K-replacement*. Applying K-replacement to f in Fig. 7 results in the workflow graph in Fig. 8. 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 following notions. 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 iff there exists no

node x_1' such that x_1' dominates x_2, x_1 dominates x_1', and $x_1 \neq x_1' \neq x_2$. For example, in Fig. 3, the nodes r and s dominate the node v and the node s is the minimal dominator of v. A set E_d of edges is the *dominator frontier* of a node x_2 iff 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. 7, 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. 8 shows the workflow graph resulting from the application of Algorithm 1 to the IOR-join j in Fig. 7.

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.
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
        Add an edge from s to $x_{e'}$

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

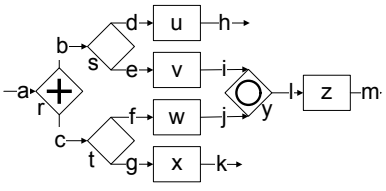

Fig. 12. The IOR-join y cannot be replaced.

this intuition as a necessary and sufficient condition for the K-replacement to produce an equivalent workflow graph:

Theorem 2 (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.*

Proof. We start by proving Lemma 2 which will be useful to prove the condition of applicability given by Thm. 2.

Lemma 2. *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:

⇒ *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 gaph 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 2 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 2 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 1, 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 3 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 $m_2 = \phi(m'_2)$.

Thus, K-replacement of the IOR-join j in Fig. 7 produces an equivalent workflow graph shown in Fig. 8 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. 12. The dominator frontier of y in Fig. 12 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. 12 in which the execution where the edges g and d are marked does not execute z.

4.4 A note on complexity:

We now argue that the condition expressed by Thm. 2 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 3 (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 polymonial 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 [4] we show how to perform a* symbolic execution *of an acyclic workflow graph in quadratic time and space with respect to the size of the workflow graph. In this paragraph, we point out how to use the results of this work to check the condition of applicability ok the K-replacement of an IOR-join j with denominator frontier D_f, i.e., to check that the for any execution σ, σ marks the outgoing edge of j iff σ marks some edge(s) in D_f. The symbolic execution assigns to each edge of the workflow graph a symbol. We described the notion of* symbol equivalence*: Two symbols s_1 assigned to an edge e_1 and s_2 assigned to an edge e_2 are* equivalent *iff for each (concrete) execution σ, σ marks e_1 iff σ marks e_2. We also showed how to compute the* equivalence *of two* symbols *assigned to the edges of the workflow graph by the symbolic execution in linear time. To check that for any execution σ, σ marks the outgoing edge of j iff σ marks some edge(s) in D_f, we check that the symbol assigned to the outgoing edge of j is equivalent to sum of 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 $O(|E|log(|N|))$ [8]. 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.*

5 The difficulty of replacing an IOR-join

As discussed above, the IOR-join y in Fig. 12 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. 12 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. 12 is represented by the workflow graph in Fig. 13. We will show that the workflow graph in Fig. 13 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. 12. In this sense, no combination of AND and XOR gateways can produce the synchronization behavior of the IOR-join in Fig. 12.

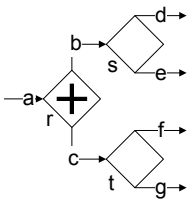

Fig. 13. Prefix that cannot be completed.

We consider the workflow graph in Fig. 13 as a *prefix* of a possible implementation. A *prefix* $P = (N_p, E_p, c_p, l_p)$ of a workflow graph $W = (N, E, c, l)$ is a workflow graph that is a subgraph of $N \cup E$ such that for each pair e_1, e_2 of elements of W such that $e_1 < e_2$, we have: If e_2 belongs to P, then e_1 belongs to P.

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. 12. We allow multiple tasks of the implementing workflow graph to be labeled with z and therefore to correspond to the task z in Fig. 12.

Definition 5 (Equivalent workflow graph properties).
Let W' be a workflow graph which has the prefix illustrated by Fig. 13. 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. 12 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 can now enunciate our result:

Theorem 4 (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. 13 as prefix, W' does not contain any IOR-join, and W' satisfies the properties of Def. 5.*

Proof. This proof rests on the following two lemmas:

Lemma 3. *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 straight-forward induction on p. Lemma 1 implies Lemma 3)

Lemma 4. *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 $e' \in E_1'$.

Proof. 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 1, 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.

Proof of Thm. 4: *We prove this theorem by contradiction: Suppose that there exists a workflow graph W' such that W' has P (illustrated by Fig. 13) 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 4, 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 3). This contradicts P1 because t_1, t_2, and a task labeled z are executed during σ^*.

