Synchronisation and Cancellation in Workflows based on Reset nets*

Moe Thandar Wynn[1], W.M.P. van der Aalst[1,2], A.H.M. ter Hofstede[1] and David Edmond[1]

Business Process Management Program
Faculty of Information Technology, 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.
{*w.m.p.v.d.aalst*}*@tm.tue.nl*

Abstract. Workflow languages offer constructs for coordinating tasks. Among these constructs are various types of splits and joins. One type of join, which shows up in various incarnations, is the OR-join. Different approaches assign a different (often only intuitive) semantics to this type of join, though they do share the common theme that synchronisation is only to be performed for active threads. Depending on context assumptions this behaviour may be relatively easy to deal with, though in general its semantics is complicated, both from a definition point of view (in terms of formally capturing a desired intuitive semantics) and from a computational point of view (how does one determine whether an OR-join is enabled?). In this paper the concept of the OR-join is examined in detail in the context of the workflow language YAWL, a powerful workflow language designed to support a collection of workflow patterns and inspired by Petri nets. The OR-join's definition is adapted from an earlier proposal and an algorithmic approach towards determining OR-join enablement is examined. This approach exploits a link that is proposed between YAWL and reset nets, a variant of Petri nets with a special type of arc that can remove all tokens from a place. Structural restriction and active projection techniques are also proposed for algorithm optimisation.

Keywords: OR-join, YAWL, Workflow patterns, Synchronizing merge, Petri nets, Reset nets.

1 Introduction

Workflow specifications should capture various aspects of business models such as the flow of control, the flow of data, the structure of the organisation, and the use of resources (see e.g. [15]). The control flow perspective captures the execution interdependencies between the tasks of a business process. An in-depth analysis and comparison of a number of commercially available workflow management systems has been performed [4] and the findings demonstrate that the interpretation of even the basic control

* This is the extended version of the paper appeared in the proceedings of ATPN 2005 [23].

flow constructs is not uniform and it is often unclear how the more complex requirements could be supported. The authors propose 20 workflow patterns to address control flow requirements in a language independent style. YAWL (Yet Another Workflow Language) is a result of this analysis, it provides direct support for most patterns [3]. YAWL has a formal semantics specified as a transition system. Although YAWL exploits concepts from Petri nets, it also provides direct support for those patterns hard to realise in Petri nets. One of these patterns corresponds to the synchronising merge better known as the OR-join, the focus of this paper. In practice, there is a need for a construct like the OR-join as is evident from e.g. the fact that many commercial workflow systems and business process modelling tools support OR-join-like constructs. Many systems and languages struggle with the semantics and implementation of the OR-join because the OR-join may require a synchronisation depending on an analysis of future execution paths, i.e., the semantic is non-local and requires some non-trivial reasoning. Workflow management systems like InConcert, eProcess, and WebSphere MQ Workflow have solved problems related to the OR-join using syntactical restrictions. IBM WebSphere MQ Workflow [20] (formerly known as MQSeries Workflow and FlowMark and also used as a basis for the new BPEL standard) offers full support for the OR-join but in order to do this it requires the workflow to be acyclic, i.e., the only way to introduce loops is by executing the entire (sub)process [2]. Other systems like Eastman and Domino Workflow use a non-local semantics similar to the one used in YAWL. Such a non-local semantics may lead to unexpected results. Moreover, a non-local semantics may result in poor performance as is stated in the manual of Eastman: "Parallel instances can accumulate at a Join workstep if the instances are routed to the workstep by preprocessing rules. These instances will eventually be joined by a RouteEngine subprocess (thread) that examines Join worksteps for such instances. This Join scavenger thread reduces system efficiency, so routing to Join worksteps using preprocessing rules should be avoided" [11]. These findings illustrate the practical relevance of the OR-join and serve as a motivation for the work reported in this paper. Experience with these systems shows that it is difficult to select a suitable semantics and implement it efficiently. For a more complete discussion on workflow systems' support for OR-join semantics, we refer to [2, 4, 16–18].

The OR-join is a control flow construct that sometimes behaves like an AND join and sometimes like an XOR join based on the current context. Consider the car servicing scenario shown in Figure 1. When a customer requests car servicing, the mechanic needs to perform a number of tasks. After scheduling the car for service, the mechanic performs two tasks: to make a checklist of servicing requirements and to check for other faults that need to be repaired. These two tasks can be done in any order. If the mechanic finds the need to repair, he/she will wait until other service requirements are identified before performing necessary repairs. If the service requirements have been identified first, the mechanic will wait for the outcome of the other task: check for faults. There are two possible outcomes from check for faults: the fault is either detected or there is no fault. When the outcome is known, servicing can be started. As a result, perform servicing task has been modelled as an OR-join. Before generating the bill, we should make sure that all the necessary checks and maintence tasks have been performed. If there is no fault, generate bill task will wait for synchronisation until perform servicing

task is completed. Otherwise, it can be started when the servicing has been completed for both regular maintence and necessary repairs. Hence, generate bill task is also modelled as an OR-join.

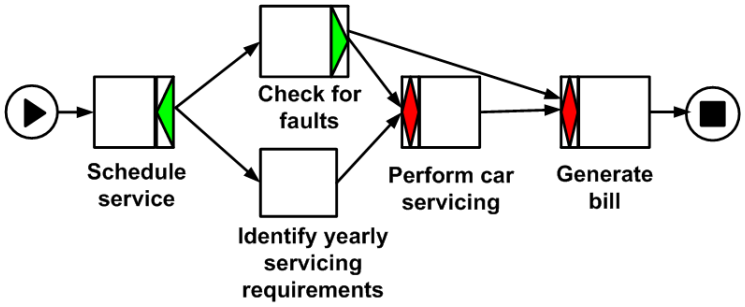

Fig. 1. Car servicing scenario

Variants and interpretations of the OR-join have been proposed in the literature. In [22], several possible interpretations of OR-join semantics in the context of Event-driven Process Chains (EPCs) are discussed. If the OR-join is preceded by a matching OR-split, the OR-join semantics is taken to be "wait for the completion of all paths activated by the matching split". If there is no matching split, there could be at least three interpretations of an OR join: "wait-for-all", "first-come" and "every-time" [22]. In [2], the authors highlight the technical, conceptual and practical problems with the formal semantics of the OR-join in EPCs. The authors suggest that there is no sound formal semantics for EPCs that seems to satisfy the intuitive semantics and that any formal semantics for EPCs will impose some restrictions or will deviate from the informal semantics to some extent. The authors demonstrate the problems using vicious circles, which are formed when two or more OR-joins are in a feedback loop and each OR-join waits for the other OR-join to complete first. On the other hand, in [17] a semantic framework for formally defining the non-local semantics of EPCs including the OR-join is proposed. The author states that "a single transition relation cannot precisely capture the informal semantics of EPCs". It is proposed that the non-local semantics be defined as a pair of transition relations and a semantic definition using techniques from fixed point theory is presented [17, 18]. The OR-join approach in YAWL [3] is intended to be a generalised approach and the formal semantics of the OR-join is defined by ignoring all other OR-joins. This approach is described as "ad hoc in some way" [17]. In this paper, we propose new OR-join semantics that takes into account preceding OR-joins. Note that the contribution of this paper is not limited to YAWL. This paper provides suitable semantics for OR-joins and gives a concrete algorithm to support the implementation.

The contributions of this paper are threefold. Firstly, we re-examine the OR-join semantics as proposed in [3], because its behaviour does not match the informal semantics in the context of OR-joins depending on other OR-joins and composite tasks with OR-joins (they cannot be treated like black boxes). Secondly, for purposes of the

OR-join definition and analysis, we propose an abstract view on YAWL, one which is formalised in terms of *reset nets* [5, 6, 9, 10, 12–14]. Reset nets are considered the most suitable formalism as reset arcs provide direct support for the cancellation feature in YAWL (another concept introduced to YAWL as a result of the workflow patterns and the difficulty of realising this feature in Petri nets). Thirdly, the mapping of YAWL nets to reset nets is exploited to find an algorithmic solution to the non-trivial problem of OR-join enablement.

The rest of the paper is organised as follows. In Section 2, we discuss the problems with the OR-join semantics in YAWL [3] and propose alternative treatments for OR-joins by taking into account other OR-joins in a YAWL net and also propose abstractions to enable reset net mappings. A YAWL net is defined formally as an EWF-net. In Section 3, the definitions of EWF-nets (Extended Workflow Nets) and reset nets are presented. In Section 4, we propose a new semantics for the OR-join in YAWL. In Section 5, we propose an algorithm for OR-join analysis based on backwards search techniques drawn from the area of Well-Structured Transition systems [9, 14]. In Section 6, we present two restriction techniques to improve the efficiency of OR-join analysis. In Section 7, we describe how the algorithm and the restriction techniques are implemented in the YAWL engine together with execution times. Section 8 concludes the paper and briefly discusses other possible optimisation techniques from the field of Petri nets.

This paper is an extended version of the ATPN2005 conference paper [23]. More examples have been added to illustrate the OR-join semantics in Section 2. We also introduce more worked examples to demonstrate the inner workings of the proposed algorithm in Section 4. One major extension is a proof for backwards firing rule used to generate predecessor markings which can be found in Section 5. In Sections 6 and 7 we also provide additional new results. In Section 6, we discuss restriction techniques to further improve the OR-join calculations. In Section 7, we describe the implementation in the context of the open source system YAWL and provide a detailed analysis of its performance.

2 OR-join semantics in YAWL

In this section, we first demonstrate the informal semantics of an OR-join using a number of examples. We then discuss two problems associated with multiple OR-joins in a YAWL net using the OR-join semantics as defined in [3]. We then propose some alternative treatments for OR-joins on the path to other OR-joins.

2.1 The OR-join in YAWL

A YAWL model is made up of tasks, conditions (in a Petri net, these would be referred to as places) and a flow relation between tasks and conditions. Each YAWL model has one start condition and one end condition. There are three kinds of split and three corresponding kinds of join in YAWL; they are AND, XOR and OR. The splits, joins, conditions and cancellation symbols for YAWL are shown in Figure 2. YAWL uses the terms tasks and conditions to avoid confusion with Petri net terminology (transitions

and places). A task is enabled when there are enough tokens in its input conditions according to the join behaviour. Informally, an AND-join task is enabled if there are tokens in all the input conditions of the AND-join. An XOR-join task is enabled if there is at least one token in one of the input conditions. The decision for enabling tasks with AND-joins or with XOR-joins can be made locally as it only depends on the existence of tokens in the input conditions. OR-join semantics, on the other hand, is non-local. An OR-join task is enabled at a marking iff at least one of its input conditions is marked and it is not possible to reach a marking that still marks all currently marked input conditions (possibly with fewer tokens) and at least one that is currently unmarked. If it is possible to place tokens in the unmarked input conditions of an OR-join in the markings reachable from the current marking, then the OR-join task should not be enabled and wait until either more input conditions are marked or until it is no longer possible to mark more input conditions. This is the desired behaviour of an OR-join and we will refer to this as *the informal semantics* of an OR-join. 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. A task can have a cancellation set associated with it. 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 is achieved by removing tokens from internal conditions of the task.

