Reduction Rules for YAWL Workflows 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]

1.Business Process Management Group, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia.
{*m.wynn,a.terhofstede,d.edmond*}*@qut.edu.au*
2.Department of Mathematics and Computer Science, Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, The Netherlands.
{*h.m.w.verbeek,w.m.p.v.d.aalst*}*@tue.nl*

Abstract. As the need for concepts such as cancellation and OR-joins occurs naturally in business scenarios, comprehensive support in a workflow language is desirable. However, there is a clear trade-off between the expressive power of a language (i.e., introducing complex constructs such as cancellation and OR-joins) and ease of verification. 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. There are a number of different approaches to deal with this complexity. Reducing the size of the workflow, while preserving its essential properties with respect to a particular analysis problem, is one such approach. In this paper, we present a set of reduction rules for workflows with cancellation regions and OR-joins and demonstrate how they can be used to improve the efficiency of verification. Our results are presented in the context of the YAWL workflow language.
Keywords: Verification of process models, soundness property, reduction rules, cancellation, OR-join, reset nets, YAWL.

1 Introduction

Verification is concerned with determining, *in advance*, whether a process model exhibits certain desirable behaviours. By performing this verification at design time, it is possible to identify potential problems, and if so, the model can be modified before it is used for execution. As some systems (e.g., workflow systems) rely on process models for execution of work, careful analysis of process models at design time can greatly improve the reliability of such systems. When dealing with complex business processes (e.g., in the context of a workflow implementation or the configuration of some process-aware information system), it is important but sometimes difficult to determine whether a process contains any errors due to its complexity. Cancellation and OR-joins are two complex features that are common in many business processes. *Cancellation* captures the interference of an activity in the execution of others in certain circumstances. Cancellation can be triggered by either a customer request (e.g., a customer wishes to withdraw a credit card application) or by exceptions (e.g., an order cannot be processed due to insufficient stock level). In general, cancellation results in one of two outcomes: disabling some scheduled activities or stopping currently running activities. The complicating factor is that due to concurrency issues, the cancellation action may or may not result in cancelling certain activities, i.e., the process may be in a state before or after the part that is supposed to be cancelled. This can introduce deadlocks

(the state where a business process is stuck and cannot proceed). 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 one or 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 selected only one product, no synchronisation is required before payment. The OR-join in its general form (see [17]) requires the ability to "wait and see" whether more completion signals can be expected based on the current state of the process.

The presence of cancellation and OR-joins poses new challenges for deciding workflow correctness. A correct workflow is said to satisfy the *soundness* property [1]. Soundness is a composite property, concerned firstly with whether a workflow, when started, can complete. Secondly, the workflow should never have tasks still running when completion is signalled. Thirdly, the workflow should not contain tasks that can never be executed. While there is a substantial body of work in the area of Petri-net based verification of workflows, the results do not easily transfer to languages that use cancellation and OR-joins. The cancellation concept is not directly supported in Petri nets as this would require the ability to directly remove all tokens from a place and its presence has substantial implications for analysis. Similar considerations apply to the OR-join. In our earlier work [18], we proposed a new verification approach to determine the soundness property of such workflows using reset nets, which are Petri nets with reset arcs that can remove tokens from arbitrary places when a transition fires [8]. To determine whether a workflow satisfies the soundness property, we explore the state space and carry out reachability analysis. The upside is that there is now a verification technique for such complex workflows. The downside is that verification can become intractable for large workflows due to a large state space that needs to be investigated. Applying reduction rules before carrying out verification alleviates this problem by cutting down the size of the workflow while preserving the essential properties including soundness.

In this paper, we present a set of soundness-preserving reduction rules for YAWL workflows with cancellation regions and OR-joins. These reduction rules are inspired by the well-known reduction rules for Petri nets [7, 9]. However, most of the classical rules do *not* apply to processes with cancellation and OR-joins. The expressiveness of these constructs invalidates the original rules, i.e., the classical reduction rules need to be adapted and new rules are added. We take Yet Another Workflow Language (YAWL) [3] as the vehicle through which our results are expressed as YAWL provides direct support for cancellation regions and OR-joins. However, the contribution of our work is not limited to the YAWL workflow language as the support for the notions of cancellation and OR-joins can also be found in many of today's process modelling languages. For instance, the Business Process Modelling Notation (BPMN) provides an "OR gateway" and various cancellation constructs. The Business Process Execution Language (BPEL) supports OR-joins (through the "flow" construct) and cancellation. Event-driven Process Chains (EPCs) can model OR-join but not cancellation. The Activity Diagrams of the Unified Modelling Language (UML) do not support OR-join but offer different cancellation behaviours. The support for efficient verification of these constructs is vital. The set of reduction rules proposed in this paper is equally applicable to other languages supporting cancellation and/or OR-joins.

The organisation of the rest of the paper is as follows. Section 2 contains background information on YAWL and reset nets and also discusses a set of reduction rules for reset nets. This is because YAWL nets with cancellation regions are formally mapped to reset nets and we make use of the reset net reduction rules to derive a set of reduction rules for YAWL nets with cancella-

tion regions. Section 3 presents ten reduction rules for YAWL nets with cancellation regions and without OR-joins and Section 4 presents two additional reduction rules for OR-joins. The reduction rules defined in Section 3 are also applicable to nets with OR-joins. Section 5 describes the implementation of our approach in the YAWL editor and presents results from verification of a complex YAWL workflow making use of YAWL reduction rules. Section 6 discusses related work and Section 7 concludes the paper.

Relationship to our other work on reduction rules: This journal article is the first published result for a set of YAWL reduction rules. In [19], a set of reduction rules for reset nets are presented. In [15], a set of reduction rules for reset/inhibitor nets are presented. Both technical reports together with a technical report version of this journal paper are available from BPM Center (http://www.bpmcenter.org).

