Reduction Rules for YAWL Workflow Nets with Cancellation Regions and OR-joins

M.T. Wynn[1], H.M.W. Verbeek[2], W.M.P. van der Aalst[1,2], A.H.M. ter Hofstede[1] and D. Edmond[1]

School of Information Systems, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia.
{*m.wynn,d.edmond,a.terhofstede*}@*qut.edu.au*
Department of Technology Management, Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, The Netherlands.
{*h.m.w.verbeek,w.m.p.v.d.aalst*}@*tm.tue.nl*

Abstract. A reduction rule can transform a large net into a smaller and simple net while preserving certain interesting properties and it is usually applied before verification to reduce the complexity and to prevent state space explosion. Reset nets have been proposed to formally describe workflows with cancellation behaviour. In our previous work, we have presented a set of reduction rules for Reset Workflow Net (RWF-net), which is a subclass of reset nets. In this paper, we will present a set of reduction rules for YAWL nets with cancellation regions and OR-joins. The reduction rules for RWF-nets combined with the formal mappings from YAWL nets provide us with the means to define a set of reduction rules for YAWL nets. We will also demonstrate how these reduction rules can be used for efficient verification of YAWL nets these features.
Keywords: Petri nets with reset arcs, reset nets, reduction rules, workflow verification, Yet Another Workflow Language, soundness property, cancellation regions, OR-joins.

1 Introduction

Verification of workflows enables the detection of certain desirable properties before actual deployment. If verification is not performed at design time, it is possible for workflows to be running for a long time before errors are discovered. This may, on the one hand, be time consuming and potentially costly to correct. On the other hand, when a workflow contains a large number of tasks and involves complex control flow dependencies, verification can take too much time or it may even be impossible. Applying reduction rules before carrying out verification could decrease the size of the problem by cutting down the size of the workflow that needs to be examined while preserving some essential properties. As a result, reduction rules could potentially decrease average case complexity of performing workflow verification.

Some have advocated the use of Petri nets for the specification of workflows among others due to the formal foundation, their graphical nature and the presence of analysis techniques [4]. There exists a body of work concerning the verification of workflow specifications expressed as Petri nets or expressed in languages for which mappings to

Petri nets have been defined [2, 3, 18]. In either case, verification boils down to examining certain properties of Petri nets. Unfortunately, these results are not transferable to situations where languages are involved that use concepts not easily expressed through Petri nets.

Two typical concepts difficult to express in terms of Petri nets are cancellation regions and OR-joins [7]. *Cancellation* is used to capture the interference of one task in the execution of others. If a task is within the cancellation region of another task, it may be prevented from being started or its execution may be terminated. For example, you might want to simply cancel other order processing tasks if a customer's credit card payment did not go though. An *OR-join* is used in situations when we need to model "wait and see" behaviour for synchronisation. For example, a purchase process could involve the separate purchase of two different items and the customer can decide whether he/she wants to purchase one or the other or both. The subsequent payment task is to be performed only once and this requires synchronisation if the customer has selected both products. If the customer selects only one product, no synchronisation is required before payment. Cancellation and OR-joins change the behaviour of a workflow and verification needs to take into account the effects that these constructs have on the execution of the workflow.

Cancellation and OR-joins occur naturally in workflow specifications and are mentioned in the collection of so called *workflow patterns* identified in [9]. This collection comprises 20 workflow patterns that were proposed to address control flow requirements in a language independent style. They are based on an in-depth analysis and a comparison of a number of commercially available workflow management systems. The findings highlight the need for an expressive workflow language that can support all of these workflow patterns. We believe that it is preferable to support business analysts with an expressive workflow language where verification consequently is more complex rather than to restrict the expressive power of the language in order to make verification simpler (see e.g., also [6]).

We took on the challenge of finding more sophisticated verification techniques for workflows that use cancellation and OR-joins. We are interested in determining whether a workflow possesses the following desirable properties. Firstly, it is important to know that a workflow, when started, can complete. Secondly, it should never have tasks still running when completion is signalled. Thirdly, the workflow should not contain tasks that can never be executed. These requirements encompass *the soundness property* of a workflow specification as expressed in [4]. In [22], we proposed a new verification approach for the soundness property in workflows with cancellation and OR-joins using reset nets. In this paper, we aim to build on this approach through the exploration of possible reduction rules for workflows with cancellation regions and OR-joins. We will take YAWL as the vehicle through which our results are expressed. YAWL is a general and powerful language grounded in workflow patterns and in Petri nets [8]. YAWL provides direct support for cancellation regions and general OR-joins. YAWL has a formal foundation and an open-source support environment is available. The YAWL Editor (version 1.4) provides verification support based on the approach described in [22]. A mapping exists for YAWL specifications without OR-joins to reset nets [20]. In order to study the verification of YAWL specification containing OR-joins, we also need to

study reduction rules at the YAWL level. The reduction rules for reset nets as shown in [23] combined with the formal mappings from YAWL specifications to reset nets provide us with the means to formally prove the correctness of reduction rules at the YAWL specification level.

In this paper, we define a set of reduction rules at YAWL net level and demonstrate how these reduction rules can be used for efficient verification of YAWL nets with cancellation regions and OR-joins[1]. This is done for the following reasons. Firstly, by applying reduction rules at YAWL level, problematic tasks and conditions could be highlighted with ease and meaningful error messages could be provided based on YAWL terminology. Secondly, reduction rules for YAWL nets without OR-joins can still be applied to YAWL nets with OR-joins and in particular to those parts of the net that do not use any OR-join constructs. This enables us to reduce the complexity of the verification process for YAWL nets with OR-joins. Finally, by first abstracting from non OR-join constructs in YAWL nets with OR-joins, the resulting YAWL nets with OR-joins may become much simpler. This allows us to define some reduction rules for YAWL nets with OR-joins, which will be explained in more detail in Section 4.

The organisation of the paper is as follows. Section 2 provides the formal foundation by introducing reset nets and Reset Workflow Nets(RWF-nets) and discusses briefly a set of reduction rules for RWF-nets. Section 3 discusses ten reduction rules for YAWL nets without OR-joins. The proof for each rule makes use of a series of transformations at reset net level. Section 4 describes additional reduction rules for YAWL nets with OR-joins. Section 5 describes the implementation of our approach in the YAWL editor. Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminaries

The formal semantics of YAWL is expressed in terms of a transition system [8] and while inspired by Petri nets, YAWL should not be seen as an extension of these. YAWL constructs such as OR-join, cancellation and multiple instances are not directly supported by Petri nets. The cancellation feature of YAWL is theoretically closely related to reset nets, which are Petri nets with reset arcs. Next, we present background definitions for Petri nets and Reset nets.

2.1 Petri nets and Reset nets

Petri nets were originally introduced by Carl Adam Petri [15] and since then, they are widely used as mathematical models of concurrent systems for various domains [14, 10]. Numerous analysis techniques exist to determine various properties of Petri nets and its subclasses [14, 10, 13, 16, 17].

Definition 1 (Petri net [15, 14]). *A Petri net is a tuple (P, T, F) where P is a (non-empty finite) set of places, T is a set of transitions, $P \cap T = \varnothing$ and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs.*

[1] The bulk of this work was done while visiting Eindhoven University of Technology in close collaboration with Dr. Eric Verbeek and Professor Wil van der Aalst.

A reset net is a Petri net with special *reset arcs*, that can clear the tokens in selected places.

Definition 2 (Reset net [12]). *A reset net is a tuple* (P, T, F, R) *where* (P, T, F) *is a Petri net and* $R : T \rightarrow \mathbb{P}(P)$ *provides the reset places for the transitions*[2].

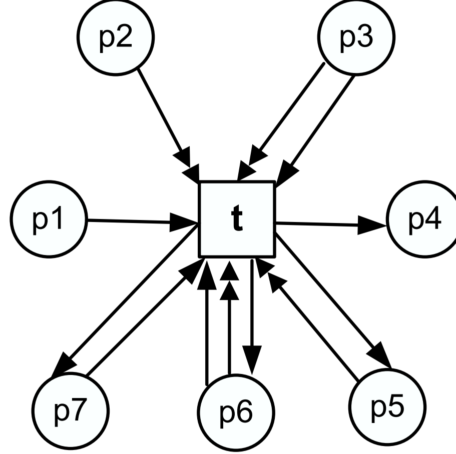

Fig. 1. An example reset net

In the remainder of the paper, when we use the function $F(x, y)$, it evaluates to 1 if $(x, y) \in F$ and 0 if $(x, y) \notin F$. We write F^+ for the transitive closure of the flow relation F and F^* for the reflexive transitive closure of F. R^{-1} is the (straightforward) inverse function of R where $R^{-1} \in P \rightarrow \mathbb{P}(T)$. The notation $R(t)$ for a transition t returns the (possibly empty) set of places that it resets. We also write $R^{\leftharpoonup}p$ for a place p, which returns the set of transitions that can reset p.

Let N be a reset net and $x \in (P \cup T)$, we use $\bullet x$ and $x\bullet$ to denote the set of inputs and outputs. If the net involved cannot be understood from the context, we explicitly include it in the notation and we write $\overset{N}{\bullet} x$ and $x \overset{N}{\bullet}$. A marking is denoted by M and, just as with ordinary Petri nets, it can be interpreted as a vector, function, and multiset over the set of places P. $M(p)$ returns the number of tokens in a place p if $p \in dom(M)$ and zero otherwise. We can use notations such as $M \leq M'$, $M + M'$, and $M \dot{-} M'$. $M \leq M'$ iff $\forall_{p \in P} M(p) \leq M'(p)$. $M + M'$ and $M \dot{-} M'$ are multisets such that $\forall_{p \in P}$: $(M + M')(p) = M(p) + M'(p)$ and $(M \dot{-} M')(p) = M(p) \dot{-} M'(p)$ [3]. We represent a multiset by simply enumerating the elements, e.g., $2a + 3b + c$ is the multiset containing two a's, three b's and one c. If X is a set over Y, it could also be interpreted as a bag which assigns to each element a weight of 1.

The notation $\mathbb{M}(N)$ is used to represent possible markings of a reset net N.

[2] Where $\mathbb{P}$ is a power set of P, i.e., $X \in \mathbb{P}$ if and only if $X \subseteq P$.

[3] For any natural numbers a, b: $a \dot{-} b$ is defined as $\max(a - b, 0)$.

Definition 3 **($\mathbb{M}(N)$).** *Let $N = (P, T, F, R)$ be a reset net, then $\mathbb{M}(N) = P \rightarrow \mathbb{N}$ is the set of possible markings.*

A transition is *enabled* when there are enough tokens in its input places. Note that reset arcs do not change the requirements of enabling a transition.

Definition 4 **(Enabling rule).** *Let N be a reset net, $t \in T$, and $M \in \mathbb{M}(N)$. Transition t is enabled at M, denoted as $M[t\rangle$, if and only if $\forall p \in \bullet t : M(p) \geq 1$.*

The concept of firing a transition t in a net N is formally defined in Definition 5 and denoted as $M \overset{N,t}{\rightarrow} M'$. If there can be no confusion regarding the net, the expression is abbreviated as $M \overset{t}{\rightarrow} M'$ and if the transition is not relevant, it is written as $M \rightarrow M'$.

Definition 5 **(Forward firing).** *Let $N = (P, T, F, R)$ be a reset net, $t \in T$ and $M, M' \in \mathbb{M}(N)$.*

$$M \overset{N,t}{\rightarrow} M' \Leftrightarrow M[t\rangle \wedge$$
$$M'(p) = \begin{cases} M(p) - F(p,t) + F(t,p) & \text{if} \quad p \in P \setminus R(t) \\ F(t,p) & \text{if} \quad p \in R(t). \end{cases}$$

It is possible to fire a sequence of transitions from a given marking in a reset net resulting in a new marking using the forward firing rule defined above. This sequence of transitions is represented as an occurrence sequence.

Definition 6 **(Occurrence sequence).** *Let $N = (P, T, F, R)$ be a reset net and $M, M_1, ..., M_n \in \mathbb{M}(N)$. If $M \overset{t_1}{\rightarrow} M_1 \overset{t_2}{\rightarrow} ... \overset{t_n}{\rightarrow} M_n$ are firing occurrences then $\sigma = t_1 t_2 ... t_n$ is an occurrence sequence leading from M to M_n and it is written as $M \overset{\sigma}{\rightarrow} M_n$.*

We now define the concepts of reachability and coverability of markings from a given marking in a reset net. A marking M' is reachable from another marking M in a reset net, if there is an occurrence sequence leading from M to M'.

Definition 7 **(Reachability).** *Let $N = (P, T, F, R)$ be a reset net and $M, M' \in \mathbb{M}(N)$. M' is reachable in N from M, denoted $M \overset{N}{\rightarrow} M'$, if there exists an occurrence sequence σ such that $M \overset{\sigma}{\rightarrow} M'$.*

The*reachability set* is the minimal set of markings that can be reached from a given marking M in a reset net after firing all possible occurrence sequences.