Figure 3 is a YAWL net where A is an OR-split task and E is an OR-join task. An OR-split task is called multiple choice as one or more paths can be selected after executing the task. Consider a marking $M = c1 + c5$ (i.e. a marking with two tokens one in $c1$ and one in $c5$). This marking results from the scenario where two outgoing paths leading to B and to C, were selected after completing task A, and where task C has been executed. At M, there is a token in the input condition $c5$ of OR-join task E. To determine whether task E should be enabled at M, we need to find out whether tokens

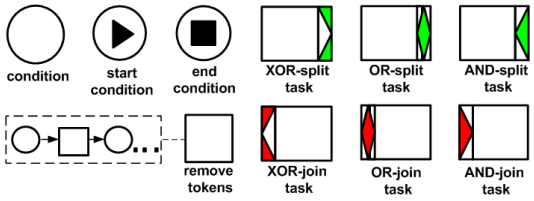

Fig. 2. Splits, joins, conditions and cancellation in YAWL

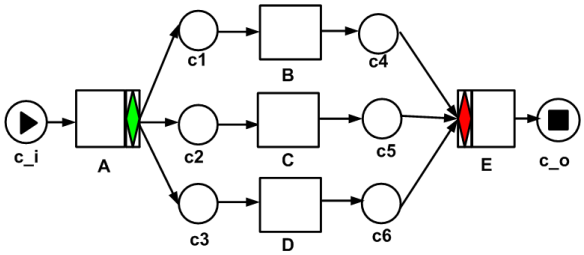

Fig. 3. A structured YAWL net with an OR-split task A and an OR-join task E

could be put into $c4$ or $c6$ in the reachable markings of M. It is possible to reach a new marking $M' = c4 + c5$ from M by firing task B and therefore, E should not be enabled at M. Now consider whether task E would be enabled at marking $M' = c4 + c5$. At M', $c4$ and $c5$ have one token each and there are no other tokens in the net. Hence, there is no possibility of another token arriving in $c6$ in the reachable markings of M'. Task E should be enabled at M'. As this is a "structured" YAWL net, task E would not be enabled until the tokens from all the active threads from task A reach the input conditions of E.

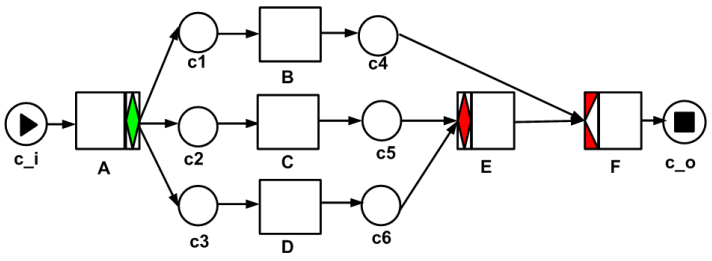

Fig. 4. A YAWL net modified from figure 3

From the above example, it could be thought that an OR-join evaluation only depends on the number of active paths out of an OR-split. If that is true, it is possible to know in advance the number of active paths to wait for synchronisation. We have slightly modified the example in Figure 3 to demonstrate that this is false. In Figure 4, $c4$ is an input condition of task F and $c5$ and $c6$ are input conditions of task E. Consider a marking M=$c1 + c5$. In this case, there is no reachable marking from M that can put more tokens into $c6$ and therefore, E should be enabled at M. So, even though two active paths are chosen after OR-split task A, the OR-join evaluation should not wait for tokens from both paths, as it is possible that not all the tokens might be on the path to an OR-join task.

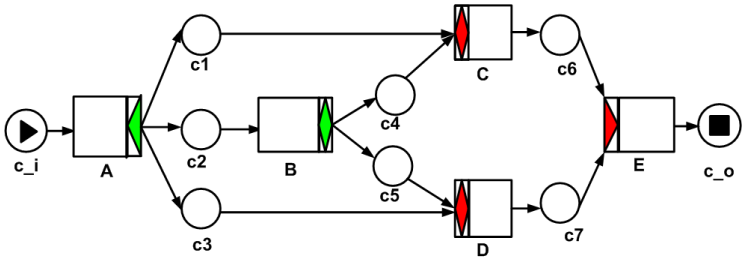

Fig. 5. A YAWL net with two OR-join tasks C and D

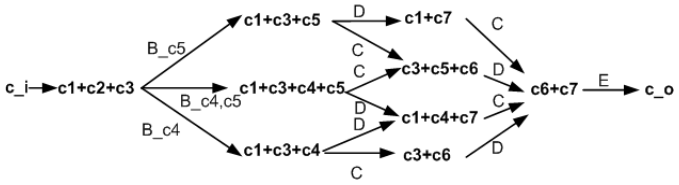

Fig. 6. Reachability graph of the YAWL net in Figure 5

Next, we will demonstrate the behaviour of OR-joins using an example with one OR-split and two OR-joins. The example in Figure 5 demonstrates a YAWL net with AND-split task A, AND-join task E, OR-split task B and OR-join tasks C and D. The reachability graph of Figure 6 shows the reachable markings from the initial marking c_i to the end marking c_o. A node in the reachability graph represents a reachable marking and an edge represents a task that is executed to reach that particular marking. [1] First consider a marking $M = c1 + c2 + c3$ where there is a token in input condition $c1$ of OR-join task C and in input condition $c3$ of OR-join task D in addition to the token in input condition of task B. To determine whether tasks C or D should be enabled at M, we need to find out whether tokens could be put into $c4$ or $c5$ in the reachable markings from M. We can see that by executing task B, it is possible to reach markings $c1 + c3 + c5$ or $c1 + c3 + c4 + c5$ that mark $c5$, an input condition of task D not marked in M. Alternatively, markings $c1 + c3 + c4$, $c1 + c3 + c4 + c5$ could be reached by executing task B and they mark $c4$, an input condition of task C not marked in M. As it is possible to reach a new marking from M which can put a token in an unmarked input condition of the OR-join tasks C and D, neither task C or D should be enabled at M. If a marking $M' = c1 + c3 + c4$ is considered, where all the input conditions of C (i.e., $c1$ and $c4$) are marked, then C would be enabled at M'. Task D will also be enabled at M' as it is not possible for another token to arrive at input condition $c5$. Note that in the scenario where we move from M to M', task D was not enabled in M and, although no tokens were added to the input conditions of this task, it became enabled in M'. In this example, the two OR-joins do not interfere with one another as they do not share input conditions.

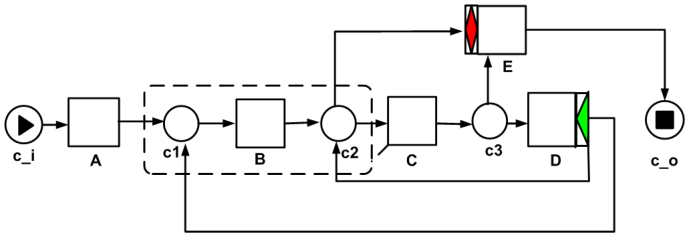

Fig. 7. Cancellation task C with an infinite loop and cancellation

[1] Note the overloading of notation, i.e., here c_o is a multiset denoting the marking with one token in condition c_o.

Now, let us consider OR-joins in the context of cancellation. In Figure 7, we describe a YAWL net with (i) task C removing tokens from the conditions $c1$, $c2$ and from internal conditions of task B when firing, (ii) an OR-join task E and (iii) two infinite loops between $c1$, $c2$, $c3$, C and D. At a marking $M = c2$, one of the input conditions of E is marked and an analysis needs to be performed to decide whether both $c2$ and $c3$ could be marked in reachable markings of M. We can observe the following sequence of reachable markings from M: $c2 \stackrel{C}{\rightarrow} c3 \stackrel{D}{\rightarrow} c1 + c2 \stackrel{B}{\rightarrow} 2c2 \stackrel{C}{\rightarrow} c3$. Similarly, there is another sequence: $c2 \stackrel{C}{\rightarrow} c3 \stackrel{D}{\rightarrow} c1 + c2 \stackrel{C}{\rightarrow} c3$, note that this is due to the cancellation feature of C removing tokens from $c2$ when firing. Regardless of which path is taken from the marking $c2$, it can only reach a marking of $c3$ (i.e. at the expense of $c2$). The conclusion is that it is not possible to reach a marking $c2 + c3$ or bigger from M and therefore, E should be enabled at M.

Suppose now that task C no longer has a cancellation set associated with it in Figure 7. From the marking $M = c2$, we can observe the following sequence of reachable markings: $c2 \stackrel{C}{\rightarrow} c3 \stackrel{D}{\rightarrow} c1 + c2 \stackrel{B}{\rightarrow} 2c2 \stackrel{C}{\rightarrow} c2 + c3$. As it is possible to reach $c2 + c3$ which marks more input conditions of E, E should not be enabled at M. This example demonstrates the possible effect that the cancellation feature of a task has on the OR-join analysis.

From the above examples, it is clear that the OR-join semantics requires careful analysis and the decision to enable an OR-join cannot be made locally. Any OR-join algorithm must evaluate possible reachable markings from a current marking to determine whether there is a possibility of a token arriving at a currently unmarked input condition of an OR-join (while all input conditions which were already marked remain marked though possibly with fewer tokens). This algorithm potentially needs to be applied every time a marking changes and the OR-join analysis could place a significant load on any workflow engine required to execute it, cf. the quote from the manual of Eastman [11] in the introduction.

2.2 Problems with OR-join semantics as defined in [3]

Two problems may be identified with the OR-join semantics of YAWL as it has been defined in [3]. The first problem is related to the treatment of other OR-joins preceding an OR-join under consideration. The OR-join semantics as defined in [3] ignores other OR-joins when analysing whether a particular OR-join should be enabled at a given marking. In Figure 8, there are two OR-join tasks, E and F in the YAWL net. Consider a marking $M = c1 + c3$ where the OR-join analysis for F is performed. After executing task C, it is possible to reach either $c3 + c4$, $c3 + c5$ or $c3 + c4 + c5$. One possible occurrence sequence is $c1 + c3 \stackrel{C}{\rightarrow} c3 + c4 + c5 \stackrel{D}{\rightarrow} c3 + c4 + c6 \stackrel{E}{\rightarrow} c3 + c7$. Hence, $M' = c3 + c7$ is a reachable marking from M. However, the OR-join semantics as defined in [3] ignores other OR-joins on the path to F, so task E and the associated conditions will not be taken into account, and M' is therefore not considered as a reachable marking during the OR-join analysis of F. As a result, the analysis will conclude incorrectly that there is no possibility of another token arriving in $c7$, F would be enabled at M and no synchronisation takes place. This behaviour is probably not what one would expect from this model. It would also result in multiple executions of F and then more

than one token would be produced for c_o. A YAWL model which can produce a token for the output condition c_o while still having tokens in the other conditions is considered as not having proper completion and is therefore not sound [1]. We have seen that as the analysis of a given OR-join does not consider the possibility of a token arriving from a path which has an OR-join, this could result in premature enabling and multiple execution of OR-join tasks.

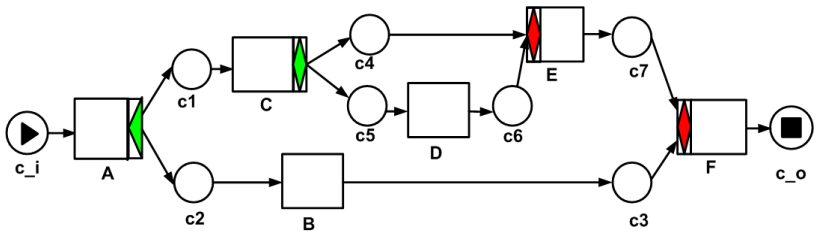

Fig. 8. A YAWL net with two OR-join tasks E and F

The second problem relates to unfolding of composite tasks during an OR-join analysis. This implies that a YAWL net at a lower level cannot be considered as a black box. If a YAWL net at a lower level contains OR-joins, it will impact on the OR-join analysis at a higher level net. Consider a specification where task B in Figure 8 is a composite task that is unfolded into a YAWL net with an OR-split and an OR-join task as shown in Figure 3. The composite task B will be unfolded to the net in Figure 3 (including the OR-join task E at lower level). In practise, it can be started with a token in $c2$ and after completion, will put a token in $c3$. However, during OR-join analysis for F at a marking $M = c2 + c7$, the net will be unfolded and OR-join task E at the lower level is ignored. The OR-join semantics in [3] will then conclude incorrectly that F should be enabled at M because it is not possible to have a token in $c3$. This demonstrates that composite tasks cannot be seen as black boxes and the analysis takes into account the lower level net that make up a YAWL specification.