2 Preliminaries

The workflow language YAWL [3] is a general and powerful language grounded in workflow patterns [4] and in Petri nets [9]. YAWL is the result of an in-depth analysis in control flow constructs in workflows, and provides direct support for those patterns hard to realise in Petri nets such as cancellation, the OR-join and multiple instances. A YAWL net is made up of tasks, conditions and flow relations between them. There are three kinds of split and three corresponding kinds of join; AND, XOR, and OR (see Figure 1). A task is enabled when there are enough tokens in its input conditions according to the join behaviour. When a task is executed, it takes tokens out of the input conditions and puts tokens in its output conditions according to the join and split behaviour respectively. An OR-join in YAWL waits for synchronisation, wherever possible. A task can have a cancellation region associated with it (dotted lines denote the cancellation region of a task). If a task is within the cancellation region of another task, it may be prevented from being started or its execution may be terminated (depending on the timing). If there is a cancellation region associated with a task, cancellation is triggered on the completion of that task.

A reset net is a Petri net with special *reset arcs*, that can clear the tokens in selected places [8]. The nature of reset arcs matches closely with the concept of cancellation in workflow modelling and reset nets are used as a formalism for modelling workflows with cancellation. A mapping from a YAWL net *without OR-joins* to a reset net has been defined in [17]. We briefly mention the main idea behind these transformations here using Figure 1. 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). A task within a cancellation region in YAWL is mapped to a reset arc (shown as a double-headed arrow) of its intermediate place in the corresponding reset net. An example of this is given in the bottom-left corner of Figure 1. If there is a cancellation region associated with a task, the execution of the task removes all the tokens from the conditions and from the intermediate places of tasks in the cancellation region.

A YAWL net satisfies the following three structural restrictions: there is (1) exactly one begin condition, (2) exactly one end condition and (3) every node in the graph is on a directed path from the begin condition to the end condition (see Figure 16). Reset Workflow Nets (RWF-nets) represent a subclass of reset nets with the same structural restrictions. Transforming a YAWL net without OR-joins according to Figure 1 results in the corresponding RWF-net. A YAWL net without OR-joins is considered sound if and only if the corresponding RWF-net is sound. An RWF-net

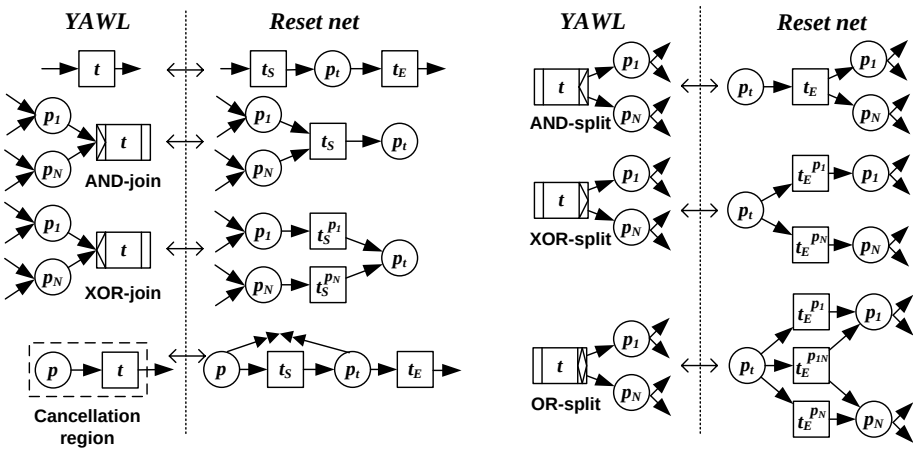

Fig. 1. Transformations from YAWL constructs to reset net constructs

is sound if and only if the net satisfies the following three properties; an option to complete (for every marking reachable from the initial marking with a token in the input place, there is an occurrence sequence leading to the end marking with a token in the output place), proper completion (the end marking is the only marking with at least one token in the output place), and no dead transitions (for every transition, there is a marking reachable from the initial marking such that that transition can be enabled) [16].

We now briefly summarise seven soundness preserving reduction rules for RWF-nets (see Figure 2). These rules are based on well-known Petri net reduction rules [7, 9] and have been extended and generalised as necessary to accommodate reset arcs. These reset net reduction rules are used to derive a set of reduction rules for YAWL nets with cancellation regions. They have been proven correct with respect to the soundness property in [16].

1. Fusion of series places rule for RWF-nets: (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. The rule holds if all places are reset by the same set of transitions. In particular, it means that the rule is applicable if no place is a reset place. 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 the rule holds. If one transition resets a place, then the other transitions must also reset the same place. 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 and 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.
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. This rule is very effective in reducing YAWL models as it will be seen in the next section.

In this section, we have consolidated our previous work in the area of workflow verification and reduction rules for RWF-nets. These RWF-net reduction rules together with the mappings from YAWL to reset nets are used to derive the YAWL reduction rules. In the next two sections, we focus on the new results: a set of reduction rules for workflow nets with cancellation regions and OR-joins.

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

Due to the mappings shown in Figure 1, it is possible to perform reduction of YAWL nets with cancellation regions and without OR-joins by first transforming the net into the corresponding RWF-net and then applying the RWF-net reduction rules. However, we decided to define a set of reduction rules at the YAWL net level so that (1) YAWL nets can be reduced without mapping to reset nets first and (2) it is possible to derive additional reduction rules for YAWL nets with OR-joins.

We now present ten soundness preserving reduction rules for various YAWL constructs[1]. In this paper, we limit ourselves to describing one of the rules, the *Fusion of series tasks rules* (ϕ^Y_{FST}), in full detail and briefly explaining the other rules due to space reasons. The detailed formalisation of all the rules can be found in [16]. First, we present some notations used in the definition of the ϕ^Y_{FST} rule. A YAWL net is formally represented by the tuple $(C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem)$ where C is a set of conditions, $i\mathbf{i}$, $o\mathbf{o}$ are unique input and output conditions, T is a set of tasks, F is the flow relation, $split$ and $join$ specify the split and join behaviours of each task and rem specifies the cancellation region for a task. The notations $split(t)$ and $join(t)$ for a task $t \in T$ return one of AND, XOR, OR. The notation $rem(t)$ for a task t returns the (possibly empty) set of tasks and conditions that it resets. We also write $rem^{\leftarrow}(x)$ for any $x \in (C \cup T)$, which returns the set of tasks that can reset x. For any $x \in (C \cup T)$, $\bullet x$ and $x\bullet$ 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 ${}^{N}\!\bullet x$ and $x\bullet^{N}$.

[1] We do not claim these rules to be complete.