6 On the translation to Petri nets and related work

We have shown that, in some sense, the IOR-join cannot always be replaced by a combination of AND and XOR gateways. Workflow graphs without IOR gateways are essentially equivalent to free-choice workflow nets, a class of Petri nets for which efficient analysis algorithms exist [2]: A workflow graph without IOR gateways can be translated into an isomorphic free-choice workflow net of about the same size [10] but it can also be shown that a free-choice workflow net can be translated into an isomorphic workflow graph without IOR of about the same size. Hence, there is no free-choice workflow net implementing the IOR-join in Fig. 12 in the sense discussed above.

To translate an acyclic workflow graph into a non-free-choice Petri-net, one can use Dead Path Elimination [1, 14] 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. Alternatively, Mendling, van Dongen, and van der Aalst [9] use the theory of regions to synthesize a Petri net from the reachability graph of the process model. In both approaches, the resulting Petri net is exponentially larger than the original workflow graph and does not preserve the structure of the original process.

A more direct approach to implement the IOR-join from Fig. 12 as a Petri net is shown in Fig. 14, 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. 12. 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.

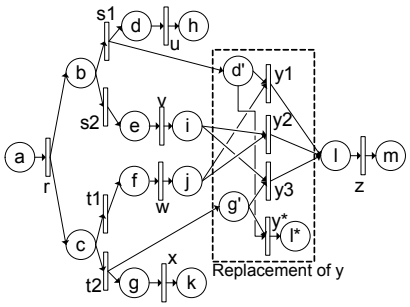

Fig. 14. A non-free-choice Petri net that is equivalent to the one in Fig. 12.

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. 14, 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.

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, Theorem 4 gives a strong argument why IOR-joins cannot be easily implemented by a free-choice net, it points to some difficulty when trying to translate to general Petri nets, and no general polynomial-space translation to general Petri nets is known.

7 Conclusion

In this paper, we studied the difficulty of replacing an IOR-join. For acyclic processes, we established a necessary and sufficient condition that characterizes IOR-joins that can be locally replaced. For the IOR-joins that can be locally replaced, we proposed a generic local replacement. We presented a non-local replacement strategy which allows us to translate some of the IOR-joins that cannot be replaced locally. We have shown that computing this replacement and checking its condition of application can be done in polynomial time with respect to the size of the original workflow graph.

While these results have been presented for the replacement of a single IOR-join in a sound acyclic business process model, they can be applied to replace multiple IOR-joins in an acyclic workflow graph. Moreover, it can be shown that both replacement techniques do not alter the soundness of the process, i.e., cannot introduce or fix a control-flow error which makes these replacement strategies applicable to replace IOR-joins in process models of which the soundness is unknown such as, for example, when performing a control-flow analysis. We have used elsewhere [3] *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 in processes containing cycles.

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 net 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.

References

1. A. Alves et al. Web services business process execution language version 2.0. *OASIS Standard*, 11, 2007.
2. J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.
3. D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, 2011.
4. C. Favre and H. Völzer. Symbolic Execution of Acyclic Workflow Graphs. In *BPM*, volume 6336 of *LNCS*, pages 260–275. Springer, 2010.
5. C. Favre and H. Völzer. The Difficulty of Replacing an Inclusive OR-Join. In *BPM*, LNCS. Springer, 2012.
6. C. Favre and H. Völzer. The Difficulty of Replacing an Inclusive OR-Join. Technical report, IBM Research, RZ1234, 2012.
7. B. Kiepuszewski, A. ter Hofstede, and W. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, 2003.

8. T. Lengauer and R. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–141, 1979.
9. J. Mendling, B. van Dongen, and W. van der Aalst. Getting rid of or-joins and multiple start events in business process models. *Enterprise Information Systems*, 2(4):403–419, 2008.
10. W. van der Aalst, A. Hirnschall, and H. Verbeek. An Alternative Way to Analyze Workflow Graphs. In *CAiSE02, volume 2348 of LNCS*, pages 535–552, 2002.
11. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.
12. R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.
13. J. Vanhatalo, H. Völzer, F. Leymann, and S. Moser. Automatic Workflow Graph Refactoring and Completion. In *ICSOC*, volume 5364 of *LNCS*, 2008.
14. H. Völzer. A New Semantics for the Inclusive Converging Gateway in Safe Processes. In *BPM*, volume 6336 of *LNCS*, pages 294–309. Springer, 2010.
15. M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond. Business Process Verification–Finally a Reality! *Business Process Management Journal*, 15(1):74–92, 2009.