2.3 Optimistic and pessimistic approaches

The informal semantics of an OR-join can be guaranteed when there is only one OR-join in a given YAWL net. However, when dealing with multiple OR-joins where one precedes the other, the semantics are not well-defined. Instead of ignoring other OR-join tasks during the analysis, we propose two alternative treatments for those OR-joins: either as XOR-joins (*optimistic*) or as AND-joins (*pessimistic*). Both optimistic and pessimistic approaches support the informal semantics by delaying enablement when there is a possibility of more tokens arriving to unmarked input conditions of an OR-join. We believe that these two alternatives result in formal semantics which is more closely related to the informal semantics of OR-joins and still allow for sound semantics (i.e., avoids the fixpoint problems discussed in [2]).

The treatment of an OR-join on the path to another OR-join as an XOR-join is an *optimistic* approach. It is considered *optimistic* as the analysis will wait for synchronisa-

tion if there is at least one token to enable the resulting XOR-join. Consider a marking $M = c1 + c3$ in Figure 8 where an OR-join analysis for task F would be performed. Instead of ignoring the OR-join task E during the analysis, it will be treated as an XOR-join task. It means that the occurrence sequence $c1 + c3 \xrightarrow{C} c3 + c4 \xrightarrow{E} c3 + c7$ would be considered. As a result, F is not enabled at M. This interpretation of OR-join task E as an XOR-join, prevents F from being enabled prematurely and it matches more closely with the informal semantics of an OR-join.

The treatment of an OR-join on the path to another OR-join as an AND-join is a *pessimistic* approach, as this approach now requires tokens in all input conditions of the AND-join and if it is not possible, the OR-join will be enabled. Consider again $M = c1 + c3$ in Figure 8 where an OR-join analysis for task F would be performed. This time, instead of ignoring task E, it will be treated as an AND-join task. Due to the OR-split behaviour of task C, tokens can be present in $c4$ or $c5$ or both after firing C. The occurrence sequence $c1 + c3 \xrightarrow{C} c3 + c4 + c5 \xrightarrow{D} c3 + c4 + c6 \xrightarrow{E} c3 + c7$ is possible. As a token can be put in $c7$ while $c3$ remains marked, F is not enabled at M. This preserves the same informal semantics as an optimistic approach, and both approaches result in delaying the enablement of the OR-join task F.

We have also found that when OR-joins are in conflict, there might not be any satisfactory treatment for OR-joins. n Figure 9, we have an unusual situation described as a vicious circle in [17] where the OR-joins are in conflict and it is unclear what the informal semantics of the model should be. In Figure 9, there are two OR-join tasks B and C which are in conflict with each other and this Figure is inspired by [17]. Condition $c3$ is an output condition of C and an input condition of B and $c4$ is an output condition of B and an input condition of C. Consider a marking $c1 + c2$ where an OR-join analysis is to be carried out for tasks B and C. Using the *optimistic* approach, we treat task C as an XOR-join task during the analysis for B. As a result, we can find a reachable marking $c1 + c3 + c6$, which marks both input conditions of B. Therefore, B should not be enabled at $c1 + c2$. Similarly, we will treat B as an XOR-join task for the analysis of task C and there is a reachable marking $c2 + c4 + c5$. Therefore, task C should not be enabled at $c1 + c2$. As a result of this *optimistic* approach, the YAWL net will *deadlock*. Using the *pessimistic* approach, we treat task C as an AND-join task during the analysis for B. At the marking $c1 + c2$, it is not possible to enable C due to the AND-join semantics, and therefore, task B will be enabled and can be fired, which yields the marking $c_2 + c_4 + c_5$. This will enable task C and after firing C, the marking $c_3 + c5 + c6$ results. Therefore, tasks B and C could potentially keep firing alternatingly thus resulting in a potentially infinite number of firings of task D. The same is true for the analysis of task C. We can see that the *pessimistic* approach would result in multiple tokens in the end condition. The original semantics that ignores other OR-joins would also result in a similar behaviour to the *pessimistic* approach. In this case, it is hard to see what formal semantics to choose and it is not possible to define the formal semantics accurately.

From the above discussions, it can be seen that there is no ideal treatment for non-local OR-join semantics in YAWL. Any formal semantics will impose some restrictions or deviate from the informal semantics to some extent. In some cases, we observe that treating other OR-joins on the path as XOR-joins using an optimistic approach is more

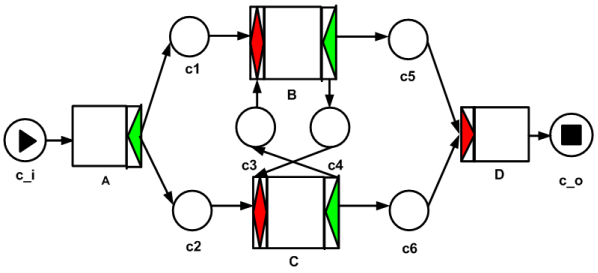

Fig. 9. OR-join tasks B and C in conflict

appropriate for the analysis. Consider a scenario where task C in Figure 8 is an XOR-split task rather than the OR-split task. Let us consider a marking $c1 + c3$ and treat task E as an AND-join task. As it is not possible for task E to fire due to the XOR-split and AND-join combination, the OR-join analysis will conclude that F should be enabled. As a result, task F could be executed more than once and the YAWL net does not have proper completion. The analysis will reach the same conclusion as the current semantics in YAWL where the semantics ignores the OR-join dependencies. Hence, we chose to use the optimistic approach (XOR-join treatment) for formal semantics during OR-join analysis.

2.4 Abstractions

We propose to abstract from the constructs in YAWL that do not affect an OR-join analysis. They include multiple instances, internal conditions of a task and composite tasks. We can assume that if a multiple instances task is enabled and executed, it will complete and put tokens into the appropriate output conditions of the task. Similarly, with the state transitions and internal conditions within a task, we can abstract from these transitions and only consider the input and output conditions of a task. We propose to treat a YAWL net as a *flat* net, and ignore the hierarchical structure for the purpose of an OR-join analysis. Composite tasks will be treated as black boxes during OR-join analysis. The assumption is that if a composite task can be enabled and executed, it will terminate at some time, and tokens will be placed in the appropriate output condition(s) of the composite task. As a result, even if there is an OR-join task in the composite task, it will not influence the decision to enable an OR-join task at a higher level. We recognise that due to the semantics of only considering tasks at the same level, the OR-join task could wait and result in a deadlock if a composite task is not sound and could deadlock. Because of the proposed abstractions from a YAWL net, we can now able to map to a Petri net like formalism.

3 Establishing a formal foundation

The formal semantics of YAWL is expressed in terms of a transition system [3] and while inspired by Petri nets, YAWL should not be seen as an extension of these. New

concepts were introduced in YAWL to suitably deal with some of the workflow patterns [4]. YAWL constructs such as the OR-join, cancellation and multiple instances are not directly supported by Petri nets. To perform an OR-join analysis, cancellation plays an important role (as shown in Figure 7). During an OR-join analysis, we are only required to consider the split and join behaviours of tasks and the cancellation set that is associated with a task. The cancellation feature of YAWL is theoretically closely related to reset nets, which are Petri nets with reset arcs. For an OR-join analysis, we propose to map a YAWL model represented as an EWF-net (Extended Workflow Net) to a reset net. In this section, we first present the definition of EWF-nets and we then present the definition and firing rules for reset nets.

3.1 EWF-nets

A YAWL specification is formally defined as a nested collection of EWF-nets [3]. A YAWL specification supports hierarchy and a composite task is mapped onto an EWF-net. As we will abstract from composition, it suffices to consider only one EWF-net in isolation when evaluating an OR-join. We refer the reader to [3] for a formal definition of a YAWL specification.

Definition 1 (EWF-net [3]). *An extended workflow net (EWF-net) N is a tuple* $(C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ *such that*[2]

- C *is a set of conditions and* T *is a set of tasks,*
- $\mathbf{i} \in C$ *is the unique input condition and* $\mathbf{o} \in C$ *is the unique output condition,*
- $F \subseteq (C \setminus \{\mathbf{o}\} \times T) \cup (T \times C \setminus \{\mathbf{i}\}) \cup (T \times T)$ *is the flow relation,*
- *every node in the graph* $(C \cup T, F)$ *is on a directed path from i to o,*
- *split:* $T \rightarrow \{AND, XOR, OR\}$ *specifies the split behaviour of each task and join:* $T \rightarrow \{AND, XOR, OR\}$ *specifies the join behaviour of each task,*
- *rem:* $T \nrightarrow \mathbb{P}(T \cup C \setminus \{\boldsymbol{i}, \boldsymbol{o}\})$ *specifies the cancellation region for a task;*
- *nofi:* $T \nrightarrow \mathbb{N} \times \mathbb{N}^{\boldsymbol{inf}} \times \mathbb{N}^{\boldsymbol{inf}} \times \{$*dynamic, static*$\}$ *specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).*

In an EWF-net, it is possible for two tasks to have a direct connection. We will add an implicit condition $c_{(t_1,t_2)}$ between two tasks t_1, t_2 if there is a direct connection between them. We denote as C^{ext} the set of conditions extended to include implicit conditions, and denote the extended flow relation as F^{ext}. We now define an explicit extended workflow net (E2WF-net) using C^{ext} and F^{ext} as follows:

Definition 2 (E2WF-net). *Let* $N = (C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ *be an EWF-net, the corresponding explicit EWF-net (E2WF-net) is defined as* $(C^{ext}, \mathbf{i}, \mathbf{o}, T, F^{ext}, split, join, rem, nofi)$ *where*

$$
\begin{aligned}
C^{ext} &= C \cup \{c_{(t_1,t_2)} \mid (t_1, t_2) \in F \cap (T \times T)\} \text{ and} \\
F^{ext} &= (F \setminus (T \times T)) \\
&\quad \cup \{(t_1, c_{(t_1,t_2)}) \mid (t_1, t_2) \in F \cap (T \times T)\} \\
&\quad \cup \{(c_{(t_1,t_2)}, t_2) \mid (t_1, t_2) \in F \cap (T \times T)\}.
\end{aligned}
$$

[2] Note that we are using basic mathematical notations such as $\nrightarrow$ for a partial function, $\mathbb{P}$ for powerset, $\mathbb{N}$ for natural numbers, and $\mathbb{N}^{inf}$ for $\mathbb{N} \cup \{inf\}$.

Let N be an E2WF-net and $x \in C^{ext} \cup T$, we use $\bullet x$ and $x\bullet$ to denote the set of inputs and outputs of a node i.e. $\bullet x = \{y|(y,x) \in F^{ext}\}$ and $x\bullet = \{y|(x,y) \in F^{ext}\}$. 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 $x \overset{N}{\bullet}$ to describe input and outputs of a node in N. A marking is denoted by M and, just as with ordinary Petri nets, it can be interpreted as a vector, function, and multiset. M is an m-vector, where m is the total number of conditions. Let $\mathcal{C}$ be all possible conditions and $M : C \to \mathbb{N}$, where $C \subseteq \mathcal{C}$. $M(c)$ returns the number of tokens in a condition c if $c \in dom(M)$ and zero otherwise. We can use notations such as $M \leq M'$, $M + M'$, and $M - M'$. $M \leq M'$ iff $\forall_{c \in \mathcal{C}} M(c) \leq M'(c)$. $M + M'$ and $M - M'$ are multisets such that $\forall_{c \in \mathcal{C}}$: $(M + M')(c) = M(c) + M'(c)$ and $(M - M')(c) = M(c) - M'(c)$.