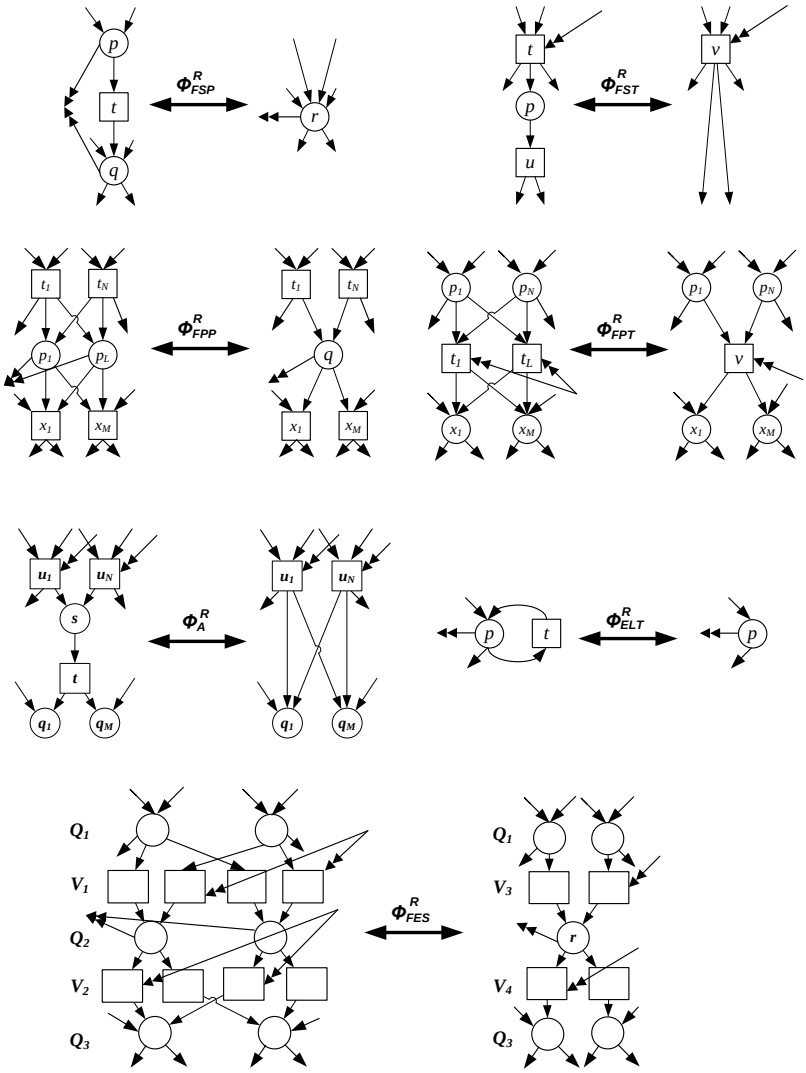

Fig. 2. Reduction rules for RWF-nets (Please note that the figure does not capture all the requirements for some rules (i.e., if a transition cannot have a reset arc or a place cannot be reset, the figure will not show this.)

– Fusion of series tasks rule: ϕ^Y_{FST}
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. Figure 3 visualises the ϕ^Y_{FST} rule. 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 resets certain places. 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 1 (Fusion of Series Tasks Rule: ϕ^Y_{FST}). *Let N_1 and N_2 be two YAWL 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. $F_2 = (F_1 \cap ((C_2 \times T_2) \cup (T_2 \times C_2))) \cup ({}^{N_1}{\bullet}t \times \{v\}) \cup (\{v\} \times u^{N_1}{\bullet}) \cup (\{v\} \times ({}^{N_1}t{\bullet} \setminus \{p\}))$
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. $join_2 = \{(z, join_1(z)) | z \in T_2 \cap T_1\} \cup \{(v, join_1(t))\}$

Theorem 1 (The ϕ^Y_{FST} rule is soundness preserving). *Let N_1 and N_2 be two YAWL 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 3 visualises the ϕ^Y_{FST} rule and sketches the proof of this rule.

– Fusion of series conditions rule: ϕ^Y_{FSP}
The *Fusion of Series Conditions Rule for YAWL nets* (ϕ^Y_{FSP}) rule 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 r (see Figure 4). The application requirements are similar to those for the ϕ^R_{FSP} rule. In addition, we require that task t is cancelled by the same set of tasks that remove tokens from conditions p and q.

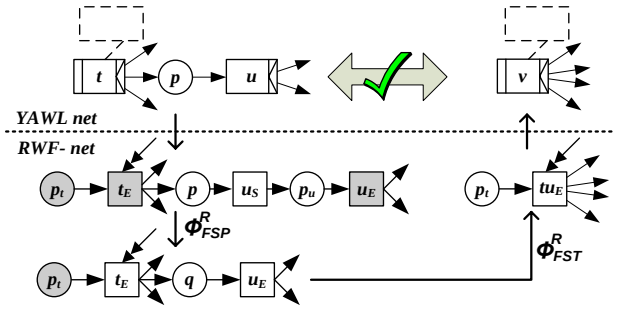

Fig. 3. Fusion of Series Tasks Rule for YAWL nets: ϕ^Y_{FST}

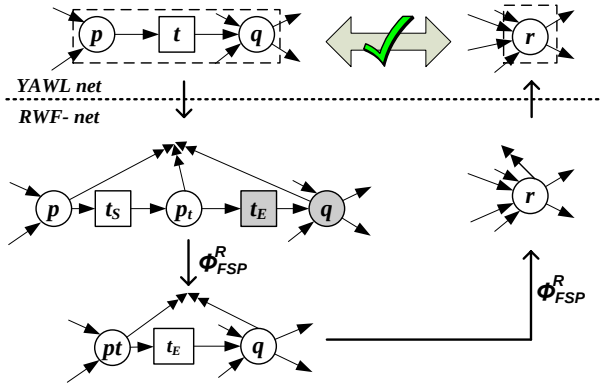

Fig. 4. Fusion of Series Conditions Rule for YAWL nets: ϕ^Y_{FSP}

– Fusion of parallel conditions rule: ϕ^Y_{FPP}
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 (see Figure 5). We require that all input tasks for these conditions are AND-split tasks and all output tasks for these conditions are AND-join tasks. In addition, we require that these conditions are reset by the same set of tasks, if any.
– Fusion of alternative conditions: ϕ^Y_{FAP}
The *Fusion of Alternative Conditions Rule for YAWL nets* (ϕ^Y_{FAP}) allows for the merging of two or more alternative conditions in a YAWL net with the same input tasks and output tasks (Figure 6). This rule is similar to the ϕ^Y_{FSP} rule except for the fact that all input tasks for these conditions are XOR-split tasks and all output tasks are XOR-join tasks.
– Fusion of parallel tasks rule: ϕ^Y_{FPT}
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 (see Figure 7). Two tasks are *identical* if both have the same input set and output set, the same split and join behaviour, do not cancel anything, and are not cancelled by any other tasks. We require that all tasks are AND-split and AND-join tasks.