Definition 8 **(Reachability set).** *Let $N = (P, T, F, R)$ be a reset net and $M \in \mathbb{M}(N)$. The reachability set of the marked net (N, M), denoted $N[M\rangle$, is the minimal set that satisfies the following conditions:*

1. *$M \in N[M\rangle$ and*
2. *if transition $t \in T$ and markings $M_1, M_2 \in \mathbb{M}(N)$ exist such that $M_1 \in N[M\rangle$ and $M_1 \overset{N_1,t}{\rightarrow} M_2$, then $M_2 \in N[M\rangle$.*

Definition 9 **(Directed labelled graph).** *A directed labelled graph $G = (V, E)$ over label set $\mathcal{L}$ consists of a set of nodes V and a set of labelled edges $E \subseteq V \times \mathcal{L} \times V$.*

The *reachability graph* is a directed labelled graph where the elements of the reachability set form the nodes and the tuple consisting of a source marking that enables a transition, the transition and the target marking that is reached by firing the transition form the edges. The graph can be used to determine the possible states of a reset net from an initial marking.

Definition 10 (Reachability graph). *Let $N = (P, T, F, R)$ be a reset net and $M \in \mathbb{M}(N)$. The directed labelled graph $G = (V, E)$ with label set $\mathcal{L} = T$ is the reachability graph of the marked net (N, M) iff*

1. *$V = N[M\rangle$ and*
2. *for any transition $t \in T$ and markings $M_1, M_2 \in \mathbb{M}(N) : M_1 \xrightarrow{t} M_2 \Leftrightarrow (M_1, t, M_2) \in E$.*

Liveness, boundedness and safeness are defined as in previous work [14, 13]. Liveness, boundedness and safeness can be determined from the reachability graph.

Definition 11 (Liveness, boundedness, safeness [14, 13]). *A transition is live if it can be enabled from every reachable marking. A place is safe if it never contains more than one token at the same time. A place is k-bounded if it will never contain more than k tokens. A place is bounded if it is k-bounded for some k.*

If all places in a reset net are bounded, the reset net is also bounded and hence, it is possible to generate a finite reachability set. If a place is unbounded, the reachability set contains an infinite number of states (*an infinite state space*). In such cases, reachability of a marking cannot be determined but coverability can be determined. Coverability is a relaxed notion that can handle unbounded behaviour. A marking M_2 is said to be *coverable* from another marking M_1 in a reset net if there is a reachable marking M' from M_1 such that M' is bigger than or equal to M_2.

Definition 12 (Coverability). *Let $N = (P, T, F, R)$ be a reset net and $M_1, M_2 \in \mathbb{M}(N)$. M_2 is coverable from M_1 in N, if there exists a marking M' such that $M' \in N[M_1\rangle$ and $M' \geq M_2$.*

We conclude this section with the notion of *Backward firing* that is used to generate coverable markings for a reset net by firing transitions backwards.

Definition 13 (Backward firing [21]). *Let (P, T, F, R) be a reset net and $M, M' \in \mathbb{M}(N)$. $M' \dashrightarrow^t M$ iff it is possible to fire a transition t backwards starting from M and resulting in M'.*

$$M' \dashrightarrow^t M \Leftrightarrow M[R(t)] \leq t\bullet[R(t)] \wedge$$
$$M'(p) = \begin{cases} (M(p) \dot{-} F(t,p)) + F(p,t) & \text{if} \quad p \in P \setminus R(t) \\ F(p,t) & \text{if} \quad p \in R(t). \end{cases}$$

For places that are not reset places, the number of tokens in M' is determined by the number of tokens in M for p and the production and consumption of tokens. If a place is an output place of t and not a reset place, one token is removed from $M(p)$ if $M(p) > 0$. If a place is an input place of t and not a reset place, one token is added to $M(p)$. For any reset place p, $M(p) \leq F(t,p)$ because it is emptied when firing and then $F(t,p)$

tokens are added. We do not require $M(p) = F(t, p)$ for a reset place p because the aim is coverability and not reachability. M', i.e., the marking before (forward) firing t, should *at least* contain the *minimal* number of tokens required for enabling t and resulting in a marking of *at least* M. Therefore, only $F(p, t)$ tokens are assumed to be present in a reset place p.

2.2 Reset WorkFlow nets (RWF-nets)

Definition 14 (WF-net [3, 18]). *Let $N = (P, T, F)$ be a Petri net. The net N is a WF-net iff the following three conditions hold:*

1. *there exists exactly one $i \in P$ such that $\bullet i = \varnothing$, and*
2. *there exists exactly one $o \in P$ such that $o\bullet = \varnothing$, and*
3. *for all $n \in P \cup T$: $(i, n) \in F^*$ and $(n, o) \in F^*$.*

The notion of a Reset WorkFlow net (RWF-net) is introduced to represent workflows with cancellation features. We define Reset WorkFlow nets (RWF-nets) which are reset nets with the same structural restrictions as WF-nets.

Definition 15 (RWF-net [19]). *Let $N = (P, T, F, R)$ be a reset net. The net N is an RWF-net iff (P, T, F) is a WF-net.*

In an RWF-net, there is an input place i and an output place o and an initial marking M_i and an end marking M_o is defined as follows:

Definition 16 (Initial marking and End marking). *Let $N = (P, T, F, R)$ be an RWF-net and i, o be the input and output places of the net. The initial marking of N is denoted as M_i and it represents a marking where there is a token in the input place i (i.e., $M_i = i$). Similarly, the end marking of N is denoted as M_o and it represents a marking where that is a token in the output place o (i.e., $M_o = o$).*

A WF-net is an RWF-net iff R is empty (for all $t \in T : R(t) = \varnothing$). Thus (P, T, F) suffices (we may omit R).

The *soundness* definition for an RWF-net is based on the soundness definition from [8] for WF-nets. An RWF-net is sound if and only if it satisfies the three criteria: option to complete, proper completion and no dead transitions.

Definition 17 (Soundness [19]). *Let $N = (P, T, F, R)$ be an RWF-net and M_i, M_o be the initial and end markings. N is sound iff:*

1. *option to complete: for every marking M reachable from M_i, there exists an occurrence sequence leading from M to M_o, i.e., for all $M \in N[M_i\rangle : M_o \in N[M\rangle$, and*
2. *proper completion: the marking M_o is the only marking reachable from M_i with at least one token in place o, i.e, for all $M \in N[M_i\rangle : M \geq M_o \Rightarrow M = M_o$, and*
3. *no dead transitions: for every transition $t \in T$, there is a marking M reachable from M_i such that $M[t\rangle$, i.e, for all $t \in T$ there exists an $M \in N[M_i\rangle$ such that $M[t\rangle$.*

2.3 Reduction rules for RWF-nets

We now briefly summarize seven reduction rules for RWF-nets for completeness. The detailed explanation of these rules together with associated proofs can be found in our earlier paper [23]. These rules for RWF-nets are based on existing reduction rules for Petri nets and free-choice nets [13, 10] and they have been extended and generalised as necessary. Figure 2 visualises these seven reduction rules.

1. Fusion of series places rule (ϕ^R_{FSP})
   The ϕ^R_{FSP} rule allows for the merging of two sequential places p and q with one transition t in between them into a single place r which takes on the same reset arcs as p and q (if any). The rule only holds if transition t does not have any reset arcs and the two places are reset by the same set of transitions.
2. Fusion of series transitions rule (ϕ^R_{FST})
   The ϕ^R_{FST} rule allows for the merging of two sequential transitions t and u with one place p in between them into a single transition v. Additional requirements (required to allow for reset arcs) are that place p and output places of u should not be the source of any reset arcs and transition u should not reset any place. The rule allows reset arcs from transition t and these arcs will be assigned to the new transition v in the reduced net.
3. Fusion of parallel places rule (ϕ^R_{FPP})
   The ϕ^R_{FPP} rule allows for the merging of places in Q (i.e., p_1 to p_L) that have the same inputs and outputs into a single place q. If none of the places are reset places, then it is obvious that the rule holds. If one is a reset place, then other places should also be reset by the same set of transitions. As all places in $Q = \{p_1, ..., p_L\}$ have the same input, output and reset arcs, these identical places can be merged into a single place while preserving the soundness property. Place q in the reduced net has the same input, output and reset arcs as any place $p \in Q$.
4. Fusion of parallel transitions rule (ϕ^R_{FPT})
   The ϕ^R_{FPT} rule allows for the merging of transitions V (i.e., t_1 to t_L) that have the same inputs and outputs into a single transition v. If no transition has reset arcs, then it is obvious that the rule holds. If one transition resets a place, then other transitions must also reset the same place. As all transitions in $V = \{t_1, ..., t_L\}$ now have the same input, output and reset arcs, these identical transitions could be merged into a single transition while preserving the soundness property. Transition v in the reduced net has the same input, output and reset arcs as any transition $t \in V$.
5. Abstraction rule (ϕ^R_{A})
   The ϕ^R_{A} rule allows the removal of a place s and a transition t, where s is the only input of t, t is the only output of s and there is no direct connection between the inputs for s with the outputs for t. The rule holds if and only if transition t does not reset any place, place s is not reset by any transition, and outputs for t are not reset by any transition. Input transitions for place s can have reset arcs.
6. Elimination of self-loop transitions rule (ϕ^R_{ELT})
   The ϕ^R_{ELT} rule allows removal of a transition t which has a single place as its input and its output. The additional requirement is that transition t has no reset arcs.

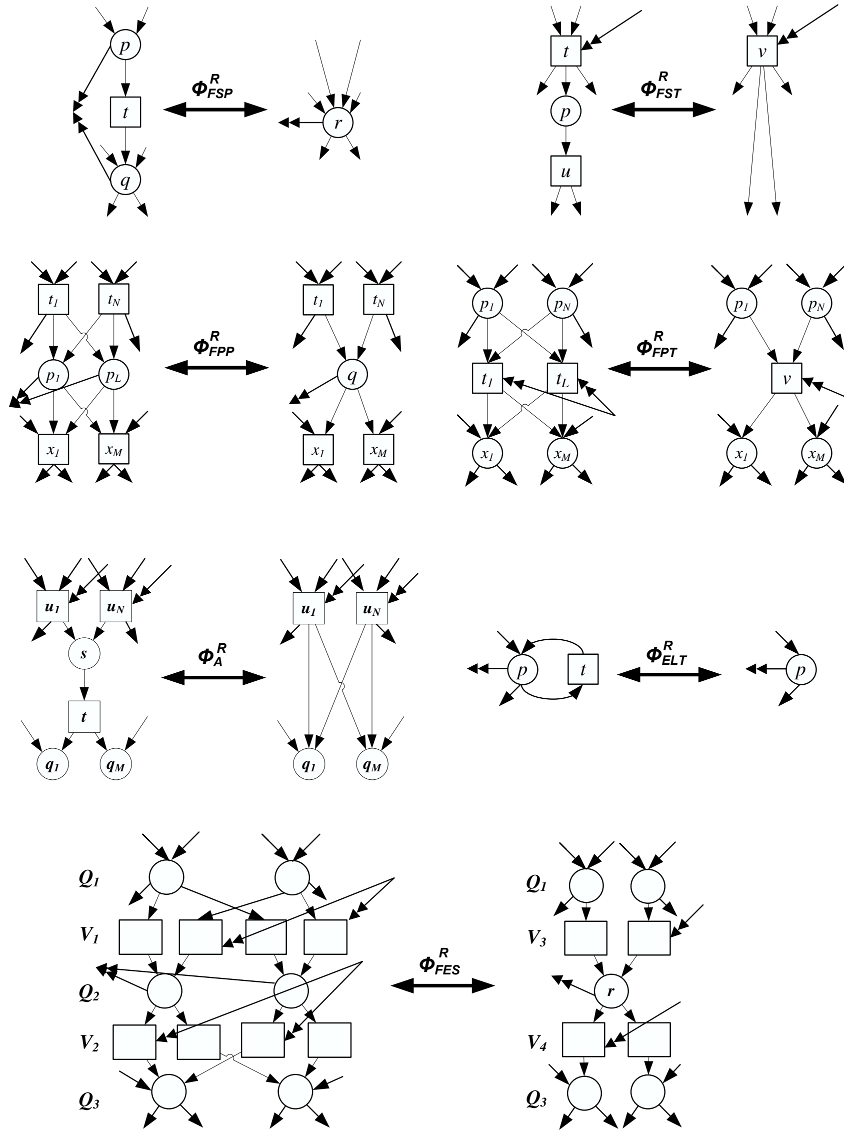

Fig. 2. Reduction rules for RWF-nets

7. Fusion of equivalent subnets (ϕ^R_{FES})
   The ϕ^R_{FES} rule allows the removal of multiple identical subnets by replacing them with only one subnet. The rule requires that pairs of transitions have the same input and output places. That is, all places in Q_2 are reset by the same set of transitions and all transition pairs in V_1 and V_3 also reset the same places. The set of transitions V_1 and V_2 are replaced by V_3 and V_4 respectively in the reduced net. The set of places Q_2 has been replaced with r. The rule is very effective in reducing YAWL models as we will see in the next section.