3.2 Reset nets

A reset net is a Petri net with special reset arcs, that can clear the tokens in selected places. Reset arcs do not change the requirements of enabling a transition but when a transition fires, they will remove tokens from the specified places. The reset arcs are used to underpin the *rem* function that models the cancellation feature of EWF-nets, cf. Definition 1. This approach allows us to leverage existing literature and techniques in the area of Petri nets and reset nets in particular [5, 6, 9, 10, 12–14].

Definition 3 (Reset net). *A Petri net is a tuple (P, T, F) where P is a set of places, T is a set of transitions, $P \cap T = \varnothing$ and $F \subseteq (P \times T) \cup (T \times P)$. A reset net is a tuple (P, T, F, R) where (P, T, F) is a Petri net and $R \in T \nrightarrow \mathbb{P}(P)$ provides the reset places for certain transitions.*

Figure 10 shows a transition t with a number of places. Single-headed arrows represent input and output arcs and double-headed arrows represent reset arcs. For example, $p1$ is an input place to t, $p4$ is an output place of t, and $p2$ is a reset place of t and tokens will be removed from $p2$ when t fires. It is possible for a place to have different kinds of arcs associated with it. For instance, $p6$ is an input place, an output place and a reset place of t.

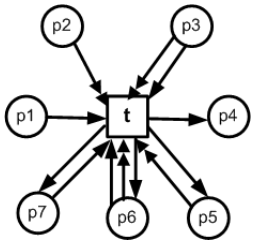

Fig. 10. An example transition with input, output and reset arcs

In the remainder of the paper, when we use the expression $F(x, y)$, it denotes 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.

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

Definition 4 ($\mathbb{M}(N)$). *Let $N = (P, T, F, R)$ be a reset net, then $\mathbb{M}(N) = P \to \mathbb{N}$.*

When a transition t of a reset net is enabled at a marking M', it can fire and a new marking M is reached.

Definition 5 (Forward firing). *Let $N = (P, T, F, R)$ be a reset net and $M' \in \mathbb{M}(N)$. A transition $t \in T$ is enabled iff $\bullet t \leq M'$* [3].

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

Definition 6 (Occurrence sequence). *Let $N = (P, T, F, R)$ be a reset net. Let $M_0, ..., M_n \in \mathbb{M}(N)$. Let $\sigma = t_0, t_1, ..., t_{n-1}$ be transitions in T. Sequence $s = M_0 t_0 M_1 ... t_{n-1} M_n$ is an occurrence sequence and σ is a firing sequence iff $M_i \xrightarrow{N,t_i} M_{i+1}$ for all $0 \leq i \leq n-1$.*

A marking M'' is reachable from a marking M, written $M \xrightarrow{N,*} M''$, iff there is an occurrence sequence with initial marking M and final/last marking M''. If there can be no confusion regarding the net, we will abbreviate it as $M \xrightarrow{*} M''$.

The notation $M[P']$ restricts M to a set of places P', i.e., a projection. For places not in P', the number of tokens is zero. Let $M_1 = p1 + p2 + p3$ and $P' = \{p1, p2\}$. $M_1[P'] = p1 + p2 + 0p3$ and dom$(M_1[P']) = \{p1 + p2 + p3\}$. Let $M_2 = p1 + 2p2$, $M_2[P'] > M_1[P']$ is true as the comparison between M and M' is restricted to a set of places in P' and M_2 has more tokens in $p2$.

Definition 7 (Projection). *Let $N = (P, T, F, R)$ be a reset net, $M \in \mathbb{M}(N)$ and $P' \subseteq P$. $M[P']$ returns a projection such that dom$(M[P']) =$ dom(M) and*

$$M[P'](p) = \begin{cases} M(p) & \text{if} \quad p \in P' \\ 0 & \text{if} \quad p \notin P'. \end{cases}$$

The notation $M{\restriction}P'$ is used to alter a marking based on a set of places P'. Let $M = p1 + p2 + p3$ and $P' = \{p1, p2\}$. $M{\restriction}P' = p1 + p2$ and dom$(M{\restriction}P') = \{p1 + p2\}$. If $P' = \{p1, p2, p3, p4\}$, $M{\restriction}P' = p1{+}p2{+}p3{+}0p4$ and dom$(M{\restriction}P') = \{p1, p2, p3, p4\}$.

Definition 8 (Filtering ${\restriction}$). *Let $N = (P, T, F, R)$ be a reset net, $M \in \mathbb{M}(N)$ and P' a set of places. $M{\restriction}P'$ returns a function such that dom$(M{\restriction}P') =$ dom(P') and*

$$M{\restriction}P'(p) = \begin{cases} M(p) & \text{if} \quad p \in (P' \cap \text{dom}(M)) \\ 0 & \text{if} \quad p \in (P' \setminus \text{dom}(M)). \end{cases}$$

The function *marked* returns the set of marked places in a reset net for a given marking M.

Definition 9 (Marked). *Let $N = (P, T, F, R)$ be a reset net and $M \in \mathbb{M}(N)$:* marked$(M) = \{p \in dom(M) \mid M(p) > 0\}$.

[3] 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 $\sqsubseteq$ relation indicates that M marks fewer or the same places as M'. This is a looser notion of smaller markings than $\leq$, because only the marking of places is considered and the number of tokens in a place is ignored. The notation $\sqsubset$ is used to indicate that M marks strictly less places than M'.

Definition 10 ($\sqsubseteq$). *Let M, M' be two markings of a reset net: $M \sqsubseteq M'$ iff marked(M) $\subseteq$ marked(M'), $M \sqsubset M'$ iff $M \sqsubseteq M'$ and $\neg(M' \sqsubseteq M)$.*

The function *superM* returns whether it is possible to reach a marking from M which marks more places in a set of places P'.

Definition 11 (superM). *Let $N = (P, T, F, R)$ be a reset net and $M \in \mathbb{M}(N)$ and $P' \subseteq P$ be a set of places for consideration, superM(N, M, P') holds iff there is a marking M' such that $M \xrightarrow{*} M'$ and $M[P'] \sqsubset M'[P']$.*

To conclude this section, we define the notion of backward firing. This notion will be used to analyse coverability and is required for the OR-join analysis as is described in the remainder of this paper.

Definition 12 (Backward firing). *Let (P, T, F, R) be a reset net and $M, M' \in \mathbb{M}(N)$. $M' \dashrightarrow^t M$ if and only if it is possible to fire a transition t backwards starting from M and resulting in M'.* [4]

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

For 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 $M(p) = F(t,p)$ 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 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.

4 Linking YAWL to Reset nets

In this section, we describe how an E2WF-net could be transformed into a reset net. After the abstractions from multiple instances, composite tasks and internal places in a YAWL net, we can consider a YAWL net as having tasks with various split and join behaviours, possible cancellation sets and explicit and implicit conditions. For an E2WF-net without OR-join tasks, there is then a straight forward mapping into a reset net. For an E2WF-net with one or more OR-join tasks, it is not possible to directly transform the net into an equivalent reset net as non-local semantics of OR-join task must be taken into account. For an E2WF-net with OR-join tasks, we propose to use the *optimistic* treatment whereby other OR-joins are replaced with XOR-joins, and perform the necessary transformation to obtain one reset net per OR-join. So, if a YAWL net has two OR-joins, there will be two corresponding reset nets, one for each OR-join.

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

4.1 Transformation of an E2WF-net without OR-joins into a reset net

For every task t in an E2WF-net, t is split into two, t_{start} and t_{end} to support XOR, OR and AND split constructs and XOR and AND join constructs. The number of t_{start} transitions depends on the join behaviour of a task and the number of t_{end} transitions depends on the split behaviour. Figure 11 illustrates the approach taken in the transformation. The function transE2WF transforms an E2WF-net without OR-joins into a reset net. To support the cancellation feature in YAWL, we have a relation R where t_{end} and its associated reset places are stored. As a transition in an E2WF-net is now split into a number of t_{start} and t_{end} transitions depending on the split and join behaviour, we also introduce a place p_t for each transition t to represent an internal place between t_{start} and t_{end}. The flow relation F' is also modified so that the newly introduced places in P' and transitions T' are properly connected.

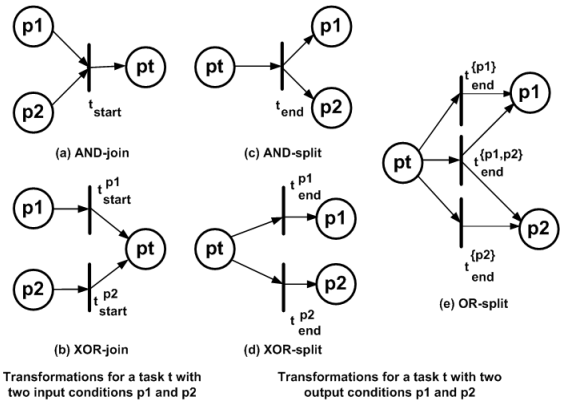

Fig. 11. Reset net transformations for YAWL split and join behaviours

Definition 13 (transE2WF). *Let $N = (C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ be an E2WF-net without OR-joins. The function transE2WF(N) returns $N' = (P, T', F', R)$ such that*

$P = C \cup \{p_t | t \in T\}$ *is a set of places,*
$T' = T_{start} \cup T_{end}$ *such that*
$T_{start} = \{t_{start} | t \in T \ \wedge \ join(t) = AND\}$
$\quad \cup \{t^p_{start} | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in {\bullet t}\}$,
$T_{end} = \{t_{end} | t \in T \ \wedge \ split(t) = AND\}$
$\quad \cup \{t^p_{end} | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t{\bullet}\}$
$\quad \cup \{t^x_{end} | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing\}$,
$F' = \{(p, t_{start}) | t \in T \ \wedge \ join(t) = AND \ \wedge \ p \in {\bullet t}\}$
$\quad \cup \{(t_{start}, p_t) | t \in T \ \wedge \ join(t) = AND\}$
$\quad \cup \{(p_t, t_{end}) | t \in T \ \wedge \ split(t) = AND\}$
$\quad \cup \{(t_{end}, p) | t \in T \ \wedge \ split(t) = AND \ \wedge \ p \in t{\bullet}\}$

$\cup\{(p, t^p_{start})|t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}$
$\cup\{(t^p_{start}, p_t)|t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}$
$\cup\{(p_t, t^p_{end})|t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\}$
$\cup\{(t^p_{end}, p)|t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\}$
$\cup\{(p_t, t^x_{end})|t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing\}$
$\cup\{(t^x_{end}, p)|t \in T \ \wedge \ split(t) = OR \ \wedge \ p \in x \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing\}$,

$R \in T_{end} \nrightarrow \mathbb{P}(P)$ *such that*

$t \in T \wedge split(t) = AND$
$\Rightarrow R(t_{end}) = \{p_{t'}|t' \in rem(t) \cap T\} \cup (rem(t) \cap C)$,
$t \in T \wedge split(t) = XOR \ \wedge \ p \in t\bullet$
$\Rightarrow R(t^p_{end}) = \{p_{t'}|t' \in rem(t) \cap T\} \cup (rem(t) \cap C)$,
$t \in T \wedge x \subseteq t\bullet \wedge x \neq \varnothing \ \wedge \ split(t) = OR$
$\Rightarrow R(t^x_{end}) = \{p_{t'}|t' \in rem(t) \cap T\} \cup (rem(t) \cap C)$.