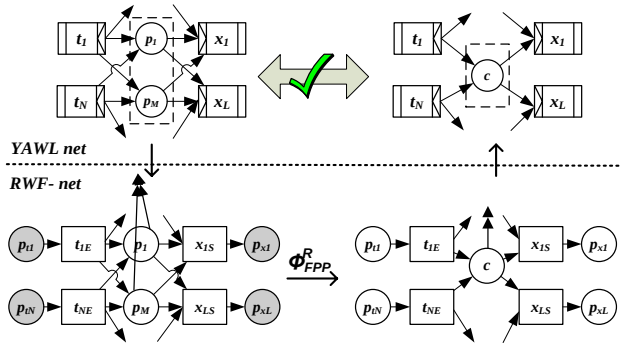

Fig. 5. Fusion of Parallel Conditions Rule for YAWL nets: ϕ^Y_{FPP}

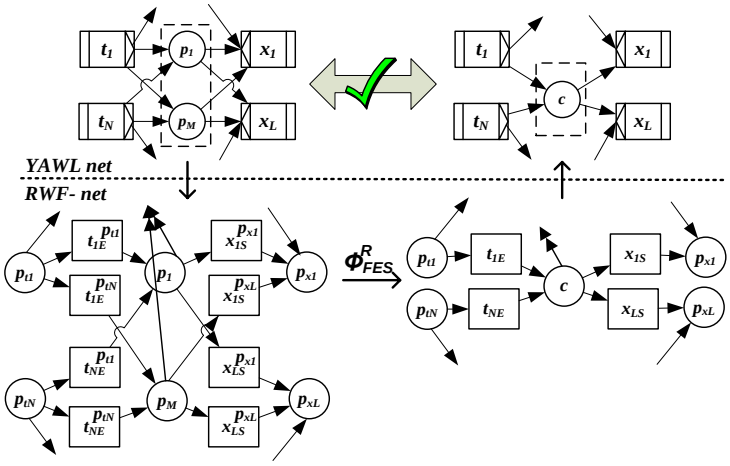

Fig. 6. Fusion of Alternative Conditions Rule for YAWL nets: ϕ^Y_{FAP}

– Fusion of alternative tasks rule: ϕ^Y_{FAT}
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 (Figure 8). 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.
– Elimination of self-loop tasks rule: ϕ^Y_{ELT}
The *Elimination of Self-Loop Tasks Rule for YAWL nets* (ϕ^Y_{ELT}) allows removal of a self-loop task in a YAWL net (see Figure 9). The ϕ^Y_{ELT} rule makes use of the ϕ^R_{FST} and the ϕ^R_{ELT} rule for RWF-nets. In addition, we require that t and p are not part of any cancellation region. Task t and associated arcs from t to p are abstracted in the reduced net.
– Elimination of self-loop conditions rule: ϕ^Y_{ELP}
The *Elimination of Self-Loop Conditions Rule for YAWL nets* (ϕ^Y_{ELP}) allows removal of a self-loop condition in a YAWL net (see Figure 10). The ϕ^Y_{ELP} rule makes use of the ϕ^R_{FSP} and the ϕ^R_{ELT} rule for RWF-nets. Condition x and associated arcs from x to t are abstracted in the reduced net.

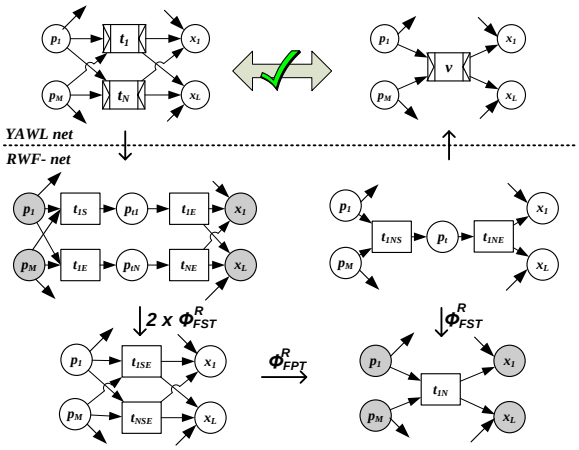

Fig. 7. Fusion of Parallel Tasks Rule for YAWL nets: ϕ^{Y}_{FPT}

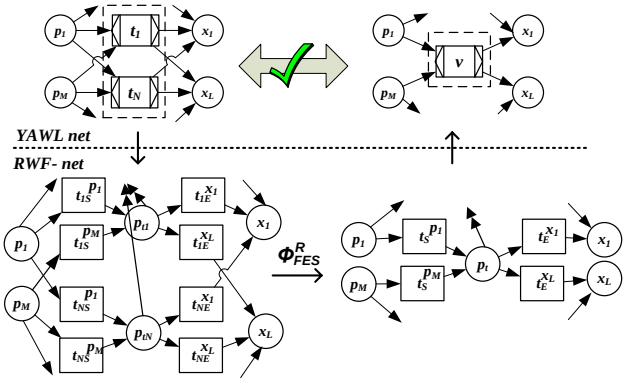

Fig. 8. Fusion of Alternative Tasks Rule for YAWL nets: ϕ^{Y}_{FAT}

– Fusion of AND-split and AND-join rule: ϕ_{FAND}
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 (see Figure 11). We require that tasks t, u, and conditions $p_1, ..., p_N$ are not part of any cancellation region nor do neither t nor u reset any places.
– Fusion of XOR-split and XOR-join tasks rule: ϕ_{FXOR}
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 (see Figure 12). We require that tasks t, u, and conditions $p_1, ..., p_N$ are not part of any cancellation region.