3 Reduction rules for nets with cancellation regions and without OR-joins

3.1 Introduction to YAWL

The control flow perspective of a workflow specification captures the execution interdependencies between the tasks of a business process. An in-depth analysis and a comparison of a number of commercially available workflow management systems had been performed [9]. The findings demonstrate that the interpretation of even the basic control flow constructs is not uniform and it is often unclear how the more complex requirements could be supported. Twenty workflow patterns were proposed to address control flow requirements in a language independent style [9]. Yet Another Workflow Language (YAWL) is the result of this analysis [8, 9]. YAWL exploits concepts from Petri nets and provides direct support for most workflow patterns proposed in [8, 9].

A YAWL specification is made up of tasks, conditions and a flow relation between tasks and conditions. YAWL uses the terms tasks and conditions to avoid confusion with Petri net terminology (transitions and places). An overview of YAWL can be found in [8]. Figure 3 shows some of the YAWL constructs used in this paper. There are three kinds of splits (AND, XOR and OR) and three corresponding kinds of joins in YAWL. A task is enabled when there are enough tokens in its input conditions according to the join behaviour. When a task is enabled, it fires and takes tokens out of the input conditions and puts tokens in its output conditions according to the join and split behaviour respectively. If there is a cancellation set associated with a task, the execution of the task removes all the tokens from the conditions and tasks in the cancellation set. Cancelling a task will stop the execution of the task.

We propose to abstract from the following features of YAWL for the purpose of verification.

- *composite tasks and hierarchy*: A YAWL specification could contain multiple YAWL nets with hierarchical structure and each composite task is unfolded into one of these nets. We analyse each YAWL net individually to determine the soundness property of the net and hence, we abstract from composite tasks and ignore the hierarchical structure.
- *multiple instances task*: A multiple instances task can be used to execute a particular task a number of times in parallel. For this abstraction, we assume that the engine is capable of keeping the multiple instances apart, and that it will synchronise them at the end. This assumption is consistent with the definition of the language and implementation. Therefore, we only need to take a single instance into account.

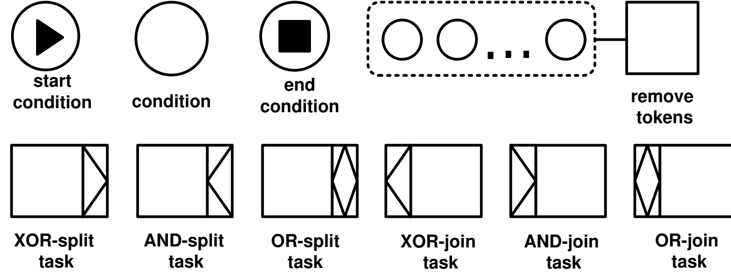

Fig. 3. Symbols in YAWL

– *data perspective*: We also abstract from data perspective and in particular, branching conditions of XOR-split and OR-split tasks are not taken into account when applying reduction rules. As a result, two XOR-split or OR-split tasks with identical input and output nodes are considered to be equivalent regardless of branching conditions assigned to each arc, i.e., choices are assumed to be non-deterministic.

A YAWL specification is formally defined as a nested collection of Extended Workflow Nets (EWF-nets) [8]. A YAWL specification supports hierarchy and a composite task is unfolded into another EWF-net. We refer the reader to [8] for a formal definition of a YAWL specification. In an EWF-net, it is possible for two tasks to have a direct connection. We add an implicit condition between two tasks if there is a direct connection between them and we define the corresponding explicit EWF-net (E2WF-net) for an EWF-net in [20]. The corresponding E2WF-net is represented by the tuple $(C, \mathbf{i}, \mathbf{o}, T, F, \mathit{split}, \mathit{join}, \mathit{rem}, \mathit{nofi})$ where C is a set of conditions, T is a set of tasks, i, o are unique input and output conditions, F is the flow relation, split and join specify the split and join behaviours of each task, rem specifies the cancellation region for a task and nofi specifies the multiplicity of each task. As we abstract from multiple instances, nofi function can be abstracted from the definition as well. From now on, we will use the tuple $(C, \mathbf{i}, \mathbf{o}, T, F, \mathit{split}, \mathit{join}, \mathit{rem})$ to represent a YAWL net. The notion of preset and postset defined for an RWF-net is also be used for an E2WF-net. For simplicity, we propose the synonyms a "YAWL net" and an "eYAWL-net" (explicit YAWL net) for an EWF-net and an E2WF-net respectively. We assume that all YAWL nets considered in this paper are first transformed into eYAWL-nets.

The concepts of reachability and coverability are defined using YAWL semantics as in [8, 20]. This definition of soundness for YAWL is very similar to the notion of soundness introduced in Definition 17. The main difference is that Definition 17 refers to RWF-nets rather than eYAWL-nets.

Definition 18 (Soundness). *Let N be an eYAWL-net, i, o be the input and output conditions of the net and M_i, M_o be the initial and end markings, i.e., $M_i = i$ is the initial state marking only condition i and $M_o = o$ is the end state marking only condition o. N is sound iff:*

– *for every marking M reachable from M_i, there exists a firing sequence leading from M to M_o (Option to complete),*

– *the marking M_o is the only marking reachable from M_i with at least one token in condition o (Proper completion) and*
– *for every task $t \in T$, there is a marking M reachable from M_i such that t is enabled at M (No dead transitions).*

A mapping from a YAWL-net *without OR-joins* to an RWF-net already exists using the transformation function defined in [20]. In general, a condition is mapped onto a place, and a task onto two sets of transitions and an intermediate place. The transitions in the first set start the task (modelling the join behaviour), whereas the transitions in the second set complete it (modelling the split behaviour). Figure 4 shows the transformation for YAWL tasks and conditions. In Figure 4, we use labels S and E to denote start transitions and end transitions and condition names to differentiate transitions within a particular set (e.g., transition $t_S^{p_1}$ represents the start transition for task t that has p_1 as its input). If the split behaviour of a YAWL task is not explicitly mentioned, it could be one of XOR, AND or OR splits. Similarly, join behaviour could be one of XOR or AND joins. Note that we first limit ourselves to nets without OR-joins. In YAWL, cancellation regions can be associated with both tasks and conditions and they are denoted as dotted lines around the elements. If a running task is cancelled, it will stop the execution immediately. A task within a cancellation region in YAWL is mapped to a reset arc of its intermediate place in the corresponding RWF-net. An example of this is given in the bottom-left corner of Figure 4.

Due to these mappings, it is possible to perform reduction of YAWL nets without OR-joins by first transforming the net into the corresponding RWF-net and then apply the reduction rules defined in [23]. However, we decide to define a set of reduction rules at YAWL net level for the following reasons. Firstly, by applying reduction rules at YAWL net level, problematic tasks and conditions could be highlighted with ease and meaningful error messages could be provided based on YAWL terminology. As complex mappings between YAWL-net and the corresponding RWF-net do not need to be kept, verification can be performed more efficiently. Secondly, reduction rules for YAWL nets without OR-joins can still be applied to YAWL nets with OR-joins and in particular to those parts of the net that do not use any OR-join construct. This enables us to reduce the complexity of verification process for YAWL nets with OR-joins. Finally, by first abstracting from other non OR-join constructs in YAWL nets with OR-joins, the resulting YAWL nets with OR-join become much simpler and this allows us to define some reduction rules for YAWL nets with OR-joins, which will be explained in more detail in Section 4. Next, we present ten reduction rules for YAWL constructs based on the reduction rules for RWF-nets. The proof for each rule makes use of a series of transformations at reset net level.

3.2 Fusion of series conditions

The *Fusion of Series Conditions Rule for YAWL nets* (ϕ^R_{FSP}) allows for the merging of two sequential conditions p and q in a YAWL net that have only one task t in between them into a single condition. The ϕ^Y_{FSP} rule makes use of the ϕ^R_{FSP} rule for RWF-nets. The application requirements are similar to those for the ϕ^R_{FSP} rule except that we refer to tasks and conditions instead of transitions and places. In addition, we require that

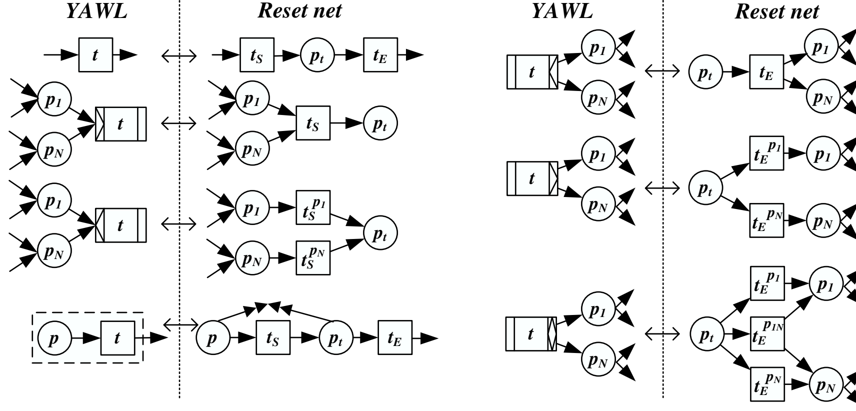

Fig. 4. Transformations from YAWL net to RWF-net

task t is cancelled by the same set of tasks that remove tokens from conditions p and q. This is because a task can be part of a cancellation region while a transition cannot. Figure 5 visualises the ϕ^{Y}_{FSP} rule. Conditions p and q are merged into a new condition r and the in-between task t is abstracted in the reduced net.

Definition 19 (Fusion of Series Conditions Rule: ϕ^{Y}_{FSP}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^{Y}_{\mathrm{FSP}}$ if there exists a task $t \in T_1$, two conditions $p, q \in C_1 \setminus \{i, o\}$ and a condition $r \in C_2 \setminus (C_1 \cup T_1)$ such that:*

Conditions on N_1:

1. $\bullet t = \{p\}$ *(p is the only input of t)*
2. $t\bullet = \{q\}$ *(q is the only output of t)*
3. $p\bullet = \{t\}$ *(t is the only output of p)*
4. $\bullet p \cap \bullet q = \varnothing$ *(any input of p is not an input of q and vice versa)*
5. $rem_1(t) = \varnothing$ *(t does not reset)*
6. $rem_1^{\leftarrow}(p) = rem_1^{\leftarrow}(q) = rem_1^{\leftarrow}(t)$ *(p, q, and t are reset by the same set of tasks)*

Construction of N_2:

7. $C_2 = (C_1 \setminus \{p, q\}) \cup \{r\}$
8. $T_2 = T_1 \setminus \{t\}$
9. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (\overset{N_1}{\bullet} p \times \{r\}) \cup (\{r\} \times q \overset{N_1}{\bullet}) \cup ((\overset{N_1}{\bullet} q \setminus \{t\}) \times \{r\})$
10. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \oplus \{(z, (rem_1(z) \cap (C_2 \cup T_2)) \cup \{r\}) | z \in rem_1^{\leftarrow}(p)\}$
11. $split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\}$
12. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\}$

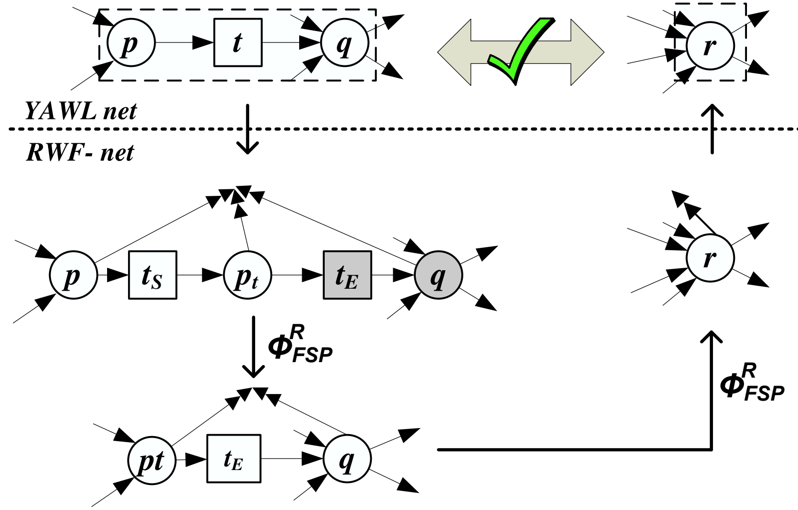

Fig. 5. Fusion of Series Conditions Rule for YAWL nets: ϕ^Y_{FSP}

Theorem 1 (The ϕ^Y_{FSP} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^Y_{\mathrm{FSP}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 5 visualises the ϕ^Y_{FSP} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. First, conditions p, q and task t in the YAWL net are transformed into corresponding places and transitions in the RWF-net. The ϕ^R_{FSP} rule for an RWF-net is then applied twice to obtain a reduced RWF-net. In the first step, places p and p_t are merged into one place pt and transition t_S is abstracted. In the second instance, places pt and q are merged into one place r and transition t_E is abstracted. Finally, the reduced RWF-net is mapped back to the YAWL level with a condition r replacing the place r. Since the ϕ^R_{FSP} rule is soundness preserving, the sequence of transformations is also soundness preserving. ∎

3.3 Fusion of parallel conditions

The *Fusion of Parallel Conditions Rule for YAWL nets* (ϕ^Y_{FPP}) allows for the merging of two or more parallel conditions in a YAWL net with the same input tasks and the same output tasks into a single condition. The ϕ^Y_{FPP} rule makes use of the ϕ^R_{FPP} rule for RWF-nets. The application requirements are the same as those for the ϕ^R_{FPP} rule except that we refer to tasks and conditions instead of transitions and places. In addition, we require that all input tasks for these conditions are AND-split tasks and all output tasks for these conditions are AND-join tasks. Figure 6 visualises the ϕ^Y_{FPP} rule. Multiple identical conditions are merged into a single condition in the reduced net with the same input, output, and reset arcs as those conditions in the original net.