4.2 Transformation of an E2WF-net with OR-joins into a reset net

Here, we propose to transform an E2WF-net with OR-join tasks into a reset net by first singling out one OR-join and removing it from the net and then by treating other OR-join tasks as XOR-joins. The same transformation rules defined in Definition 13 can then be used for transformation as the net now does not contain any OR-join tasks. This effectively converts a YAWL net with OR-joins into a reset net for a given OR-join.

Definition 14 (transE2WFOJ). *Let N be an EWF-net with OR-joins and N^{ext} be the E2WF-net of N and j be an OR-join task under consideration. The function transE2WFOJ(N, j) returns $N' = (P, T'', F'', R)$ such that P, T', T_{start}, T_{end}, F', and R are as defined in Definition 13 and T'' and F'' are defined as follows:*

$T'' = T'_{start} \cup T_{end}$,
$T'_{start} = T_{start} \ \cup \ \{t^p_{start}|t \in T \wedge \ join(t) = OR \wedge t \neq j \ \wedge \ p \in^N_\bullet t\}$, *and*
$F'' = F' \cup \{(p, t^p_{start})|p \in^N_\bullet t \ \wedge \ t \in T \wedge join(t) = OR \wedge t \neq j\}$
$\cup\{(t^p_{start}, p_t)|p \in^N_\bullet t \ \wedge \ t \in T \wedge join(t) = OR \wedge t \neq j\}$.

We now define how a given marking M in an E2WF-net can be linked to a marking M_R in the corresponding reset net. For all the conditions that exist in an E2WF-net, they will be marked exactly the same in M_R and zero tokens for the newly introduced places in the reset net.

Definition 15 (M_R). *Let (N, M) be a marked E2WF-net and N_R be the corresponding marked reset net of N, then M corresponds in a natural way to a marking of N_R. This marking marks all the places in N_R which correspond to conditions in N with the same number of tokens. We will refer to this as the corresponding marking and denote it as M_R.*

4.3 OR-join semantics

For an OR-join task *o-j* of an E2WF-net, the enabling rule for *o-j* is defined by first translating the net into a reset net using optimistic approach in Definition 14 and then

deciding whether it is possible to mark more input places of *o-j* from a given marking. This can be done using the *superM* function. If *superM* holds then the OR-join, *o-j*, should not be enabled at M. Otherwise, *o-j* is enabled at M.

Definition 16 (OR-join semantics). *Let $N = (P, T, F, R)$ be an E2WF-net, M be a marking of N, o-j be the OR-join task under consideration, $N_R = transE2WFOJ(P, T, F, R, o\text{-}j)$ be the corresponding reset net and $M_R \in \mathbb{M}(N_R)$. o-j is enabled at M iff at least one of its input places is marked and superM$(N_R, M_R, \bullet o\text{-}j)$ does not hold.*

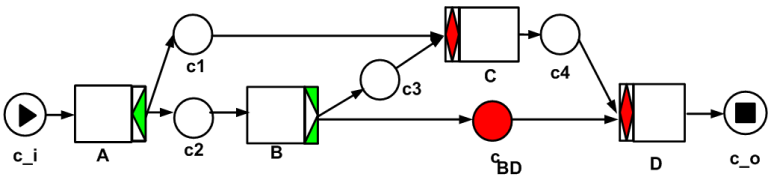

Fig. 12. An E2WF-net N with OR-join tasks C and D

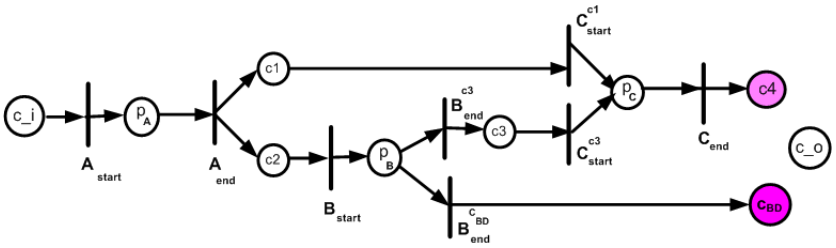

Fig. 13. A reset net for OR-join analysis of task D in Figure 12

We will now describe how the transformations will be performed for an EWF-net with two OR-join tasks C and D as shown in Figure 12. Note that an explicit condition c_{BD} has been added for the implicit condition between tasks B and D. Figure 13 shows an equivalent reset net for the E2WF-net in Figure 12 for OR-join analysis of task D. The OR-join task C is on the path to task D and task C is treated as an XOR-join task. Also note that OR-join task D has been removed from the net. Consider a marking $M = c1 + c_{BD}$ of N where OR-join analysis for task D would be performed. There is a corresponding marking for the reset net, $M_R = c1 + c_{BD}$. The input places of task D are $c4$ and c_{BD}. We need to investigate whether it is possible to reach a marking that marks both $c4$ and c_{BD}. The sequence $c1 + c_{BD} \overset{C^{c1}_{start}}{\rightarrow} p_C + c_{BD} \overset{C_{end}}{\rightarrow} c4 + c_{BD}$ exists and that it is possible to reach $M'' = c4 + c_{BD}$ from M. Therefore, superM(transE2WFOJ$(P, T, F, R, o\text{-}j), M_R, \bullet o\text{-}j)$ predicate holds as $M_R \overset{*}{\rightarrow} M''$ and $M_R[\{c4, c_{BD}\}] \sqsubset M''[\{c4, c_{BD}\}]$. The OR-join analysis for task D will conclude that D should not be enabled at marking M as it is possible to reach a marking from M that marks more input places of the OR-join than M does.

5 OR-join algorithm proposal

The main objective of the OR-join algorithm is to determine, for a given OR-join, whether or not a marking M' is reachable from a given marking M that marks more input places of that OR-join. We perform this analysis by first transforming an EWF-net (with OR-joins) into a reset net for a given OR-join task using the function transE2WFOJ and then calling our proposed OR-join algorithm. Our algorithm is based on backward search techniques for Well-Structured Transition Systems (WSTSs) [5, 9, 12–14]. The algorithm works backwards by computing the predecessor markings for a given marking, as opposed to the forward approach used in coverability tree algorithms. A reset net can be represented as a WSTS and the backwards algorithm has been successfully applied to solve the coverability problems for reset nets [9, 19].

5.1 Backward algorithm for OR-join analysis

WSTSs are "a general class of infinite state systems for which decidability results rely on the existence of a well-quasi-ordering between states that is compatible with the transitions." [14]. The existence of a well-quasi-ordering over an infinite set of states ensures the decidability of termination and coverability properties [9, 14].

Definition 17 (Well-Structured Transition System [9]). *A well-structured transition system (WSTS) is a structure $S = \langle Q, \rightarrow, \leq \rangle$ such that Q is a set of states, $\rightarrow \subseteq Q \times Q$ is a set of transitions, $\leq \subseteq Q \times Q$ is a well-quasi-ordering (wqo) on the set of states, satisfying the simple monotonicity property, $m \rightarrow m'$ for markings $m, m' \in Q$ and $m_1 \geq m$ imply $m_1 \rightarrow m_1'$ for some $m_1' \geq m'$.*

Reset nets can be seen as a WSTS $\langle Q, \rightarrow, \leq \rangle$ with Q the set of markings, $M \rightarrow M'$ if for some t, we have $M \xrightarrow{t} M'$ and $\leq$ the corresponding $\leq$ order on markings (which is a wqo) [19].

Definition 18 (Upward-closed set [14]). *Given a quasi-ordering $\leq$ on X, an upward-closed set is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. To any $x \in X$ we associate $\uparrow x =^{def} \{y | y \geq x\}$. A basis of an upward-closed I is a set I^b such that $I = \bigcup_{x \in I^b} \uparrow x$.*

Given a WSTS $\langle Q, \rightarrow, \leq \rangle$ and a set of states $I \subseteq Q$, $Pred(I)$, $pb(I)$ and $Pred^*(I)$ can be defined [19]. The immediate predecessors of I: $Pred(I) = \{x | x \rightarrow y \wedge y \in I\}$, all predecessor states of I, $Pred^*(I) = \{x | x \xrightarrow{*} y \ \wedge \ y \in I\}$ and $pb(I) = \bigcup_{y \in I} pb(y)$ where $pb(y)$ yields a finite basis of $\uparrow Pred(\uparrow\{y\})$ (i.e., $pb(y)$ yields a finite set such that $\uparrow pb(y) = \uparrow Pred(\uparrow\{y\})$) [19].

In the context of reset nets, we use the backward firing rule (cf. Definition 12) to define pb.

Definition 19 (pb). *Let (N, M) be a marked reset net. $pb(M) = \{M' | \exists_{t \in T} \ M' \dashrightarrow^t M\}$.*

Lemma 1. *Let (N, M) be a marked reset net. $\uparrow pb(M) = \uparrow Pred(\uparrow\{M\})$.*

Proof. First, we will prove that $\uparrow pb(M) \subseteq \uparrow Pred(\uparrow\{M\})$.
Let $M_1 \in \uparrow pb(M)$, we need to show that $M_1 \in \uparrow Pred(\uparrow\{M\})$. There is an $M_2 \leq M_1$ such that $M_2 \in pb(M)$. Therefore, there exists a $t \in T$ such that $M_2 \dashrightarrow^t M$. We will show that this implies that there is an M_3 such that $M_3 \geq M$ and $M_2 \xrightarrow{t} M_3$. The markings M, M_1, M_2 and M_3 with the associated firing rules are shown in Figure 14. M is described as the relationship between input, output and reset arcs of transition t. Firing transition t backwards at M results in M_2. A token will be placed into each input place of t. For instance, an input place of t that has x number of tokens will now has $x + 1$. The same is true of any output place of t with y number of tokens. At M_2, the number of tokens is reduced by 1 (if possible), i.e. $\max(y - 1, 0)$. We use the max function to ensure that the negative numbers are avoided. The same is true if the input place is also an output place. If it has z tokens before, now it will have $\max(z, 1)$. A reset place will have zero token. If a reset place is also an input place of t, it will have one token. If a reset place is also an output place, it will have zero token. If a reset place is also an input place as well as the output place, that place will have one token. By firing a transition t at M_2, we can reach a new marking M_3. One token is removed from each input place of t in M_2, one token is put into each output place of t and the tokens are removed from the reset places. Hence, we can conclude that $M_3 \in \uparrow\{M\}$, $M_2 \in Pred(\uparrow\{M\})$, and $M_1 \in \uparrow Pred(\uparrow\{M\})$.