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

As verification techniques for YAWL nets with OR-joins utilise reachability analysis using the YAWL semantics [18], state space explosion is a real concern. Our objective is to identify possible

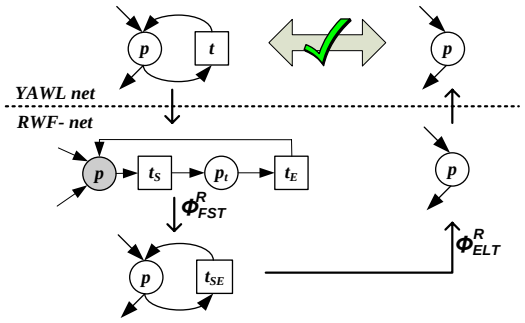

Fig. 9. Elimination of Self-Loop Tasks Rule for YAWL nets: ϕ^{Y}_{ELT}

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, first we propose to apply the reduction rules defined for YAWL nets without OR-joins to those parts of a YAWL net that do not have any OR-joins. This is possible as any OR-join behaviour is not affected by these reduction rules (i.e., If a token could arrive in one of its inputs, then it can also arrive after the reductions, and if a token could not arrive then it is also not possible after the reductions.). So, the ten reduction rules that are presented in Section 3 for YAWL nets *without* OR-joins apply equally to YAWL nets with 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. By making the assumption of safeness, we can ensure that more tokens cannot arrive at input places to an OR-join without firing the OR-join again. Hence, the OR-join enabling rule can be localised.

1. Fusion of an OR-join and another task rule: ϕ_{FOR}
   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 (see Figure 13). The rule requires that all inputs to an OR-join task (u) are from one task (t) (regardless of the split behaviour of that task) and all outputs from t go directly into u. 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.
   The ϕ_{FOR} rule is such that if *all* output conditions from t go directly into OR-join task u, then u can be abstracted. We can see that the ϕ_{FOR} rule holds under such requirements. Assume that task t is enabled at marking M and M' is reachable from 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

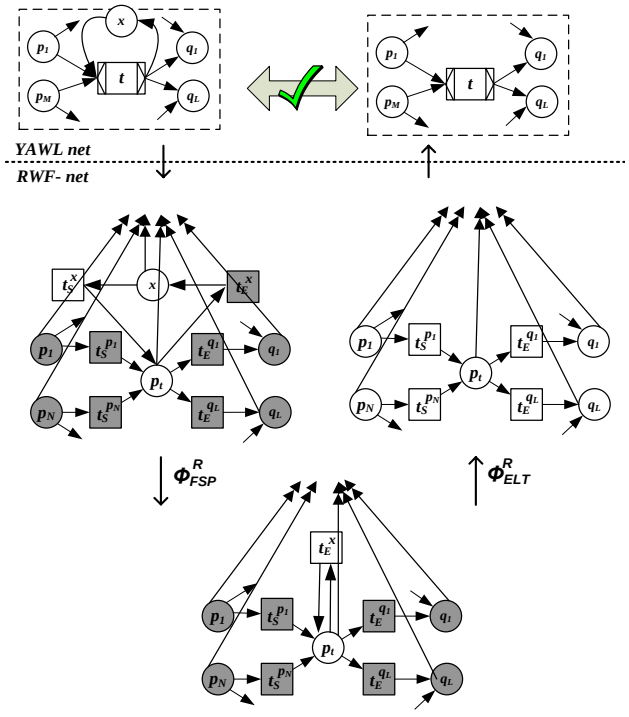

Fig. 10. Elimination of Self-Loop Conditions Rule for YAWL nets: ϕ^{Y}_{ELP}

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 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 the original net, it is also not possible to enable v in the reduced net 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. However, the rule is quite restrictive because to apply this rule, all output arcs from a task must go into an OR-join and the OR-join could not have any 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.

2. Fusion of incoming edges to an OR-join rule: ϕ_{FIE}
   The *Fusion of Incoming Edges to an OR-join for YAWL nets* (ϕ_{FIE}) rule allows for the merging

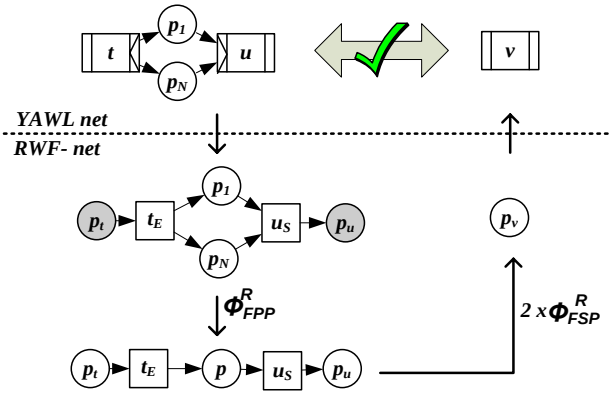

Fig. 11. Fusion of AND-split and AND-join tasks for YAWL nets: ϕ_{FAND}

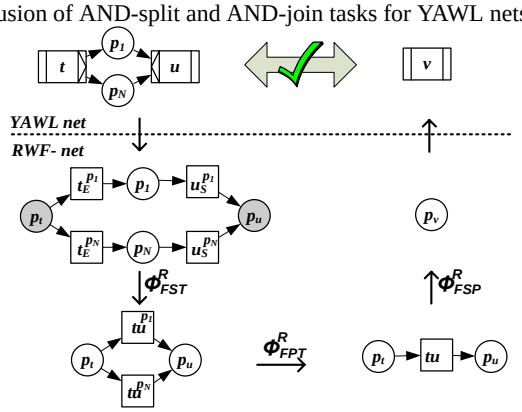

Fig. 12. Fusion of XOR-split and XOR-join tasks for YAWL nets: ϕ_{FXOR}

of two or more conditions that have the same input task and the same output task (an OR-join) into a single condition (see Figure 14). Also, we require that these conditions cannot be cancelled.

It is easy to see that the ϕ_{FIE} rule holds. As conditions in $p_1, ..., p_N$ and p have the same input task t and output task u, if some subset of conditions in $p_1, ..., p_N$ is marked at a marking in the original net, p is also marked at the corresponding marking in the reduced net. Under the assumption of safeness, if some conditions in $p_1, ..., p_N$ 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 $p_1, ..., p_N$.