Definition 20 (Fusion of Parallel Conditions Rule: ϕ^Y_{FPP}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^Y_{\mathrm{FPP}}$ if there exists tasks $T, X \subseteq T_1 \cap T_2$ where $|T| \geq 1, |X| \geq 1$, conditions $P \subseteq C_1$ where $|P| \geq 2$, and a condition $c \in C_2 \setminus (C_1 \cup T_1)$ such that:*

Conditions on N_1:

1. *for all $t \in T$: $split_1(t) = AND$ (all tasks in T are AND-split tasks)*
2. *for all $x \in X$: $join_1(x) = AND$ (all tasks in X are AND-join tasks)*
3. *for all $p \in P$: $\bullet p = T$ and $p\bullet = X$ (all conditions in P have the same input and output)*
4. *for all $p, q \in P$: $rem_1^{\leftarrow}(p) = rem_1^{\leftarrow}(q)$ (all conditions in P are reset by the same tasks)*

Construction of N_2:

5. *$C_2 = (C_1 \setminus P) \cup \{c\}$*
6. *$T_2 = T_1$*
7. *$F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (T \times \{c\}) \cup (\{c\} \times X)$*
8. *$rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_1\} \oplus \{(z, (rem_1(z) \cap (C_2 \cup T_2)) \cup \{c\}) | z \in rem_1^{\leftarrow}(p) \wedge p \in P\}$*
9. *$split_2 = split_1$*
10. *$join_2 = join_1$*

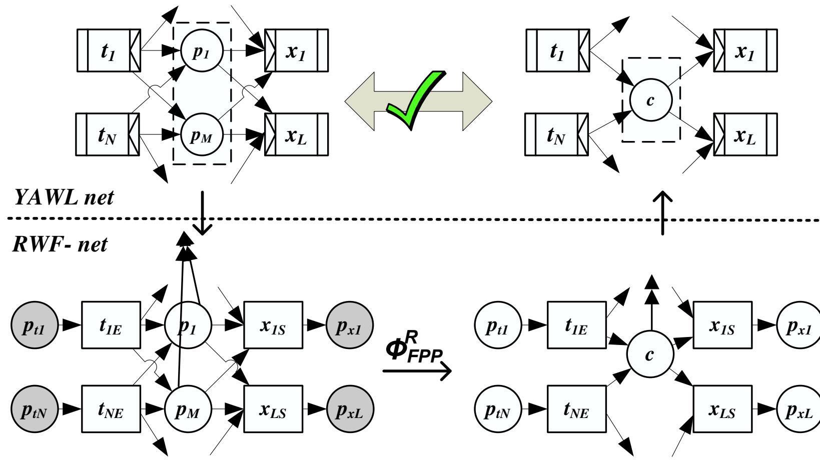

Fig. 6. Fusion of Parallel Conditions Rule for YAWL nets: ϕ^Y_{FPP}

Theorem 2 **(The ϕ^Y_{FPP} rule is soundness preserving).** *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^Y_{\mathrm{FPP}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 6 visualises the ϕ^Y_{FPP} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.4 Fusion of alternative conditions

The *Fusion of Alternative Conditions Rule for YAWL nets* (ϕ^Y_{FAP}) allows for the merging of two or more alterative conditions in a YAWL net with the same input tasks and output tasks. The ϕ^Y_{FAP} rule makes use of the ϕ^R_{FES} rule for RWF-nets. The application requirements are the same as those for the ϕ^R_{FES} rule except that we refer to tasks and conditions instead of transitions and places. In addition, we require that all input tasks for these conditions are XOR-split tasks and all output tasks for these conditions are XOR-join tasks. Figure 7 visualises the ϕ^Y_{FAP} rule. Multiple conditions are merged into a single condition in the reduced net with the same input, output, and reset arcs as those conditions in the original net.

Definition 21 **(Fusion of Alternative Conditions: ϕ^Y_{FAP}).** *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^Y_{\mathrm{FPP}}$ if there exists tasks $T, X \subseteq T_1 \cap T_2$ where $|T| \geq 1, |X| \geq 1$, conditions $P \subseteq C_1$ where $|P| \geq 2$, and a condition $c \in C_2 \setminus (C_1 \cup T_1)$ such that:*

Conditions on N_1:

1. *for all $t \in T$: $split_1(t) = XOR$ (all tasks in T are XOR-split tasks)*
2. *for all $x \in X$: $join_1(x) = XOR$ (all tasks in X are XOR-join tasks)*
3. *for all $p \in P$: $\bullet p = T$ and $p\bullet = X$ (all conditions in P have the same input and output)*
4. *for all $p, q \in P$: $rem_1^{\leftarrow}(p) = rem_1^{\leftarrow}(q)$ (all conditions in P are reset by the same tasks)*

Construction of N_2:

5. $C_2 = (C_1 \setminus P) \cup \{c\}$
6. $T_2 = T_1$
7. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (T \times \{c\}) \cup (\{c\} \times X)$
8. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_1\} \oplus \{(z, (rem_1(z) \cap (C_2 \cup T_2)) \cup \{c\}) | z \in rem_1^{\leftarrow}(p) \wedge p \in P\}$
9. $split_2 = split_1$
10. $join_2 = join_1$

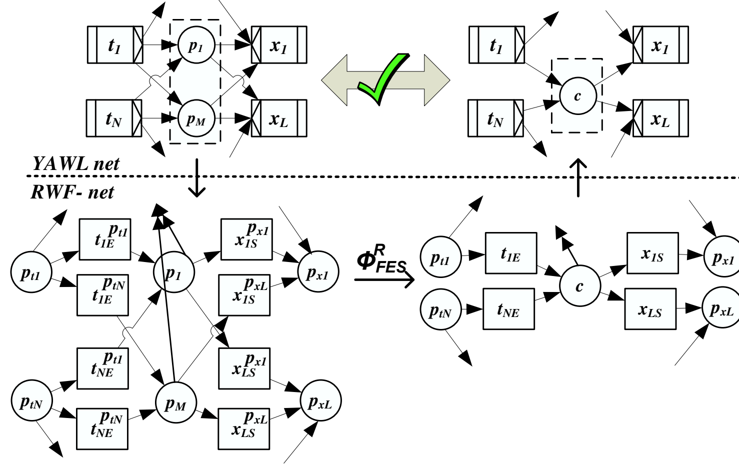

Fig. 7. Fusion of Alternative Conditions Rule for YAWL nets: ϕ^{Y}_{FAP}

Theorem 3 **(The ϕ^{Y}_{FAP} rule is soundness preserving).** *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^{Y}_{\mathrm{FAP}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 7 visualises the ϕ^{Y}_{FAP} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.5 Fusion of series tasks

The *Fusion of Series Tasks Rule for YAWL nets* (ϕ^{Y}_{FST}) allows for the merging of two sequential tasks t and u in a YAWL net that have only one condition p in between them into a single task v. The ϕ^{Y}_{FST} rule makes use of the ϕ^{R}_{FST} rule and the ϕ^{R}_{FSP} rule for RWF-nets. The application requirements are similar to those for the respective rules except that we refer to tasks and conditions instead of transitions and places. In addition, we require that tasks t and u are AND-split tasks and tasks t and u are cancelled by the same set of transitions that remove tokens from condition p (if any). It is possible that task t can reset certain places. Figure 8 visualises the ϕ^{Y}_{FST} rule. Two tasks t and u are merged into a new task v and condition p is abstracted from the reduced net. Task v also takes on the reset arcs of task t (if any).

Definition 22 **(Fusion of Series Tasks Rule: ϕ^{Y}_{FST}).** *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^{Y}_{\mathrm{FST}}$ if there exists tasks $t, u \in T_1$, a condition $p \in C_1$ and a task $v \in T_2 \setminus (T_1 \cup C_1)$ such that:*

Conditions on N_1:

1. $\{t\} = \bullet p$ *(t is the only input of p)*
2. $\{u\} = p\bullet$ *(u is the only output of p)*
3. $\{p\} = \bullet u$ *(p is the only input of u)*
4. $rem_1^{\leftarrow}(p) = rem_1^{\leftarrow}(t) = rem_1^{\leftarrow}(u) = \varnothing$ *(t, u and p are not reset by any task)*
5. $rem_1(u) = \varnothing$ *(u does not reset)*
6. *for all* $c \in u\bullet : rem_1^{\leftarrow}(c) = \varnothing$ *(outputs of u are not reset by any task)*
7. $split_1(t) = split_1(u) = AND$ *(both t and u are AND-split tasks)*

Construction of N_2:

8. $C_2 = C_1 \setminus \{p\}$
9. $T_2 = (T_1 \setminus \{t, u\}) \cup \{v\}$
10. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (\overset{N_1}{\bullet} t \times \{v\}) \cup (\{v\} \times u \overset{N_1}{\bullet}) \cup (\{v\} \times (t \overset{N_1}{\bullet} \setminus \{p\}))$
11. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \cup \{(v, rem_1(t))\}$
12. $split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, AND)\}$
13. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, AND)\}$

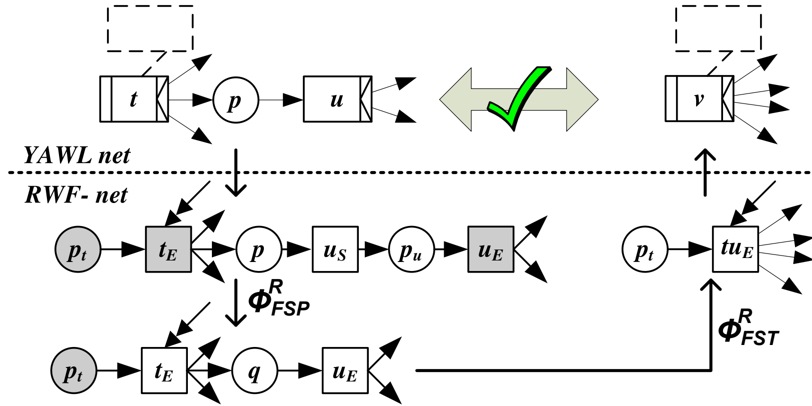

Fig. 8. Fusion of Series Tasks Rule for YAWL nets: ϕ^Y_{FST}

Theorem 4 (The ϕ^Y_{FST} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^Y_{\mathrm{FST}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 8 visualises the ϕ^Y_{FST} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.6 Fusion of parallel tasks

The *Fusion of Parallel Tasks Rule for YAWL nets* (ϕ^{Y}_{FPT}) allows for the merging of two or more identical tasks in a YAWL net into a single task. Two tasks are *identical* if both have the same input set and output set, the same split behaviour and join behaviour, empty cancellation regions, and are not cancelled by any other tasks. The ϕ^{Y}_{FPT} rule makes use of the ϕ^{R}_{FPT} rule and the ϕ^{R}_{FST} rule for RWF-nets. The application requirements are similar to those for the respective rules except that we refer to tasks and conditions instead of transitions and places. In addition, we require that all tasks AND-split and AND-join tasks. Figure 9 visualises the ϕ^{Y}_{FPT} rule. All tasks which satisfy the requirements are merged into a new task v in the reduced net. Task v takes on all characteristics of one of these tasks in the original net.