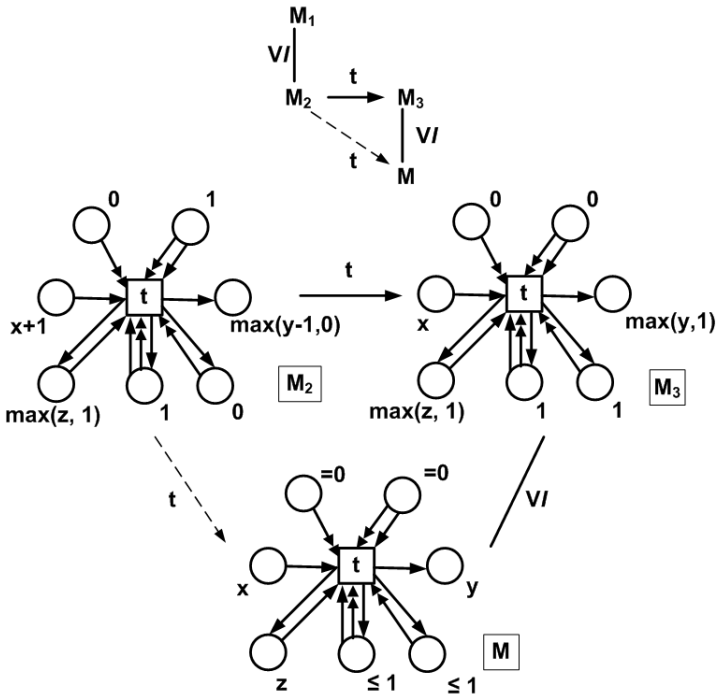

Fig. 14. Sketch of the first part of the proof

Second, we will prove that $\uparrow Pred(\uparrow\{M\}) \subseteq \uparrow pb(M)$.
Let $M_1 \in \uparrow Pred(\uparrow\{M\})$, we need to show that $M_1 \in \uparrow pb(M)$. This is shown in Figure 15. There is a $M_2 \leq M_1$ such that $M_2 \in Pred(\uparrow\{M\})$. Hence, there is an $M_3 \geq M$ such that $M_2 \xrightarrow{t} M_3$. We will show that this implies that there is a M_4 such that $M_2 \geq M_4$ and $M_4 \dashrightarrow^t M$. Such a marking M_4 can be constructed as shown in Figure 15. We can see that $M \leq M_3$ as $x' \leq x - 1$, $z' \leq z$ and $y' \leq y + 1$. Note that indeed $M_4 \leq M_2$. Clearly: $x' + 1 \leq x$ (because $x' \leq x - 1$), $\max(z', 1) \leq z$ (because $z' \leq z$ and $z \geq 1$) and $\max(y' - 1, 0) \leq y$ (because $y' \leq y + 1$ and $y \geq 0$). Since, $M_4 \in pb(M)$ and $M_1 \geq M_4$, we conclude $M_1 \in \uparrow pb(M)$.

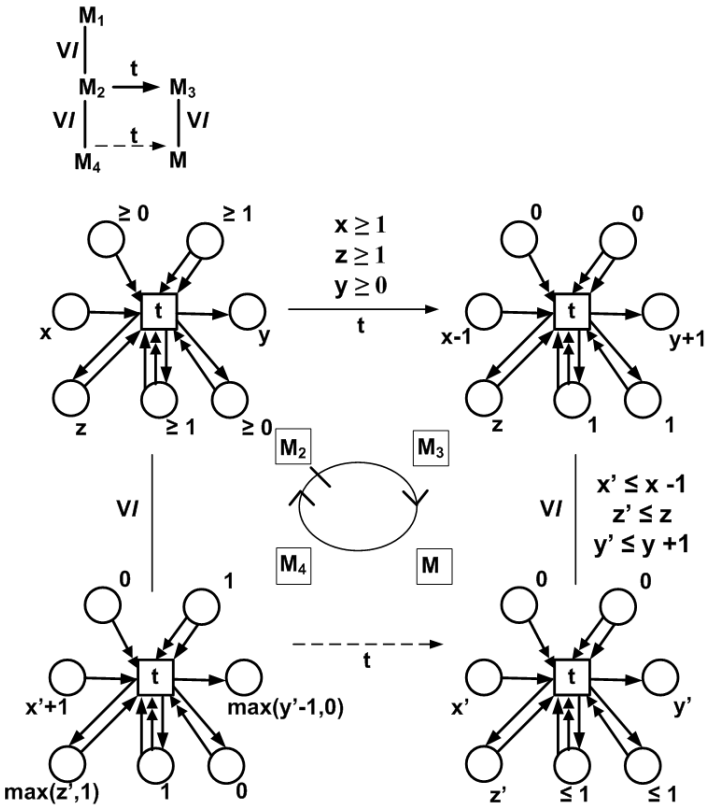

Fig. 15. Sketch of the second part of the proof

The coverability problem for a reset net is as follows: given two markings x and y, can we reach $y' \geq y$ starting from x [19]. The backwards reachability analysis can be performed to decide the coverability [9, 12, 19] provided that $\leq$ is decidable and $pb(y)$ exists and can be effectively computed [14]. A finite basis of $Pred^*(\uparrow\{y\})$ is computed as the limit of the sequence $I_0 \subseteq I_1 \subseteq ...$ where $I_0 =^{\text{def}} \{y\}$ and $I_{n+1} =^{\text{def}}$

$I_n \cup pb(I_n)$ [19]. The sequence eventually stabilises at some I_n when $\uparrow I_{n+1} = \uparrow I_n$ and we have reached a stabilisation point that has the property $\uparrow I_n = Pred^*(\uparrow\{y\})$ [19]. As $\uparrow\{y\}$ is upward-closed, $Pred^*(\uparrow\{y\})$ is upward-closed [14]. The coverability question now becomes: is there an $x' \in \uparrow I_n$ such that $x' \leq x$. $\{y\}$ is a basis of upward closed set $\uparrow\{y\}$ and we can determine that y is coverable from x if there exists a $x' \in Pred^*(\uparrow\{y\})$ such that $x' \leq x$ (because $\leq$ is a wqo).

We now present the procedures that operationalise the coverability question for reset nets. The procedure **Coverable** returns a Boolean value to indicate whether a marking y is coverable from a marking x of a reset net [19].

PROCEDURE Coverable (Marking x, y): Boolean
Marking x';
BEGIN
  for $x' \in \mathbf{FiniteBasisPred}^*(\{y\})$ **do**
    if $x' \leq x$ **then return** TRUE; **end if**;
  end for;
  return FALSE;
END

The procedure $\mathbf{FiniteBasisPred}^*$ returns a set of markings which represents a finite basis of all predecessors and is based on the method described in [19].

PROCEDURE FiniteBasisPred* (SET Marking I): SET Marking
SET Marking K, K_{next};
BEGIN
  $K := I$; $K_{next} := K \cup \mathbf{pb}(K)$;
  while not IsUpwardEqual(K, K_{next}) **do**
    $K := K_{next}$; $K_{next} := K \cup \mathbf{pb}(K)$;
  end while;
  return K;
END

The procedure call $\mathbf{IsUpwardEqual}(K, K_{next})$ is used to detect whether the stabilisation point has been reached i.e. $\uparrow K_{next} = \uparrow K$, cf. [13].

PROCEDURE IsUpwardEqual (SET Marking K, SET Marking K_{next}): Boolean

BEGIN
  return $K = K_{next}$;
END

The procedure $\mathbf{pb}(I)$ returns $pb(I)$ such that $pb(I) = \bigcup_{x \in I} pb(x)$ [19].

PROCEDURE pb (SET Marking I): SET Marking
SET Marking $Z = \varnothing$; Marking M;
BEGIN
  for $M \in I$ **do** $Z := Z \cup \mathbf{pb}(M)$; **end for**;
  return Z;
END

$\mathbf{pb}(M)$ is effectively computed for reset nets by "executing the transitions backwards and setting a place to the minimum number of tokens required to fire the transition if it caused a reset on this place" [19].[5] Note that, in our case, this minimum is one as we do not have weighted arcs. We will make use of backward firing rule as defined in Definition 12. For each transition $t \in T$, we determine whether an M' exists such that $M' \dashrightarrow^t M$. Hence, $pb(M) = \{M' | \exists_{t \in T}\ M' \dashrightarrow^t M\}$.

PROCEDURE pb (Marking M): SET Marking
SET Marking $Z = \varnothing$;
BEGIN
  for $t \in T$ **do**
    if $M[R(t)] \leq t\bullet\ [R(t)]$ **then**
      $Z := Z \cup \{((M \dot{-} t\bullet) + \bullet t)[P \setminus R(t)] + (M + \bullet t)[R(t)]\}$;
    end if;
  end for;
  return Z;
END

We can then apply the coverability findings of a reset net to the OR-join analysis. Let (N, M) be a marked E2WF-net, *o-j* be the OR-join task under consideration, X be $\bullet$*o-j*, N' be the corresponding reset net and Y be a set of markings such that each marking in Y has only one token in each of the marked input places of *o-j* in M and one token in exactly one of the unmarked input places of the *o-j* in M. To determine whether *o-j* should be enabled at M, we need to determine whether there exists a $M' \in Pred^*(M_w)$ such that $M' \leq M$ for each of the markings $M_w \in Y$ (coverability question). Each marking M_w in Y satisfies the condition $M[X] \sqsubset M_w[X]$, i.e. M_w has tokens in more input places of the OR-join *o-j* and if M_w can be reached from M, the OR-join is not enabled. The procedure $\mathbf{OrJoinEnabled}$ is called with parameters M and X and it returns a Boolean value to indicate whether *o-j* should be enabled at M.

PROCEDURE OrJoinEnabled (Marking M, SET Place X): Boolean
SET Marking Y; Marking M_w;
BEGIN
  $Y = \{q + \sum_{p \in X : M(p) > 0} p \mid q \in X\ \wedge\ M(q) = 0\}$;
  for $M_w \in Y$ **do**
    if $\mathbf{Coverable}(M, M_w)$ **then return** FALSE; **end if**;
  end for;
  return TRUE;
END

5.2 Worked examples

Throughout the paper we have shown several examples where it is a non-trivial task to decide if an OR-join is enabled or not. Clearly, the algorithm can be applied successfully

[5] Note that the algorithm described in [19] is incorrect. On Page 105 in [19], $pb(M)$ is defined in a rather naive way. Applying $pb(M)$ to the empty marking yields a counter example, since it is not a finite basis for $\uparrow Pred^*(\uparrow\{M\})$.

to these situations. To illustrate its inner working in some detail we use some more examples.

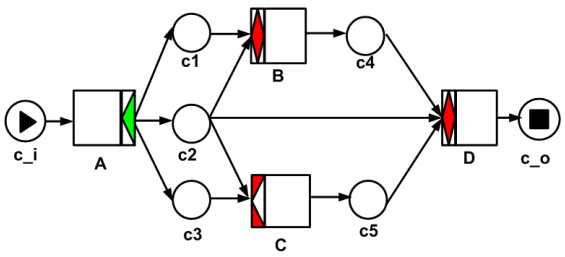

Fig. 16. A YAWL net with OR-join tasks B and D

In Figure 16, we have a YAWL net with an AND-split task A, an XOR-join task C and two OR-join tasks, B and D. At marking $M = c1 + c2 + c3$, OR-join evaluation for both tasks B and D will be performed. Task B will be enabled at M as all the input conditions of B are marked. The evaluation for task D starts with a call to the procedure $\mathbf{OrJoinEnabled}(c1+c2+c3, \{c2, c4, c5\})$. The set of markings $Y := \{c2+c4, c2+c5\}$. The finite basis of all predecessors for the marking $c2 + c4$ contains a marking $c2+c3$ as task C can be fired backwards and put a token into in $c3$. $c2+c3 \leq c1+c2+c3$ and as a result, the procedure will return FALSE, concluding that the OR-join task D should not be enabled at M.

Now consider another marking M_1 where the XOR-join task removes a token from $c2$ and places a token in $c5$ at M i.e. $c1 + c2 + c3 \xrightarrow{C} c1 + c3 + c5$. At M_1, the OR-join evaluations for tasks B and D ill be performed. The evaluation for B will start with a call to the procedure $\mathbf{OrJoinEnabled}(c1 + c3 + c5, \{c1, c2\})$. As it is not possible to put into $c2$ from reachable markings of M_1, B will be enabled at M_1. The evaluation for D will start with a call to the procedure $\mathbf{OrJoinEnabled}(c1 + c3 + c5, \{c2, c4, c5\})$. During the evaluation, OR-join task B will be treated as an XOR-join and a corresponding reset net will be generated. $Y := \{c4 + c5, c2 + c5\}$. The finite basis of all predecessors for marking $c4 + c5$ contains $c1 + c5$ (the resulting marking from backward firing of task B). As $c1+c5 \leq c1+c3+c5$, the algorithm will conclude correctly that $c4+c5$ is a reachable marking from M_1 and D will not be enabled at M_1.