5 Implementation

Reduction rules together with the three-step approach for verification using reduced nets have been implemented in the YAWL editor as follows.[2]

[2] www.sourceforge.net

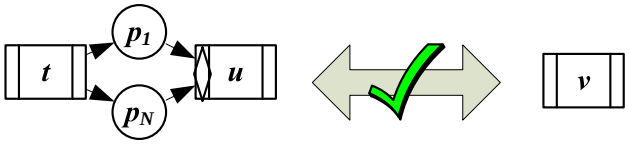

Fig. 13. Fusion of an OR-join and another task Rule for YAWL nets: ϕ_{FOR}

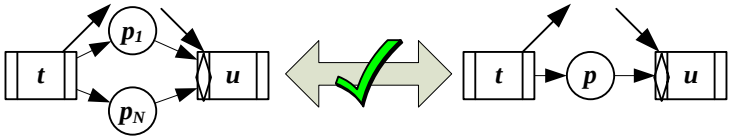

Fig. 14. Fusion of Incoming Edges to an OR-join Rule for YAWL nets: ϕ_{FIE}

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 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 for verification. The mappings between different nets are also stored for error reporting.
2. Verification is performed on the reduced net.
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.

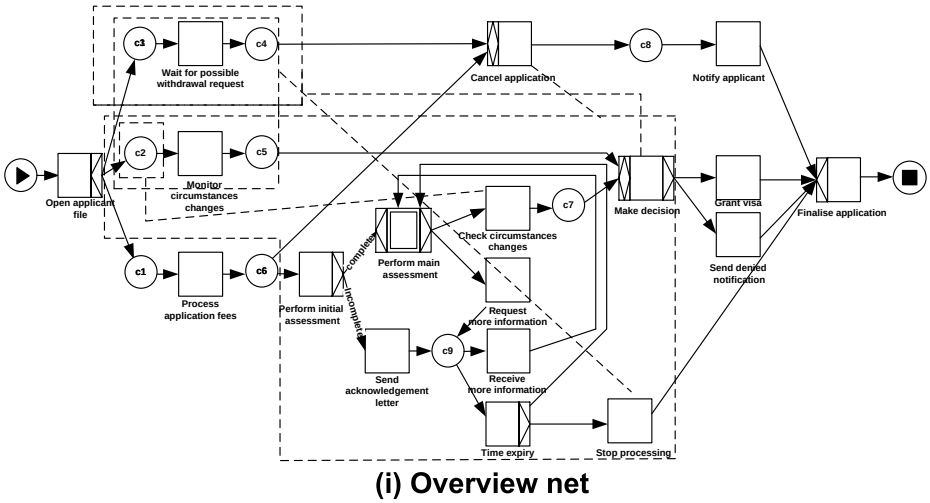

Fig. 15. *Overview*: the main YAWL net in the visa application process

We now illustrate the proposed reduction rules for verification using two process models. The first one is a a real-life process model: **Australian visa application process for general skilled migration**. This process is modelled "as is" using publicly available information from Department

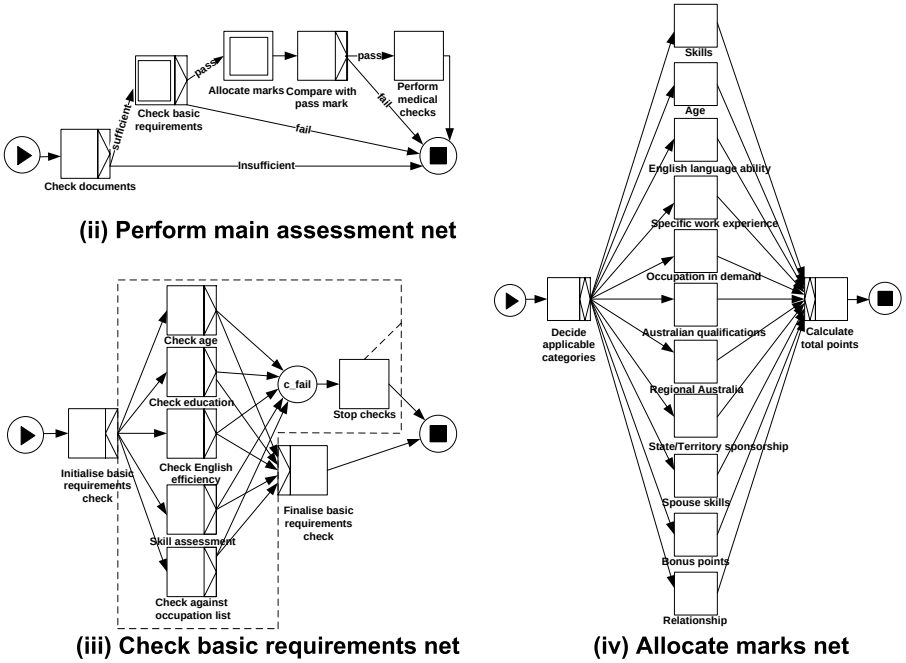

Fig. 16. YAWL (sub)nets: *Perform main assessment*, *Check basic requirements*, *Allocate marks*

of Immigration and Multicultural Affairs website.[3] This represents an example process model to showcase the various complex elements supported in YAWL. The second one is the **Document handling process during film production: YAWL4Film**. This YAWL4Film represents an real-world executable YAWL model that is currently being used to automate the document handling aspects of a film production process and provided as one of the solutions of the YAWL framework.[4]

The visa application process starts when a visa application is received by the immigration department and ends when a decision is made to grant or to deny the visa. The model represents the process from the viewpoint of a case officer who handles the visa application. The resulting YAWL workflow contains four nets *Overview*, *Perform main assessment*, *Check basic requirements*, and *Allocate marks*. Figure 15 shows the main net called *Overview*. The *Overview* net is a complex workflow as it contains multiple cancellation regions and two OR-joins, namely, *Cancel application* and *Make decision*. In the net *Overview*, the *Perform main assessment* is represented as a composite task and it unfolds into the YAWL net with the same name. Similarly, there are two composite tasks: *Check basic requirements* and *Allocate marks* in the *Perform main assessment* net and they also unfold into two YAWL nets with the corresponding names. Figure 16 shows the three subnets in the process.