Definition 23 (Fusion of Parallel Tasks Rule: ϕ^{Y}_{FPT}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^{Y}_{\mathrm{FPT}}$ if there exists tasks $T \subseteq T_1$ where $|T| > 1$, conditions $P, X \subseteq C_1$ and a task $v \in T_2 \setminus (T_1 \cup C_1)$ such that:*

Conditions on N_1:

1. *for all $t \in T : \bullet t = P \wedge join_1(t) = AND$ (all tasks in T have the same input and AND-join structure)*
2. *for all $t \in T : t\bullet = X \wedge split_1(t) = AND$ (all tasks in T have the same output and AND-split structure)*
3. *for all $t \in T : rem_1(t) = \varnothing$ (all tasks in T do not reset)*
4. *for all $t \in T : rem_1^{\leftarrow}(t) = \varnothing$(all tasks in T are not reset by any tasks)*

Construction of N_2:

5. $C_2 = C_1$
6. $T_2 = (T_1 \setminus T) \cup \{v\}$
7. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (P \times \{v\}) \cup (\{v\} \times X)$
8. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \cup \{(v, \varnothing)\}$
9. $split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, AND)\}$
10. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, AND)\}$

Theorem 5 (The ϕ^{Y}_{FPT} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^{Y}_{\mathrm{FPT}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 9 visualises the ϕ^{Y}_{FPT} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

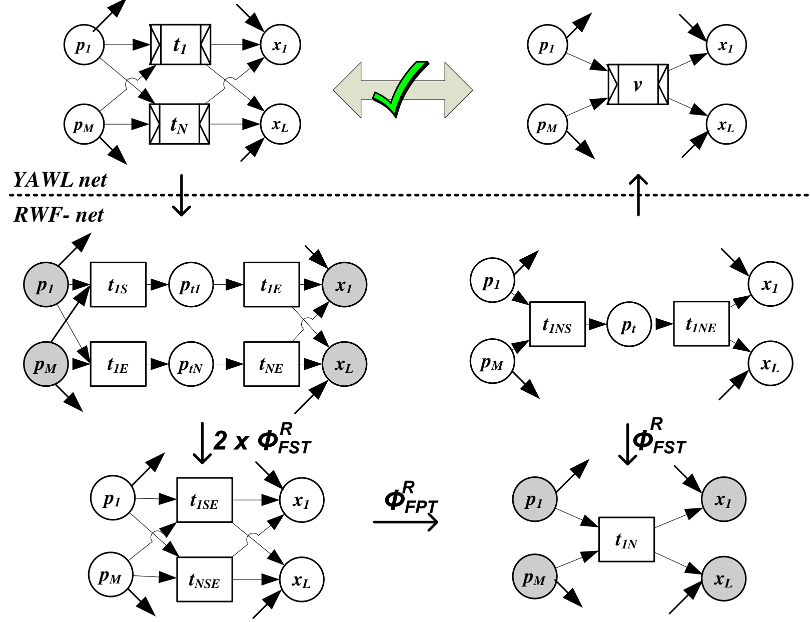

Fig. 9. Fusion of Parallel Tasks Rule for YAWL nets: ϕ^Y_{FPT}

3.7 Fusion of alternative tasks

The *Fusion of Alternative Tasks Rule for YAWL nets* (ϕ^Y_{FAT}) also allows for the merging of two or more identical tasks in a YAWL net into a single task. The only difference between the ϕ^Y_{FPT} rule and the ϕ^Y_{FAT} rule is that tasks in the ϕ^Y_{FAT} rule are XOR-split and XOR-join tasks and tasks in the ϕ^Y_{FPT} rule are AND-split and AND-join tasks. The ϕ^Y_{FAT} rule makes use of the ϕ^R_{FES} rule for RWF-nets. The application requirements are similar to those for the ϕ^R_{FES} rule except that we refer to tasks and conditions instead of transitions and places. In addition, we require that all tasks are XOR-split and XOR-join tasks and they are cancelled by the same set of tasks. Figure 10 visualises the ϕ^Y_{FAT} rule. All tasks which satisfy the application requirements are merged into a new task v in the reduced net. Task v takes on all characteristics of one of the tasks in the original net.

Definition 24 (Fusion of Alternative Tasks Rule: ϕ^Y_{FAT}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^Y_{\mathrm{FAT}}$ if there exists tasks $T \subseteq T_1$ where $|T| > 1$, conditions $P, X \subseteq C_1$ and a task $v \in T_2 \setminus (T_1 \cup C_1)$ such that:*

Conditions on N_1:

1. *for all $t \in T : \bullet t = P \wedge join_1(t) = XOR$ (all tasks in T have the same input and XOR-join structure)*

2. *for all* $t \in T : t\bullet = X \wedge split_1(t) = XOR$ *(all tasks in* T *have the same output and XOR-split structure)*
3. *for all* $tx, ty \in T : rem_1(tx) = rem_1(ty)$ *(all tasks in* T *reset the same tasks and conditions)*
4. *for all* $tx, ty \in T : rem_1^{\leftarrow}(tx) = rem_1^{\leftarrow}(ty)$ *(all tasks in* T *are reset by the same tasks)*
5. *for all* $t \in T : rem_1(t) \cap T = \varnothing$ *(all tasks in* T *should not reset themselves)*

Construction of N_2*:*

6. $C_2 = C_1$
7. $T_2 = (T_1 \setminus T) \cup \{v\}$
8. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (P \times \{v\}) \cup (\{v\} \times X)$
9. $rem_2 = (\{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \cup \{(v, (rem_1(t) \cap (C_2 \cup T_2)))\}) \oplus \{(z, (rem_1(z) \cap (C_2 \cup T_2)) \cup \{v\}) | z \in T_2 \wedge t \in T \wedge z \in rem_1^{\leftarrow}(t)\}$
10. $split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, XOR)\}$
11. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, XOR)\}$

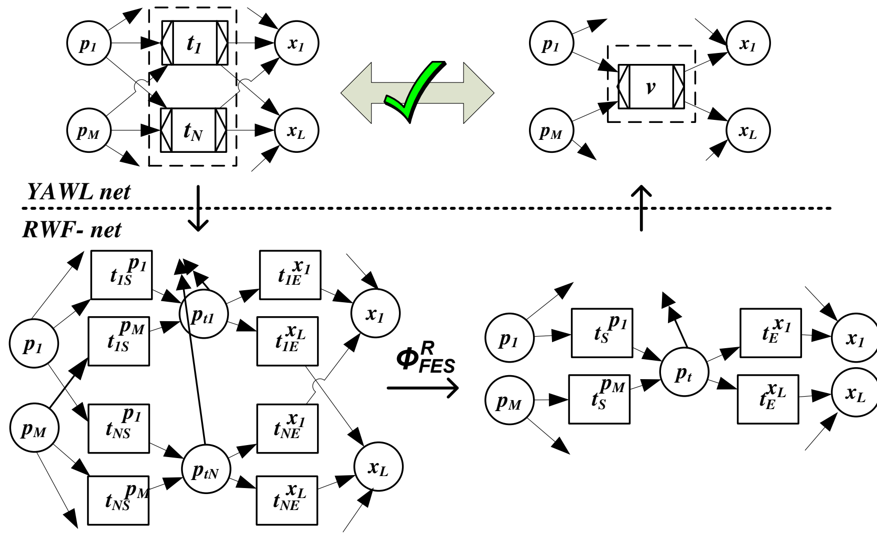

Fig. 10. Fusion of Alternative Tasks Rule for YAWL nets: ϕ^{Y}_{FAT}

Theorem 6 (The ϕ^{Y}_{FAT} **rule is soundness preserving).** *Let* N_1 *and* N_2 *be two eYAWL-nets without OR-joins such that* $(N_1, N_2) \in \phi^{Y}_{\mathrm{FAT}}$*.* N_1 *is sound iff* N_2 *is sound.*

Proof By construction. Figure 10 visualises the ϕ^{Y}_{FAT} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.8 Elimination of self-loop tasks

The *Elimination of Self-Loop Tasks Rule for YAWL nets* (ϕ^{Y}_{ELT}) allows removal of a self-loop task in a YAWL net. The ϕ^{Y}_{ELT} rule makes use of the ϕ^{R}_{FST} and the ϕ^{R}_{ELT} rule for RWF-nets. The application requirements are similar to those for the respective rules except that we refer to tasks and conditions instead of transitions and places. In addition, we require that t and p are not part of any cancellation region. Figure 11 visualises the ϕ^{Y}_{ELT} rule. Task t and associated arcs from t to p are abstracted in the reduced net.

Definition 25 (Elimination of Self-Loop Tasks Rule: ϕ^{Y}_{ELT}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^{Y}_{\mathrm{ELT}}$ if there exists a task $t \in T_1$ and a condition $p \in C_1 \cap C_2$ such that:*

Conditions on N_1:

1. $\bullet t = t\bullet = \{p\}$ *(p is the only input and output of t)*
2. $rem_1(t) = \varnothing$ *(t does not reset)*
3. $\overleftarrow{rem_1}(t) = \varnothing$ *(t is not reset by any task)*
4. $\overleftarrow{rem_1}(p) = \varnothing$ *(p is not reset by any task)*

Construction of N_2:

2. $C_2 = C_1$
3. $T_2 = T_1 \setminus \{t\}$
4. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2)))$
5. $rem_2 = \{(z, rem_1(z) | z \in T_2\}$
6. $split_2 = \{(z, split_1(z)) | z \in T_2\}$
7. $join_2 = \{(z, join_1(z)) | z \in T_2\}$

Theorem 7 (The ϕ^{Y}_{ELT} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^{Y}_{\mathrm{ELT}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 11 visualises the ϕ^{Y}_{ELT} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

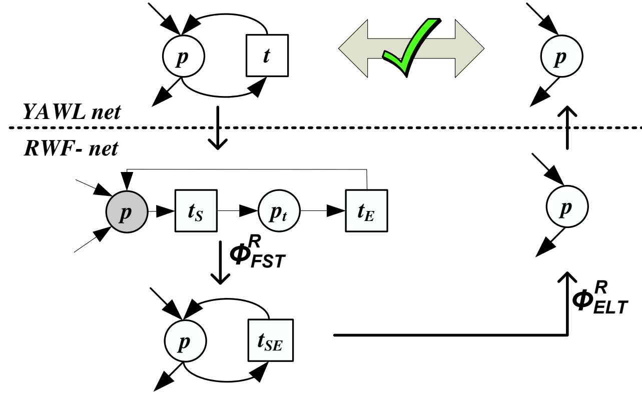

Fig. 11. Elimination of Self-Loop Tasks Rule for YAWL nets: ϕ^Y_{ELT}

3.9 Elimination of self-loop conditions

The *Elimination of Self-Loop Conditions Rule for YAWL nets* (ϕ^Y_{ELP}) allows removal of a self-loop condition in a YAWL net. The ϕ^Y_{ELP} rule makes use of the ϕ^R_{FSP} and the ϕ^R_{ELT} rule for RWF-nets. The application requirements are similar to those for the respective rules except that we refer to tasks and conditions instead of transitions and places. Figure 12 visualises the ϕ^Y_{ELP} rule. Condition x and associated arcs from x to t are abstracted in the reduced net.

Definition 26 (Elimination of Self-Loop Conditions Rule: ϕ^Y_{ELP}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi^Y_{\mathrm{ELP}}$ if there exists a task $t \in T_1 \cap T_2$ and a condition $x \in C_1$ such that:*

Conditions on N_1:

1. $\bullet x = x \bullet = \{t\}$ *(t is the only input and output of x)*
2. $split_1(t) = join_1(t) = XOR$ *(t has XOR-split and XOR-join structure)*
3. $rem_1(t) = \varnothing$ *(t does not reset)*
4. *for all* $p \in \bullet t \cup t \bullet : rem_1^{\leftarrow}(t) = rem_1^{\leftarrow}(p)$ *(t and all its input conditions and output conditions are reset by the same tasks)*

Construction of N_2:

2. $C_2 = C_1 \setminus \{x\}$
3. $T_2 = T_1$
4. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2)))$
5. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2\}$
6. $split_2 = split_1$
7. $join_2 = join_2$

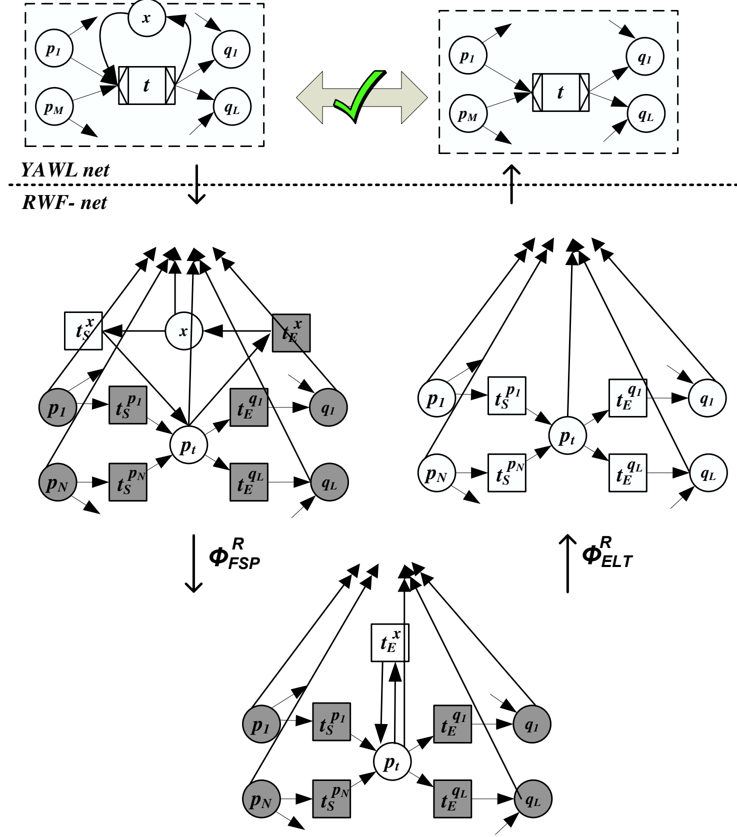

Fig. 12. Elimination of Self-Loop Conditions Rule for YAWL nets: ϕ^Y_{ELP}

Theorem 8 (The ϕ^Y_{ELP} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi^Y_{\mathrm{ELP}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 12 visualises the ϕ^Y_{ELP} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.10 Fusion of AND-split and AND-join tasks

The *Fusion of AND-split and AND-join tasks for YAWL nets* (ϕ_{FAND}) allows for the merging of structured AND-split and AND-join tasks into a single task in a YAWL net. The ϕ_{FAND} rule makes use of the ϕ^R_{FPP} rule and the ϕ^R_{FSP} rule for RWF-nets. In addition, we require that tasks t, u, and conditions $p_1, ..., p_N$ are not part of any cancellation region nor do both t and u reset any places. Figure 13 visualises the ϕ_{FAND}

rule. Tasks t and u have been merged into a new task v and v takes on the split behaviour of u and the join behaviour of t.