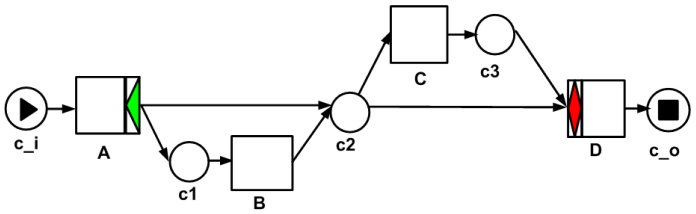

Fig. 17. A YAWL net with OR-join tasks B and D

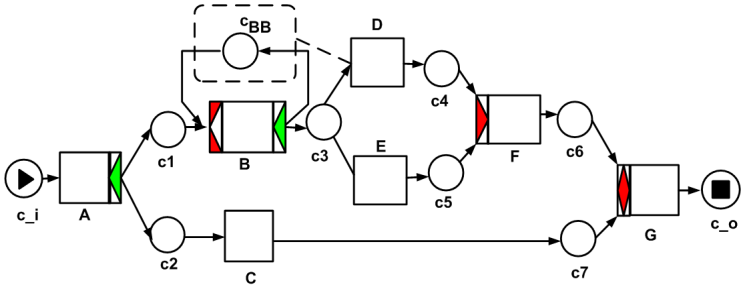

Fig. 18. A YAWL net with an OR-join task G and cancellation

In Figure 17, we have a YAWL net with an OR-join task D with input conditions $c2$ and

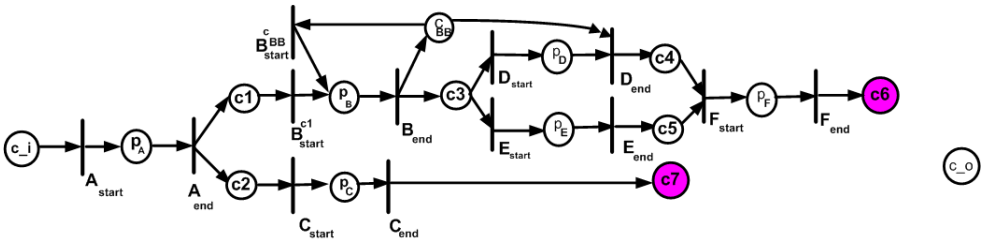

Fig. 19. A corresponding reset net for Figure 18 (note the double-headed arrow denoting the reset arc from C_{BB} to D_{end})

$c3$. The interesting aspect of this net is that task C needs to be executed to put tokens in $c3$ and this requires removal of tokens from $c2$. Consider a marking $M = c1 + c2$ where an OR-join evaluation will be carried out. The evaluation will start with a call to the procedure $\mathbf{OrJoinEnabled}(c1 + c2, \{c2, c3\})$. $Y := \{c2 + c3\}$. The finite basis of predecessors will contain $\{2c2, 2c1, c1 + c3, c1 + c2\}$. As M is in the predecessors of $c2 + c3$, D will not be enabled at M.

Consider a marking $M = c1 + c7$ in Figure 18 where the OR-join analysis for task G is carried out. It is possible to have an occurrence sequence, $c1 + c7 \xrightarrow{B} c_{BB} + c3 + c7 \xrightarrow{E} c_{BB} + c5 + c7 \xrightarrow{B} c_{BB} + c3 + c5 + c7 \xrightarrow{D} c4 + c5 + c7 \xrightarrow{F} c6 + c7$. As a result, $c6 + c7$ is a reachable marking from $c1 + c7$ and the OR-join should not be enabled at marking M. The evaluation will start with a call to the procedure $\mathbf{OrJoinEnabled}(c1 + c7, \{c6, c7\})$. $Y := \{c6 + c7\}$ and for $M_w = c6 + c7$, we will obtain a finite basis of all the predecessors of $c6 + c7$. Figure 20 illustrates the backwards reachability analysis [13], with the basis of the predecessor markings for $c6 + c7$. It can be seen that $c1 + c7$ is a predecessor of $c6 + c7$ and hence the OR-join procedure will return FALSE.

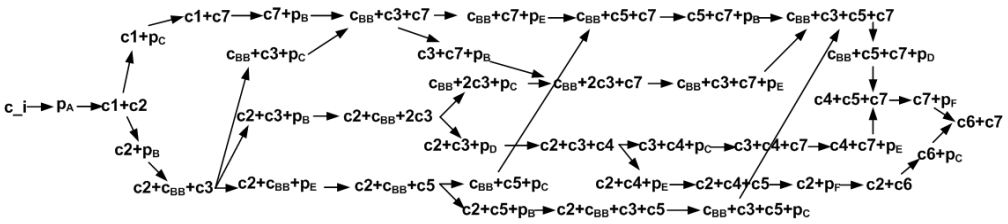

Fig. 20. Illustration of backwards reachability analysis

6 Restriction techniques

For an OR-join analysis, we can consider only a portion of the YAWL net that is relevant to the analysis and refrain from exploring those paths that do not affect the OR-join enabling behaviour. To improve the performance of OR-join evaluation algorithm, we propose here two forms of restriction: *structural restriction* and *active projection*. *Structural restriction* involves removing tasks and conditions in a YAWL net that are not on the path to the OR-join task under consideration. *Active projection* involves removing tasks and associated conditions that could not be enabled from a given marking. Active projection enables us to stop exploring those parts of the YAWL net that can never be reached from a given marking. As a YAWL net with OR-join tasks is translated into a reset net for OR-join analysis, the restriction operations will also be performed on the reset net. We exploit the translations to reset net as proposed in Section 3 and define how restriction operations are applied to a reset net.

6.1 Structural restriction

The application of structural restriction involves removing tasks and conditions from a YAWL net that are not on the path to a given OR-join task. As we are interested in whether more tokens could arrive in the input places of an OR-join task, the restriction will be based on those input places of an OR-join task. Let's call them a set of *goal places*. The function $\mathbf{res}$ describes how a reset net could be constructed so that only the transitions and places that are on the path to a set of goal places are included in the restricted net.

Definition 20 (res(N, G)). *Let $N = (P, T, F, R)$ be a reset net and $G \subseteq P$ a set of goal places. $N' = (P', T', F', R')$ is the restriction on G ($N' = res(N, G)$) where*

$P' = \{p \in P | \exists_{p' \in G}(p, p') \in F^*\}$,
$T' = \{t \in T | \exists_{p' \in G}(t, p') \in F^*\}$,
$F' = F \cap ((P' \times T') \cup (T' \times P'))$, *and*
$R' = \{(t, R(t) \cap P') | t \in dom(R) \cap T'\}$.

Note that N' is again a reset net, P' is a siphon, and $G \subseteq P'$. Hence, we can use firing rules and other functions defined for reset nets.

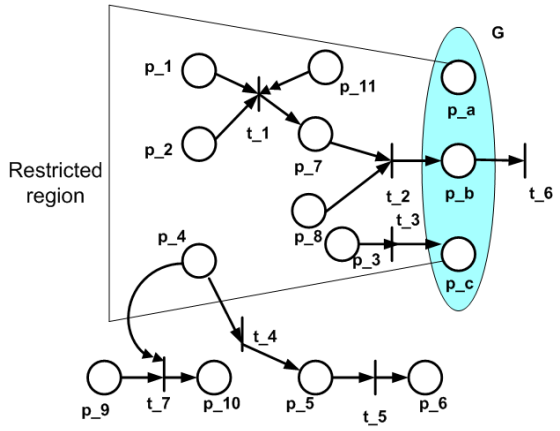

Fig. 21. Restriction diagram

Figure 21 describes how **res** function works with a set of goal places $G = \{p_a, p_b, p_c\}$. In the restricted region, all places and transitions which are on the path to G are included (e.g. p_1, p_2, t_1, t_2). On the other hand, places and transitions that are not on the path to G such as p_5, p_6, t_5, t_6 are not included in the restricted net. Also note that if a transition is in the restricted net, all its input places are also in the restricted net (e.g. p_1, p_2, t_1). It is possible for places in the restricted net to be input places of transitions that are not in the restricted net (e.g. p_4, t_4). A transition that is not in the restricted net cannot put tokens back into the restricted net (e.g. p_8 and t_4). In terms of reset arcs, R' will keep track of the reset places in P' for transitions that are in T' (e.g. p_{11}, t_1). However, we do not keep track of reset arcs for places that are in the restricted region but the transition is not in T' (e.g. p_4, t_7).

Let N, N' be two reset nets such that $N' = \text{res}(N, G)$, the structurally restricted net w.r.t. G. Lemma 2 will show that for any marking $M_2 \in \mathbb{M}(N)$ such that $M_1 \overset{N,*}{\rightarrow} M_2$, there is a corresponding marking $M_2^{'} \in \mathbb{M}(N')$ such that $M_1 \restriction P' \overset{N',*}{\rightarrow} M_2'$ and $M_2^{'} \geq M_2 \restriction P'$. That is, $M_2^{'}$ is larger than or equal to M_2 w.r.t. P'.

Lemma 2. *Let $N = (P, T, F, R)$ be a reset net, $G \subseteq P$ and $N' = res(N, G) = (P', T', F', R')$ is the restriction on G.*

$$\forall_{M_1, M_2 \in \mathbb{M}(N)} (M_1 \overset{N,*}{\rightarrow} M_2 \Rightarrow \exists_{M_2' \in \mathbb{M}(N')} M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \wedge M_2^{'} \geq M_2 \restriction P')$$

Proof. Consider a firing sequence $\sigma : M_1 \overset{N,\sigma}{\rightarrow} M_2$. Let $\sigma^{'}$ be the projection on T'.

First, we will prove that $\sigma^{'}$ is enabled in (N, M_1) and in $(N', M_1 \restriction P')$. From Definition 20, $t \notin T'$ implies that $t \bullet \cap P' = \varnothing$. Transitions that are not in the restricted net but in the firing sequence σ, i.e. $t \in \sigma$ and $t \notin \sigma^{'}$, can only remove tokens from P' and cannot put tokens into P'. Therefore, these transitions have no effect on the enabling behaviour of transitions in $\sigma^{'}$. As $\forall_{t \in \sigma'} \bullet t \subseteq P'$ and $\forall_{t \notin \sigma'} t \bullet \cap P' = \varnothing$, if σ is enabled in (N, M_1), $\sigma^{'}$ is also enabled in (N, M_1). Similarly, as $t \in (T \setminus T')$ cannot put tokens into places in P' in the restricted net, $\sigma^{'}$ is enabled in $(N', M_1 \restriction P')$.