Table 1 shows applicable YAWL and reset reduction rules for all nets in the *Visa application* process. The numbers in various columns represent the number of elements in the original net and in the corresponding reduced net, either for a YAWL net or an RWF-net. The row Reduced

[3] http://www.immi.gov.au accessed on 20 April 2006

[4] http://www.yawlfoundation.org/solutions/yawl4film.html

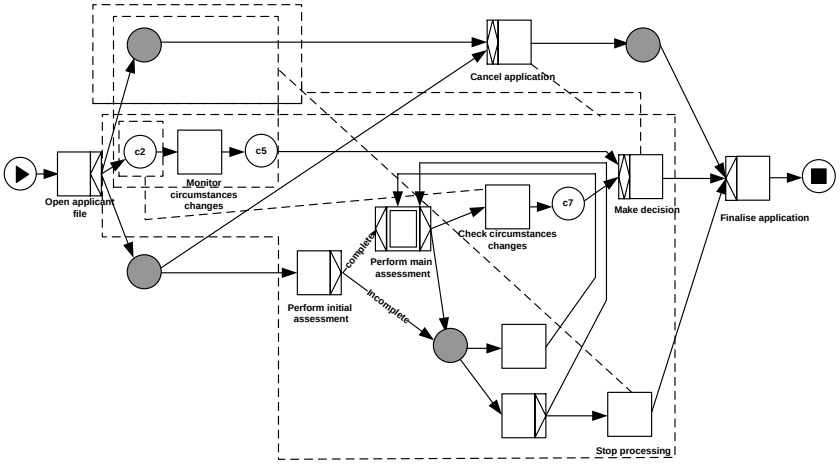

Fig. 17. *Overview*: The reduced Overview net with some of the reduced conditions shaded in grey

(both) displays the number of YAWL elements in column 1 and the number of RWF-net elements in columns 2, 3, and 4. We found that the reductions have dramatic effects at times and moderate effects on other occasions. For example, there are altogether 42 tasks and conditions (including implicit conditions between two tasks) in the original *Overview* net and after applying one of the YAWL reduction rules repeatedly, ϕ^Y_{FSP}, the number is reduced to 27. Figure 17 shows the reduced Overview net after the ϕ^Y_{FSP} rule has been applied. A YAWL condition that represents a reduced element is shaded grey. Please note that some of the reduced conditions might not be shown in the figure. The table also highlights that sometimes both YAWL and RWF-net reduction rules are necessary. For instance, the *Check basic requirements* net can only be reduced using the RWF-net reduction rules and not with YAWL reduction rules. Also, the *Allocate marks* can be reduced significantly from 37 to 3 elements if YAWL reduction rules are applied first followed by the reset reduction rules.

The efficiency we gain from applying reduction rules is *quite significant*. The time it takes to verify the soundness property of the *Overview* net decreased from 24.3 sec to 4 sec when reduction rules are used. Similar gains can be seen for the other two nets: *Check basic requirements* and *Perform main assessment*. As for the *Allocate marks* net, the results are quite spectacular after applying the ϕ^Y_{FSP} rule and the ϕ_{FOR} rule under the assumption of safeness. The net has a large state space due to the various possible OR-split and OR-join combinations. Without the use of reduction rules, the net suffers from the state explosion problem when determining the soundness property. After applying the reduction rules, the reduced *Allocate marks* net becomes quite trivial with just one input place, one output place and a task in between. As a result, the soundness check is completed almost instantaneously (less than one second). This is a huge improvement considering the fact that the soundness check for the *Allocate marks* net could not be completed in a reasonable time frame (more than 5 mins) due to state explosion.

In addition to the visa application process model, we have also verified the soundness property of a real YAWL process model used in the film production process with and without using these reduction rules. The YAWL4Film process is data-intensive and the process captures the various activities that need to be carried out for each shooting day on a film set. The process model makes use of a number of iterative loops to describe the repetitive modification of the documents

Net	Overview	Main Assessment	Basic Requirements	Allocate Marks
Original (YAWL)	42	11	21	37
Reduced (YAWL)	27	7	None	3
YAWL rules	ϕ^{Y}_{FSP}	ϕ^{Y}_{FSP}	None	ϕ^{Y}_{FSP},ϕ_{FOR}
Original (reset)	N/A	23	42	N/A
Reduced (reset)	N/A	9	32	N/A
Reset rules	N/A	ϕ^{R}_{FST}	ϕ^{R}_{FSP}	N/A
Reduced (both)	27	9	32	3
Both rules	ϕ^{Y}_{FSP}	ϕ^{Y}_{FSP},ϕ^{R}_{FSP},ϕ^{R}_{FST}	ϕ^{R}_{FSP}	ϕ^{Y}_{FSP},ϕ_{FOR},ϕ^{R}_{FST}
Soundness (original)	24.3 sec	1.9 sec	26.4 sec	$>$5 mins
Soundness (reduced)	4 sec	0.8 sec	4.1 sec	0.7 sec

Table 1. Demonstrating the effects of reduction rules on the soundness property check for *Visa application* process

involved. The process also contains an OR-join task to determine whether all the activities need to be repeated for another shooting day. Figure 18 represents a screenshot from the YAWL Editor that shows the soundness property results for the YAWL4Film process model. Even though the number of elements in the model is relatively small (54 elements) and there are only 6755 markings in the state space, the state space analysis required for verification is time consuming as the YAWL4Film model contains an OR-join and seven iterative loops on the path for different shooting days. Without the use of YAWL reduction rules the soundness check takes more than 30 mins to complete due to the combination of the seven loops which results in many reachable markings and the multiple calls to the expensive OR-join analysis algorithm requiring each enabling decision to take into account the many iterative loops on the path and construct a reset net every time. In fact, this real-world process model with a number of iterative loops highlighted the need for an additional reduction rule (the ϕ^{Y}_{ELP} rule) which removes self-loop conditions from a net. This new reduction rule has now been added to our collection. After applying the YAWL reduction rules repeatedly (in this case, the ϕ^{Y}_{FSP} rule, the ϕ^{Y}_{ELP} rule and the ϕ_{FAND} rule), the net can be reduced from 54 to 29 elements, the number of reachable markings in the state space is now 105 and the soundness property check is performed in less than one second. This experiment further highlights the importance of applying reduction rules for verification of YAWL nets.