Definition 27 (Fusion of AND-split and AND-join Rule: ϕ_{FAND}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi_{\mathrm{FAND}}$ if there exists two tasks $t, u \in T_1$, a task $v \in T_2 \setminus (T_1 \cup C_1)$, and conditions $P \subseteq C_1$ where $|P| \geq 1$ such that:*

Conditions on N_1:

1. *for all $p \in P : \bullet p = \{t\} \wedge p\bullet = \{u\}$ (all conditions in P have input t and output u)*
2. *$split_1(t) = join_1(u) = AND$ (t is an AND-split task and u is an AND-join task)*
3. *for all $p \in P : rem_1^{\leftarrow}(p) = \varnothing$ (all conditions in P are not reset by any task)*
4. *$rem_1^{\leftarrow}(t) = rem_1^{\leftarrow}(u) = \varnothing$ (t and u are not reset by any task)*
5. *$rem_1(t) = rem_1(u) = \varnothing$ (t and u do not reset)*

Construction of N_2:

6. *$C_2 = C_1 \setminus P$*
7. *$T_2 = (T_1 \setminus \{t, u\}) \cup \{v\}$*
8. *$F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (\overset{N_1}{\bullet} t \times \{v\}) \cup (\{v\} \times u \overset{N_1}{\bullet})$*
9. *$rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \cup \{(v, \varnothing)\}$*
10. *$split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, split_1(u))\}$*
11. *$join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup (\{(v, join_1(t))\}$*

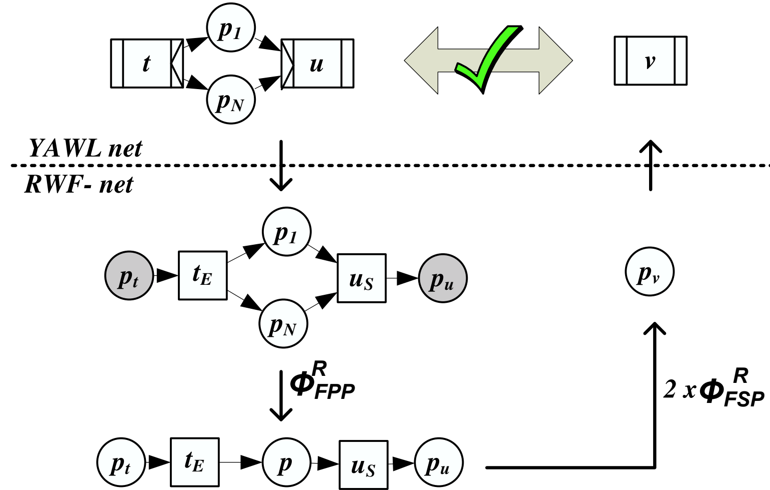

Fig. 13. Fusion of AND-split and AND-join tasks for YAWL nets: ϕ_{FAND}

Theorem 9 (The ϕ_{FAND} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi_{\mathrm{FAND}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 13 visualises the ϕ_{FAND} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

3.11 Fusion of XOR-split and XOR-join tasks

The *Fusion of XOR-split and XOR-join tasks for YAWL nets* (ϕ_{FXOR}) allows for the merging of structured XOR-split and XOR-join tasks into a single task in a YAWL net. The ϕ_{FXOR} rule makes use of the ϕ^R_{FSP} rule and the ϕ^R_{FST} rule for RWF-nets. In addition, we require that tasks t, u, and conditions $p_1, ..., p_N$ are not part of any cancellation region. Figure 14 visualises the ϕ_{FXOR} rule. Tasks t and u have been merged into a new task v and v takes on the split behaviour of u and the join behaviour of t.

Definition 28 (Fusion of XOR-split and XOR-join tasks Rule: ϕ_{FXOR}). *Let N_1 and N_2 be two eYAWL-nets without OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi_{\mathrm{FAND}}$ if there exists two tasks $t, u \in T_1$, a task $v \in T_2 \setminus (T_1 \cup C_1)$, and conditions $P \subseteq C_1$ where $|P| \geq 1$ such that:*

Conditions on N_1:

1. *for all $p \in P : \bullet p = \{t\} \wedge p\bullet = \{u\}$ (all conditions in P have input t and output u)*
2. *$split_1(t) = join_1(u) = XOR$ (t is an XOR-split task and u is an XOR-join task)*
3. *for all $p \in P : \overleftarrow{rem_1}(p) = \varnothing$ (all conditions in P are not reset by any task)*
4. *$\overleftarrow{rem_1}(t) = \overleftarrow{rem_1}(u) = \varnothing$ (t and u are not reset by any task)*
5. *$rem_1(t) = rem_1(u) = \varnothing$ (t and u do not reset)*

Construction of N_2:

6. *$C_2 = C_1 \setminus P$*
7. *$T_2 = (T_1 \setminus \{t, u\}) \cup \{v\}$*
8. *$F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (\overset{N_1}{\bullet} t \times \{v\}) \cup (\{v\} \times u \overset{N_1}{\bullet})$*
9. *$rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \cup \{(v, \varnothing)\}$*
10. *$split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, split_1(u))\}$*
11. *$join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup (\{(v, join_1(t))\}$*

Theorem 10 (The ϕ_{FXOR} rule is soundness preserving). *Let N_1 and N_2 be two eYAWL-nets without OR-joins such that $(N_1, N_2) \in \phi_{\mathrm{FXOR}}$. N_1 is sound iff N_2 is sound.*

Proof By construction. Figure 14 visualises the ϕ_{FXOR} rule and sketches the proof of this rule. The proof is given in terms of a number of transformations to and from reset nets that are soundness preserving. ∎

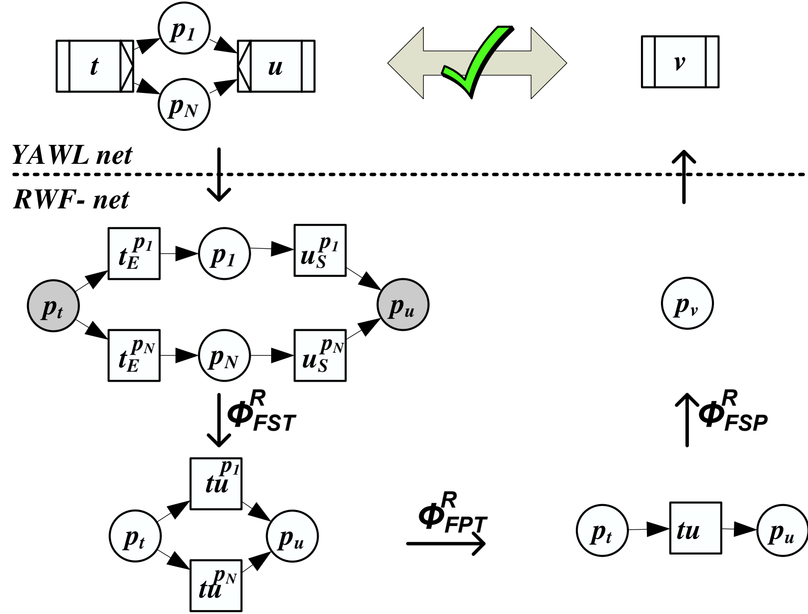

Fig. 14. Fusion of XOR-split and XOR-join tasks for YAWL nets: ϕ_{FXOR}

4 Reduction rules for nets with cancellation regions and OR-joins

A YAWL net with OR-joins requires special attention as the decision to enable an OR-join task cannot be made locally as seen in [20]. As verification techniques for YAWL nets with OR-joins utilise reachability analysis using the YAWL semantics as seen in [19] and hence, state space explosion is a real concern. Our objective is to identify possible reduction rules for YAWL nets with OR-joins that could be used under certain context assumptions so that verification can be performed more efficiently.

To achieve this, we propose to apply reduction rules defined for nets without OR-joins as presented in Section 3 to those parts of the net without OR-joins. These reduction rules have been defined for tasks without OR-join behaviour. However, the reset net transformations still hold and therefore, the reduction rules also apply. In addition, these reduction rules do not affect the OR-join semantics in the net. We now demonstrate this concept using an example. Figure 15 shows a YAWL net with an OR-join task G. First, consider task B and its associated input and output condition. It is clear that the ϕ^{Y}_{FSP} rule could be used to abstract task B if we are dealing with a net without OR-joins. The same is true for tasks C, E, and F. Applying the ϕ^{Y}_{FSP} rule to the net would result in the (top) reduced net as shown in Figure 16. In this reduced net, it is now possible to merge tasks A and D into a new task X using the ϕ_{FAND} rule. Note that task X takes on the split behaviour of D and the join behaviour of A. The bottom net in Figure 16 shows the resulting net after applying ϕ_{FAND} rule to the top net. When the original net in Figure 15 is compared with the reduced net in Figure 16, it is clear

that the enabling requirements for the OR-join task G are the same. As these reduction rules do not reduce an OR-join task, they do not invalidate the OR-join semantics of task G.

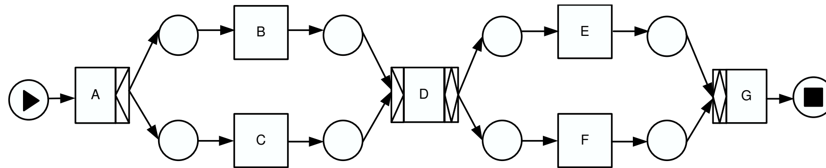

Fig. 15. A YAWL net with an OR-join task G

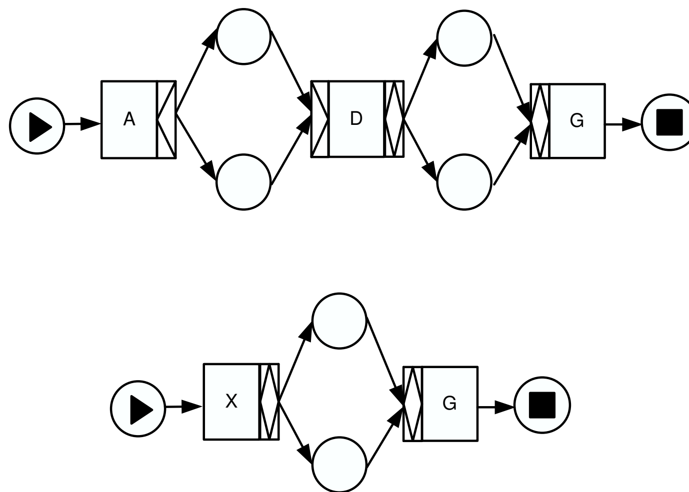

Fig. 16. Reduced nets after applying the ϕ^{Y}_{FSP} rule and the ϕ_{FAND} rule to Figure 15

From the above discussion, we can see that it is possible to apply YAWL reduction rules to those parts of a YAWL net that do not have any OR-joins. Next, we present two additional reduction rules directly related to the OR-join construct: the ϕ_{FOR} rule and the ϕ_{FIE} rule. Both reduction rules presented here are provided under the context assumption of safeness. A condition is *safe* if it is not possible to have more than one token at a time. This is especially important for conditions which are on the path to an OR-join task. Figure 17 shows a net with conditions c_2, c_3, and c_4 that could contain more than one token at a time. When task A fires, a marking $c_1 + c_2$ is reached. From that marking, A can fire again and again, put more tokens into place c_2. As a result, it is possible for conditions $c3$ and $c4$ to have multiple tokens as well. The OR-join task C will wait for both input conditions $c3$ and $c4$ to be marked before enabling. This means

that task C is only enabled after multiple firings of task B. By making the assumption of safeness, we can ensure that if task B is enabled and fired and tokens are put into the input conditions of task C, then more tokens cannot arrive at C without firing task B again. Hence, the OR-join enabling rule can be localised in this particular circumstance. Next, we formalise this concept and present two reduction rules specific for the OR-join construct.

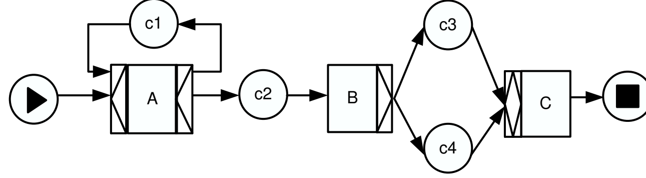

Fig. 17. An example of a YAWL net with unsafe conditions

4.1 Fusion of an OR-join and another task

The *Fusion of an OR-join and another task for YAWL nets* (ϕ_{FOR}) rule enables certain OR-join tasks to be abstracted from the net under the context assumption of safeness. Figure 18 visualises the ϕ_{FOR} rule. The rule requires that all inputs to an OR-join task are from one task (regardless of the split behaviour of that task). In addition, tasks t and u are not allowed to have cancellation regions and all output places for t as well as tasks t and u are part of the same cancellation regions (if any). If all requirements are satisfied, tasks t and u are merged into a new task v in the reduced net. Task v takes on the split behaviour of u and the join behaviour of t.

Definition 29 (Fusion of an OR-join and another task Rule: ϕ_{FOR}).
Let N_1 and N_2 be two eYAWL-nets with OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi_{\mathrm{FOR}}$ if there exists two tasks $t, u \in T_1$, a task $v \in T_2 \setminus (T_1 \cup C_1)$ and conditions $Q_1 \subseteq C_1$ such that:

Conditions on N_1:

1. *for all $p \in Q_1 : \bullet p = \{t\} \wedge p\bullet = \{u\}$ (t is the only input and u is the only output of all places in Q_1)*
2. *$t\bullet = \bullet u$ (output set of t and input set of u are identical)*
3. *$join_1(u) = OR$ (u is an OR-join task)*
4. *for all $p \in Q_1 : rem_1^{\leftarrow}(p) = rem_1^{\leftarrow}(t) = rem_1^{\leftarrow}(u)$ (t, u, and Q_1 are reset by the same tasks)*
5. *$rem_1(t) = rem_1(u) = \varnothing$ (t and u do not reset)*

Construction of N_2:

6. $C_2 = C_1 \setminus Q_1$
7. $T_2 = (T_1 \setminus \{t, u\}) \cup \{v\}$
8. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup (\bullet t \times \{v\}) \cup (\{v\} \times u\bullet)$
9. $rem_2 = \{(z, rem_1(z) \cap (C_2 \cup T_2)) | z \in T_2 \cap T_1\} \oplus \{(z, (rem_1(z) \cap (C_2 \cup T_2)) \cup \{v\}) | z \in rem_1^{\leftarrow}(t) \cap T_2\}$
10. $split_2 = \{(z, split_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, split_1(u))\}$
11. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, join_1(t))\}$

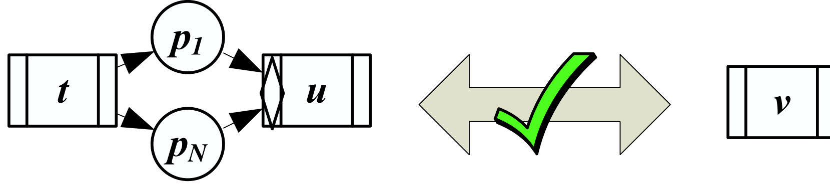

Fig. 18. Fusion of an OR-join and another task Rule for YAWL nets: ϕ_{FOR}

Assume that task t is enabled at M and $M \rightarrow M'$. If t is an AND-split task, M' marks all output conditions of t which are also input conditions of u and u is enabled. If t is an XOR-split task, M' marks a subset of conditions in $\bullet u$. The unmarked conditions in $\bullet u$ cannot be marked again without potentially adding more tokens into already marked conditions in $\bullet u$ and thus making this condition unsafe. As a result, no more tokens can be put into $\bullet u$ under the safeness assumption and u is also enabled. The same is true when the split type of t is an OR-split task. Under the context assumption of safeness, we can see that OR-join task u will fire once for every firing of t as $t\bullet = \bullet u$. Hence, we can conclude that if there is a marking M that enables t, the reachable marking M' will enable OR-join task u. If there is no marking that enables t, then both t and u are not enabled.

In the reduced net, tasks t and u are replaced by task v. The ϕ_{FOR} rule requires that inputs for t and v are the same, outputs for u and v are the same, the join behaviours of t and v are the same and the split behaviours of u and v are the same. Therefore, a marking that enables t also enables v. After the sequence tu fires, it puts tokens into $u\bullet$ depending on the split behaviour of u. As v has the same split behaviour as u, the resulting marking after firing the sequence tu is also the same as the marking after firing v. If it is not possible to enable t in N_1, it is also not possible to enable v in N_2 and hence, the behaviour is still the same.

The ϕ_{FOR} rule is very useful as it can potentially transform the net into one without OR-joins. It is then possible to perform verification of the resulting reduced YAWL net without OR-joins using reset net analysis [19]. However, the rule is quite restrictive because it requires that all output arcs from a task to go into an OR-join and the OR-join could not have additional input arcs from any other tasks. As a result, the ϕ_{FOR} rule is not applicable to OR-joins with input arcs from multiple tasks. Hence, we propose a

weaker rule that is intended to remove arcs and not the OR-join. Even though the rule does not remove OR-joins, it will help reduce the complexity of the model.

4.2 Fusion of incoming edges to an OR-join

The *Fusion of Incoming Edges to an OR-join for YAWL nets* ($\phi_{\rm FIE}$) rule allows for the merging of two or more conditions that have the same input task and the same output task (an OR-join) into a single condition. Also, these conditions cannot be in any cancellation region. Figure 19 visualises the $\phi_{\rm FIE}$ Rule. The $\phi_{\rm FIE}$ rule is a weaker rule compared to the $\phi_{\rm FOR}$ rule and it could also be applied to nets that can be reduced by the $\phi_{\rm FOR}$ rule.

Definition 30 (Fusion of Incoming Edges to an OR-join Rule: $\phi_{\rm FIE}$).
Let N_1 and N_2 be two eYAWL-nets with OR-joins, where $N_1 = (C_1, i, o, T_1, F_1, split_1, join_1, rem_1)$ and $N_2 = (C_2, i, o, T_2, F_2, split_2, join_2, rem_2)$. $(N_1, N_2) \in \phi_{\rm FIE}$ if there exists two tasks $t, u \in T_1 \cap T_2$, conditions $Q \subseteq C_1$ where $|Q| \geq 2$, and a condition $p \in C_2 \setminus (C_1 \cup T_1)$ such that:

Conditions on N_1:

1. *for all $p \in Q_1 : \bullet p = \{t\} \wedge p\bullet = \{u\}$ (t is the only input and u is the only output of all places in Q_1)*
2. *$join_1(u) = OR$ (u is an OR-join task)*
3. *for all $p \in Q_1 : rem_1^{\leftarrow}(p) = \varnothing$ (conditions in Q_1 are not reset by any task)*

Construction of N_2:

4. *$C_2 = (C_1 \setminus Q_1) \cup \{p\}$*
5. *$T_2 = T_1$*
6. *$F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup \{(t, p), (p, u)\}$*
7. *$rem_2 = rem_1$*
8. *$split_2 = split_1$*
9. *$join_2 = join_1$*

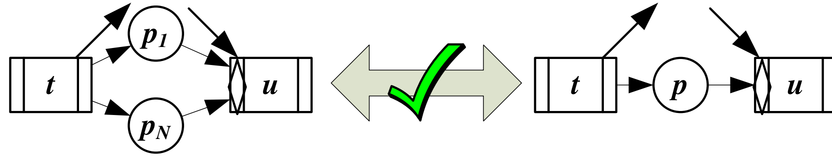

Fig. 19. Fusion of Incoming Edges to an OR-join Rule for YAWL nets: $\phi_{\rm FIE}$

As conditions in $Q_1 = \{p_1, ..., p_N\}$ and p have the same input task t and output task u, if some subset of places in Q_1 is marked at a marking in N_1, p is also marked at

the corresponding marking in N_2. Under the assumption of safeness, if some conditions in Q_1 cannot get marked, they cannot get marked later as this would enable currently marked places to be marked twice, which is not safe. If p cannot be marked, then conditions in $p_1, ..., p_N$ cannot be marked. Therefore, the OR-join enabling behaviour of u is identical in both nets regardless of whether there is only one condition p or multiple conditions Q_1.

Remark: Please note that reachability analysis needs to be carried out to determine whether the safeness assumption holds for a net. Currently, the implementation assumes safeness and no checking is done before applying the two reduction rules (ϕ_{FOR} and ϕ_{FIE}).

5 Implementation

Reduction rules presented in this paper have been implemented in the YAWL editor. The algorithm requires two input parameters: a net in the form of an XML file and the name of a particular reduction rule. The rule is then applied exhaustively to the net and if it is possible to reduce the net, the reduced net is returned as an XML file, otherwise null is returned. At the moment, the reduction rules cannot be invoked individually from the editor. The tests are carried out by first drawing the model in the editor and then exporting it as an XML file. If a rule reduces the net then the net is exported back as another XML file. Otherwise, a message is displayed to indicate that the net cannot be reduced by the rule. The resulting XML file is then imported back into the editor so that the reduced net can be displayed on the screen.

5.1 YAWL reduction rules

We first demonstrate how reduction rules can be applied to YAWL nets without OR-joins using the example in Figure 20. The total number of elements in the net is 27. Also note that task F has a cancellation region with five elements: c_1, c_2, c_3, C, and D. The editor displays F in grey and elements in the cancellation region in red. To make the cancellation region more obvious, the screenshot has been modified by adding a dotted line around the cancellation elements. First, the ϕ^{Y}_{FSP} rule is applied recursively to the net until the net cannot be reduced further using this rule. This results in the reduction of elements from 27 to 15 and the reduced net is given in Figure 21. Note that the sequence of tasks and conditions between G and L has been abstracted using this rule and replaced by two conditions identified as null-135 and null-138. The same applies for conditions between tasks B and H which are now replaced by a new condition (null-136). As all elements in the cancellation region of F are cancelled by the same task, the cancellation region of task F now contains only one condition (the condition between tasks A and H - null-145). As the YAWL editor does not display conditions without names or labels, tasks B and F are shown as having a direct connection even though there is a condition in between them. The same holds for the condition between tasks A and B and also between tasks F and G. Therefore, there are only 12 elements shown in the diagram even though the net has 15 elements. Next, the ϕ_{FXOR} rule is applied to abstract the XOR-split task G and the XOR-join task L and the resulting net is shown

in Figure 22. Finally, the ϕ^{R}_{FST} rule is applied to abstract task G and this results in the reduced net with 10 elements shown in Figure 23.

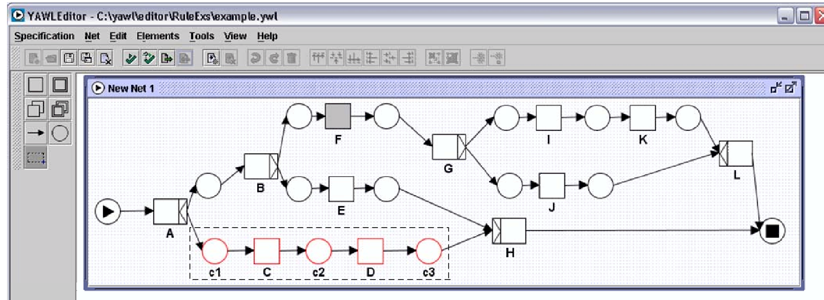

Fig. 20. A YAWL net without OR-joins

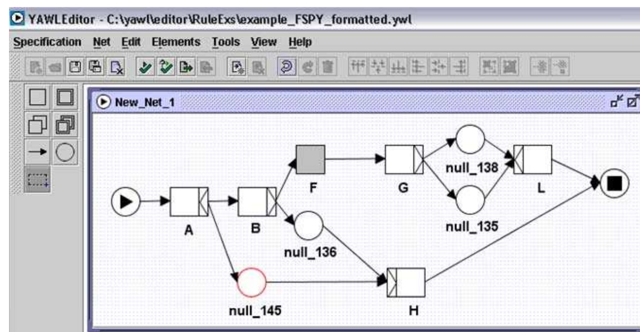

Fig. 21. The reduced net after applying the ϕ^{Y}_{FSP} rule to Figure 20

Next, we demonstrate how the two reduction rules for OR-joins (ϕ_{FIE} and ϕ_{FOR}) can be applied together with other YAWL reduction rules to a net with OR-joins. Figure 24 shows a YAWL net with OR-join tasks D and H. First, the ϕ^{Y}_{FSP} rule is applied to minimise the number of elements in the net and this results in reducing the number of elements from 20 to 12. Next, the ϕ_{FIE} rule is applied to replace multiple conditions between A and D with just one (condition null-12) and also for multiple conditions between E and H with just one(condition null-11). Figure 25 shows two reduced nets for the net in Figure 24. The net on the left is obtained after applying the ϕ^{Y}_{FSP} rule exhaustively to the net in Figure 24 and the net on the right is obtained after applying the ϕ_{FIE} rule to the net on the left, resulting in 10 elements. Figure 26 shows two more reduced nets. The net on the left is obtained by applying the ϕ^{Y}_{FSP} rule to the right net in Figure 25. The net on the right is obtained by applying the ϕ_{FOR} rule to the left net.