6 Related work

Different verification approaches for workflows have been proposed in the literature. Graph reduction techniques are proposed to identify certain conflicts [10] and the authors claim that the correctness of a workflow can be determined by whether it can be reduced to an empty graph after repeated application of the reduction rules. In [2], it is shown that the reduction rules presented by Sadiq and Orlowska are not complete and as a result some correct models are classified as incorrect. In [14], the authors present an approach to decide the relaxed soundness property of workflows with cancellation regions and OR-joins using invariants. The approach taken results in an approximation of the OR-join semantics and transformation of YAWL nets into Petri nets with inhibitor arcs to represent cancellation regions. Reduction rules have been suggested to be used together with Petri nets for the verification of workflows. A number of authors have investigated

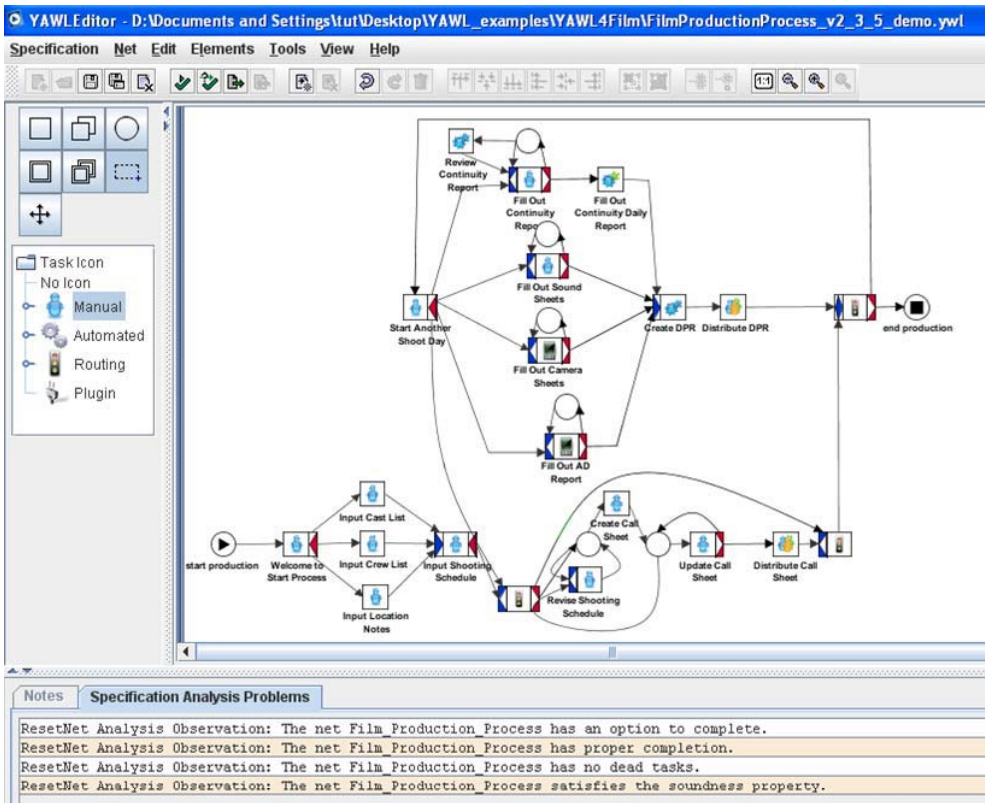

Fig. 18. *YAWL4Film*: Soundness property results for a real-world YAWL model with an OR-join

reduction rules for Petri nets [5, 6, 9] and for free-choice Petri nets [7] and time Petri nets [11]. In Verbeek's thesis [13], the author proposes reduction rules for WF-nets based on the Petri net reduction rules. 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. In Streit's paper [12], simple YAWL reduction rules have been presented for the purpose of collapsing large business processes. This represents a starting point for the use of reduction rules in visualisation. However, their reduction rules do not hold for YAWL nets with cancellation regions.

7 Conclusion

Given that deployed workflows may execute for a long time and may take many actions that cannot be undone in a simple manner, it is desirable to detect errors at design time. Existing Petri net based verification techniques with reduction rules cannot be used for workflows with cancellation regions and OR-joins as these constructs have non-local implications. In [18], a new verification technique for workflows with cancellation and OR-joins was proposed using reset nets and reachability analysis. For large workflows with cancellation regions and OR-joins, reduction rules can undoubtedly speed up the verification process. In this paper, a set of soundness-preserving re-

duction rules has been proposed to speed up the verification process. We have also demonstrated the effectiveness of the proposed YAWL reduction rules in achieving a more efficient analysis of workflows with cancellation regions and OR-joins using the visa application example and a real process model used for film production.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. G. Berthelot. Checking properties of nets using transformation. In *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets-selected papers*, pages 19–40, London, UK, 1986. Springer-Verlag.
6. G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets, Proceedings of an Advanced Course, Part 1*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376, Bad Honnef, September 1986. Springer-Verlag.
7. 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.
8. 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.
9. T. Murata. Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
10. W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, *Proceedings of the 11th Conference on Advanced Information Systems Engineering (CAiSE 1999)*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209, Heidelberg, Germany, June 1999. Springer-Verlag.
11. R.H. Sloan and U.A. Buy. Reduction Rules for Time Petri Nets. *Acta Informatica*, 33(7):687–706, 1996.
12. Alexander Streit, Binh Pham, and Ross Brown. Visualization support for managing large business process specifications. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of 3rd International Conference on Business Process Management, BPM 2005, Nancy, France, September 5-8, 2005*, volume 3649, pages 205–219, 2005.
13. H.M.W. Verbeek. *Verification of WF-nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, June 2004.
14. H.M.W. Verbeek, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Relaxed Soundness and Invariants. *The Computer Journal*, 50(3):294–314, 2007.
15. H.M.W. Verbeek, M.T. Wynn, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Reduction rules for Reset/Inhibitor Nets. Technical report BPM-07-13, BPM Center (bpmcenter.org), 2007.

16. M.T. Wynn. *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*. PhD Thesis, Faculty of Information Technology, Queensland University of Technology, Nov 2006.
17. 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.
18. 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.
19. 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.