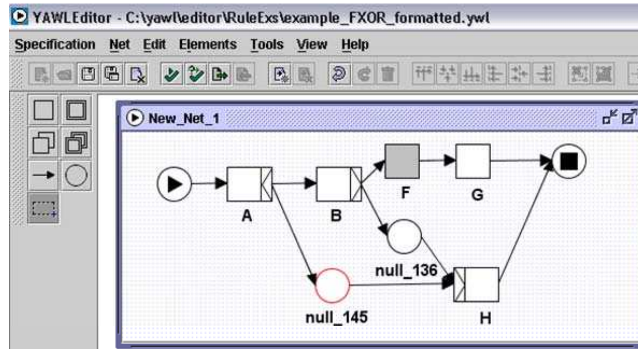

Fig. 22. The reduced net after applying the ϕ_{FXOR} rule to Figure 21

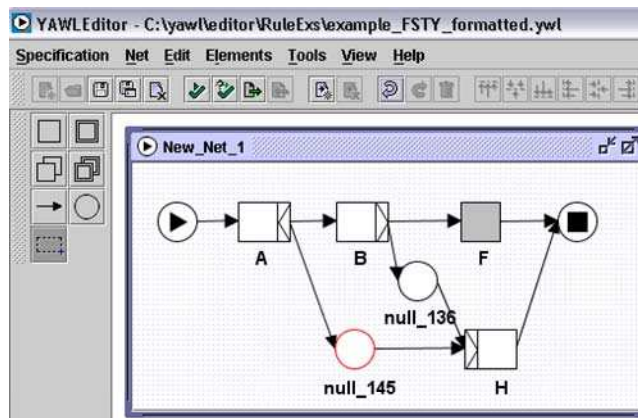

Fig. 23. The reduced net after applying the ϕ^{Y}_{FST} rule to Figure 22

The reduced net on the right has only three elements with one input place, one output place and a task in between, and hence, it is a trivial net.

We have seen how YAWL reduction rules can be used to reduce the number of elements in a net. The same is also true for reset net reduction rules. The reduction rules for RWF-nets presented in [23] have also been implemented in the YAWL editor and they will be used together with verification techniques for YAWL nets without OR-joins.

5.2 Linking reduction rules to verification

As all reduction rules proposed in this paper are soundness preserving, it is possible to perform verification on the reduced nets instead of the original net. This improves the efficiency of the verification process. We propose a three-step process for verification.

1. Reduction rules are applied exhaustively to a net until it cannot be reduced further. For a YAWL net with OR-joins, YAWL reduction rules are applied to obtain a

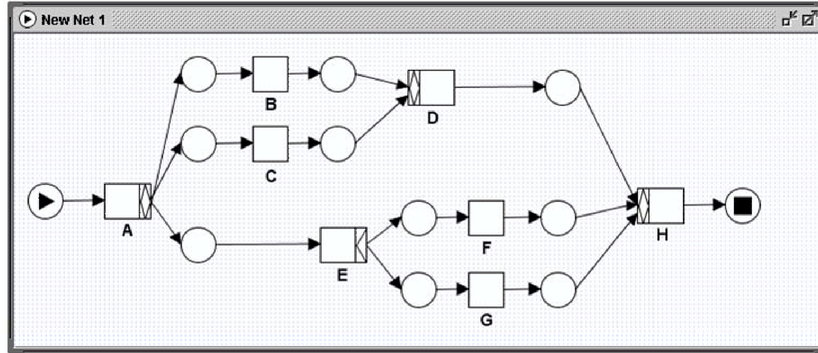

Fig. 24. A YAWL net with OR-joins D and H

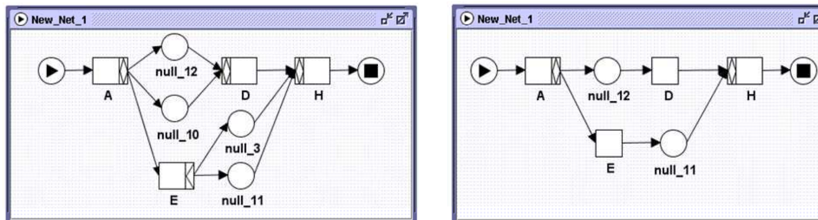

Fig. 25. Reduced nets after applying the ϕ^{Y}_{FSP} rule and the ϕ_{FIE} rule

reduced net. For a YAWL net without OR-joins, the net is first translated to an RWF-net and reset reduction rules are then applied to obtain a reduced RWF-net net for verification. The mappings between different nets are also stored for error reporting.

2. Verification is performed on the reduced net as defined in [19].
3. If there are any warnings to be reported, the element names in the reduced net are mapped back to the names of tasks and conditions in the original net.

Reduction rules together with the three-step approach for verification using reduced nets are implemented in the YAWL editor. For each element in the reduced YAWL net, a mapping to a set of original YAWL elements is kept. For each element in the reduced

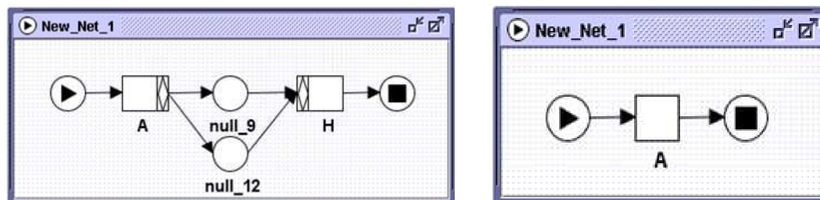

Fig. 26. Reduced nets after applying the ϕ^{Y}_{FSP} rule and the ϕ_{FOR} rule

RWF-net, a mapping to a set of original RWF elements is kept. These mappings are used for reporting error messages. Figures 27 and 28 show the results from the soundness property checks for nets in Figures 20 and 24 using reduction rules.

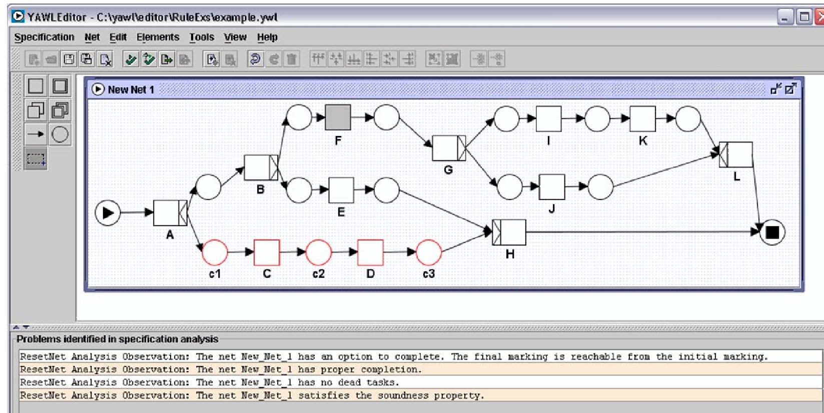

Fig. 27. Soundness property results for the net in Figure 20

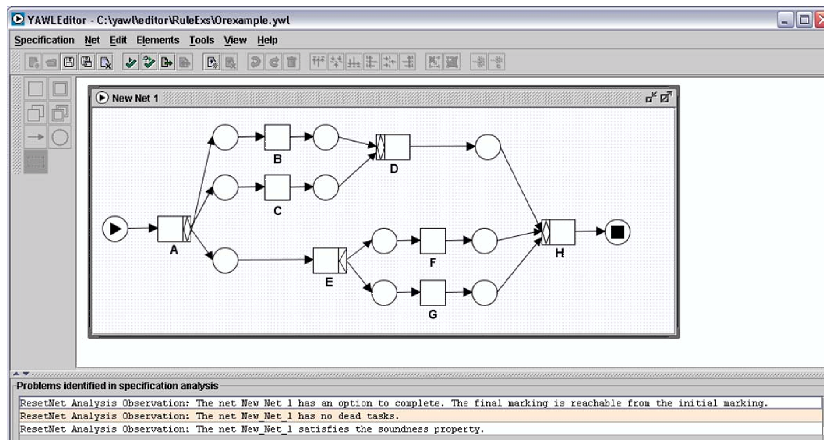

Fig. 28. Soundness property results for the net in Figure 24

6 Related work

Reduction rules have been suggested to be used together with Petri nets for the verification of workflows (cf. Chapter 4 of the book by van der Aalst and van Hee [5]). Six

reduction rules that preserve correctness for EPCs including reduction rules for trivial constructs, simple splits and joins, similar splits and joins, XOR loop and optional OR-loop are proposed [11]. Some reduction rules presented for EPCs such as reduction rules for simple splits and joins and reduction rules for similar splits and joins are related to reduction rules that we have defined for YAWL nets. However, these reduction rules for EPCs do not take cancellation into account and the OR-join construct uses local semantics. In Verbeek's thesis [18], the author proposes reduction rules for WF-nets based on the reduction rules from Murata and Desel and Esparza [13, 10]. We follow a similar approach with a set of reduction rules for workflow nets with cancellation regions and OR-joins using reset nets. The difference is that our approach takes into account possible cancellation regions in workflows.

7 Conclusion

An important correctness notion for a workflow net is the soundness property. A workflow net is sound if it has the option to complete, proper completion, and no dead transitions. Verification can be used to detect whether a net satisfies the soundness property. When a workflow language supports complex constructs such as cancellation and OR-joins, verification becomes time consuming, challenging and sometimes not even possible. In our previous work [22], we proposed a new verification technique for workflows with cancellation and OR-joins using reset nets and reachability analysis. This approach can be used to detect four important structural properties of YAWL nets: the weak soundness property, the soundness property, reducible cancellation regions and convertible OR-joins. The verification feature has been incorporated in the graphical editor of YAWL.

A reduction rule can transform a large net into a smaller and simple net while preserving certain interesting properties and it is usually applied before verification to reduce the complexity and to prevent state space explosion. In our previous work [23], we have presented a set of reduction rules for RWF-nets. In this paper, we have demonstrated how these reduction rules for RWF-nets can be applied to YAWL, a workflow language that provides direct support for cancellation regions and OR-joins. Whave presented a set of reduction rules for YAWL nets with cancellation regions and OR-joins that are soundness preserving. We have also extended the YAWL editor with the ability to reduce YAWL nets based on the results presented in this paper [1]. We also propose two reduction rules for YAWL specifications with OR-joins which hold under the safeness assumption.

References

1. Yawl home page. http://www.yawl.fit.qut.edu.au/ accessed on 21 May 2006.
2. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G.Balbo, editors, *Proceedings of Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, France, 1997. Springer-Verlag.
3. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

4. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri Net-Based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Proceedings of Business Process Management: Models, Techniques and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, page 161. Springer-Verlag, 2000.
5. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods and Systems*. MIT press, Cambridge, MA, 2004.
6. W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An alternative way to analyze workflow graphs. In Anne Banks Pidduck, John Mylopoulos, Carson C. Woo, and M. Tamer Özsu, editors, *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, volume 2348 of *Lecture Notes in Computer Science*, pages 534–552, Toronto, Canada, May 2002. Springer-Verlag.
7. W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In Kurt Jensen, editor, *Proceedings of the Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 560 of DAIMI, pages 1–20, Aarhus, Denmark, August 2002. University of Aarhus.
8. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, June 2005.
9. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.
10. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, United Kingdom, 1995.
11. B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek. Verification of EPCs: Using Reduction rules and Petri Nets. In O.Pastor and J. Falcão e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE 2005)*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386, Porto, Portugal, June 2005. Springer-Verlag.
12. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.
13. T. Murata. Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
14. J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, USA, 1981.
15. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962. In German.
16. W. Reisig and G. Rozenberg. Lectures on Petri Nets I: Basic Models. In W. Reisig and G. Rozenberg, editors, *Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, Berlin, Germany, 1998. Springer–Verlag.
17. W. Reisig and G. Rozenberg. Lectures on Petri Nets II: Basic Models. In W. Reisig and G. Rozenberg, editors, *Advances in Petri Nets*, volume 1492 of *Lecture Notes in Computer Science*, Berlin, Germany, 1998. Springer–Verlag.
18. H.M.W. Verbeek. *Verification of WF-nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, June 2004.
19. M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Verifying workflows with Cancellation Regions and OR-joins: An Approach Based on Reset nets and Reachability Analysis. Technical report BPM-06-12, BPM Center (bpmcenter.org), 2006.
20. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International conference on*

Application and Theory of Petri nets and Other Models of Concurrency, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443, Miami, USA, June 2005. Springer-Verlag.

21. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Synchronisation and Cancellation in Workflows based on Reset Nets. Technical report, submitted to an international journal, jan 2006, Queensland University of Technology, 2006.
22. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Verifying workflows with Cancellation Regions and OR-joins: An Approach Based on Reset nets and Reachability Analysis. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Proceedings of 4th International Conference of Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 389–394, Vienna, Austria, Sep 2006. Springer-Verlag.
23. M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Reduction rules for Reset Workflow Nets. Technical report BPM-06-25, BPM Center (bpmcenter.org), 2006.