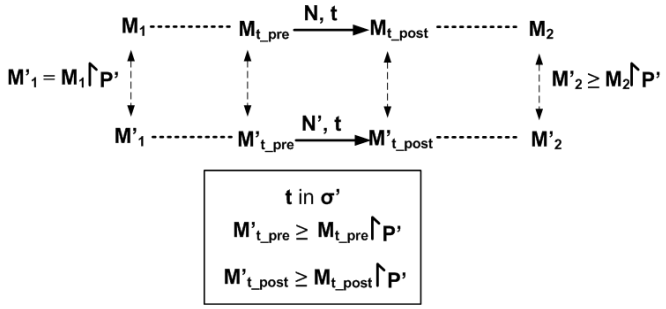

Fig. 22. Transition firings in both the original net and the restricted net

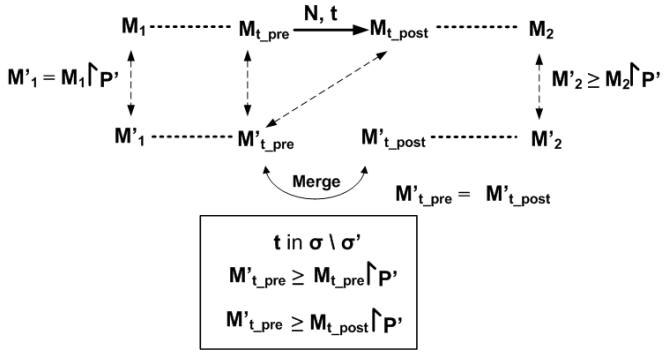

Fig. 23. Transition firings in the original net only

Next, we will prove that there exists $M_2^{''} \in \mathbb{M}(N)$ and $M_2^{'} \in \mathbb{M}(N')$ such that $M_1 \stackrel{N,\sigma^{'}}{\rightarrow} M_2^{''} : (M_1{\restriction}P' \stackrel{N',\sigma^{'}}{\rightarrow} M_2^{'}) \wedge (M_2^{''}{\restriction}P' = M_2^{'})$. As shown before, $\sigma^{'}$ is enabled in $(N', M_1{\restriction}P')$. Figure 22 gives the states in both models for transitions that can be fired in both nets (N, N'). Assume that $M_{t-pre}{\restriction}P' \geq M_{t-pre}^{'}$ and $t \in T'$. As $\stackrel{N'}{\bullet} t = \bullet^N t$, $t\stackrel{N'}{\bullet} = t\stackrel{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post}{\restriction}P' = M_{t-post}^{'}$. The effect of firing t is identical on the places in P'. Hence, the marking resulting from $\sigma^{'}$ is at least as large as M_2 w.r.t. P'. Figure 23 gives the states in both models for transitions that can only be fired in the net N. Assume that $M_{t-pre}{\restriction}P' \geq M_{t-pre}^{'}$ and $t \notin T'$. Since the effect of firing t can only remove tokens from places in P' and we do not have a corresponding marking M_{t-post} in N', we deduce $M_{t-pre}^{'} \geq M_{t-post}^{'}{\restriction}P'$.

Lemma 3 will demonstrate that for any marking $M_2^{'} \in \mathbb{M}(N')$ reachable from $M_1{\restriction}P'$, there is a corresponding marking $M_2 \in \mathbb{M}(N)$ reachable from M_1 such that $M_2^{'} = M_2{\restriction}P'$. That is, the two markings are the same w.r.t P'.

Lemma 3. *Let $N = (P, T, F, R)$ be a reset net and $N' = res(N, G) = (P', T', F', R')$ is the restriction on G.*

$\forall_{M_1 \in \mathbb{M}(N), M_2' \in \mathbb{M}(N')}(M_1{\restriction}P' \stackrel{N',*}{\rightarrow} M_2^{'} \Rightarrow \exists_{M_2 \in \mathbb{M}(N)} M_1 \stackrel{N,*}{\rightarrow} M_2 \wedge M_2{\restriction}P' = M_2^{'})$

Proof. Consider a firing sequence $\sigma : M_1 \restriction P' \overset{N',\sigma}{\rightarrow} M_2^{'}$. We first show that σ is enabled in (N, M_1) and then that there is marking $M_2 : M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_2 \restriction P' = M_2^{'}$. As σ is enabled in $(N', M_1 \restriction P')$ and $\forall_{t \in \sigma} \bullet t \subseteq P'$, this implies that σ is also enabled in (N, M_1).

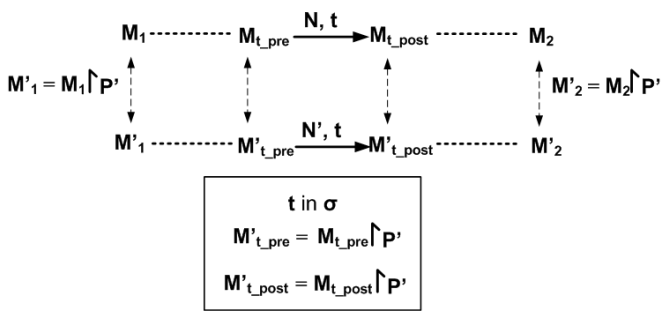

Fig. 24. Transition firings in both the original net and the restricted net

Figure 24 gives the states in both models for transitions that can be fired in both nets (N, N'). Assume that $M_{t-pre} \restriction P' = M_{t-pre}^{'}$ and $t \in T'$. As $\overset{N'}{\bullet} t = \bullet^N t$, $t \overset{N'}{\bullet} = t \overset{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post} \restriction P' = M_{t-post}^{'}$. The effect of firing t is identical on the places in P'. Hence, the marking resulting from σ is the same as M_2 w.r.t. P'. This can be repeated for all $t \in \sigma$, hence: $M_2' = M_2 \restriction P'$.

Corollary 1. *Let $(N, M_1) = ((P, T, F, R), M_1)$ be a marked reset net and $N' = res(N, G) = (P', T', F', R')$ its restriction on G.*

$\exists_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G])$ if and only if $\exists_{M_2' \in \mathbb{M}(N')}(M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \wedge M_1[G] \sqsubset M_2^{'}[G])$

Proof. ($\Rightarrow$) First, we will prove that $\exists_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G])$ implies that $\exists_{M_2' \in \mathbb{M}(N')}(M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \wedge M_1[G] \sqsubset M_2^{'}[G])$. Assume $M_2 \in \mathbb{M}(N)$ such that $M_1 \overset{N,*}{\rightarrow} M_2$ and $M_1[G] \sqsubset M_2[G]$. Using Lemma 2, we can show that there is an $M_2^{'}$ such that $M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \wedge M_2^{'} \geq M_2 \restriction P'$. Restricting $M_2^{'}$ to G gives $M_2^{'}[G] \geq M_2[G]$ as $G \subseteq P'$. We now have $M_1[G] \sqsubset M_2[G]$ and $M_2^{'}[G] \geq M_2[G]$ and therefore, $M_1[G] \sqsubset M_2^{'}[G]$.

($\Leftarrow$) Second, we will prove that $\exists_{M_2' \in \mathbb{M}(N')}(M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \wedge M_1[G] \sqsubset M_2^{'}[G])$ implies that $\exists_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G])$. Assume $M_2^{'} \in \mathbb{M}(N')$ such that $M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'}$ and $M_1[G] \sqsubset M_2^{'}[G]$. Using Lemma 3, we can show that there is an M_2 such that $M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_2^{'} = M_2 \restriction P'$. Restricting M_2 to G shows $M_2^{'} \restriction P'[G] = M_2^{'}[G]$. Hence, $M_2^{'} \restriction G = M_2 \restriction G$. Combined with $M_1 \restriction G \sqsubset M_2^{'} \restriction G$, this yields $M_1[G] \sqsubset M_2[G]$.

An OR-join task *o-j* is enabled at a marking M of an E2WF-net N, if it is not possible to reach a marking M_n such that $M \xrightarrow{*} M_n$ and $M[\bullet$*o-j*$] \sqsubset M_n[\bullet$*o-j*$]$.To determine whether *o-j* should be enabled at M, we proposed to perform the following analysis. Let $N_R = \text{transE2WFOJ}(N,$ *o-j*$)$ be the rest net, $G = \bullet$*o-j* and M_R be the corresponding marking of M in the reset net. Instead of using N_R to perform the analysis, we can reduce the search space by applying the structural restriction so that $N_R^{'} = \text{res}(N, G,)$. Using Corollary 1, we can determine whether there is a marking $M_R^{'} \in \mathbb{M}(N_R)$ such that $M_R \xrightarrow{N_{R,}*}$ and $M_R[G] \sqsubset M_R^{'}[G]$. If it does, this implies that more tokens can be placed into the input places of *o-j* in the reachable markings from M_R. Hence, the OR-join analysis can take place in the restricted net N_R and *o-j* should not be enabled at M.

6.2 Active projection

In addition to applying structural restriction to a YAWL net, it is also possible to further restrict the net using the current marking. As a transition that cannot be enabled in the reachable markings from the current marking cannot be fired and its output places can never be reached, we can safely exclude this transition from the restricted net. Applying active projection involves removing tasks and conditions from a YAWL net that cannot be reached from a given marking. This enables us to only consider the selected paths of a YAWL net that can be reached from the current marking. As a YAWL net is translated into a reset net, the active projection restriction will also be performed on the reset net.

The function $\mathbf{ap}$ describes how a reset net could be constructed so that only the transitions and places that can be reached from a given marking are included in the restricted net. Figure 25 shows the effect of active projection function $\mathbf{ap}$ on a reset net with a marking M where marked$(M) = \{p_a, p_b, p_c\}$. The restricted region contains all the places that could potentially be marked in the reachable markings of M (e.g. $p_1, p_2, p_4, p_5, p_6, p_8$). A transition t is in the restricted net if and only if all its input places are in the restricted region ($\bullet t \subseteq P'$). See t_5 with its only input place p_5 in the restricted net. For transition t_4, not all input places of t_4 are in the restricted region and therefore, $t_4 \notin T'$. Relation R' will keep track of the reset places in P' for any transition $t \in T'$ with reset arcs. For example, both transitions t_9 and t_{10} could reset P_2 but, R' will only contain (t_9, p_2) as t_{10} is not in T'.

Definition 21 (ap(N, M)). *Let $(N, M) = ((P, T, F, R), M)$ be a marked reset net. $N' =$ ap$(N, M) = (P', T', F', R')$ is the active projection of (N, M) where*

$P' = \{p \in P | \exists_{p' \in \textit{marked}(M)} (p', p) \in F^*\}$,
$T' = \{t \in T | \bullet t \subseteq P'\}$,
$F' = F \cap ((P' \times T') \cup (T' \times P'))$, *and*
$R' = \{(t, R(t) \cap P') | t \in \textit{dom}(R) \cap T'\}$.

Let N, N' be two reset nets such that $N' = \text{ap}(N, M_1)$, after applying active projection, for a given marking M_1. Lemma 4 will demonstrate that for any marking $M_2 \in \mathbb{M}(N)$ reachable from M_1, there is a corresponding marking $M_2^{'} \in \mathbb{M}(N')$

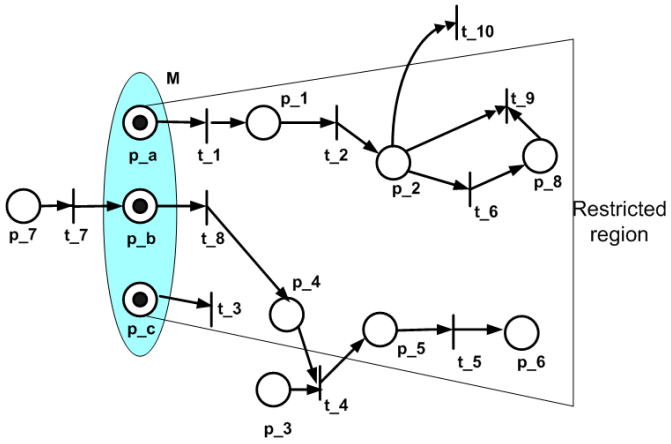

Fig. 25. Active projection diagram

reachable from $M_1 \restriction P'$ such that $M_2^{'} = M_2 \restriction P'$. That is, both markings are the same w.r.t P'.

Lemma 4. *Let* $(N, M_1) = ((P, T, F, R), M_1)$ *be a marked reset net and* $N' = ap(N, M_1) = (P', T', F', R')$ *its active projection.*

$\forall_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \Rightarrow M_1 \restriction P' \overset{N',*}{\rightarrow} M_2 \restriction P')$

Proof. Consider a firing sequence $\sigma : M_1 \overset{N,\sigma}{\rightarrow} M_2$.

First, we will prove that σ is enabled in $(N', M_1 \restriction P')$. From Definition 21, $t \in T \backslash T'$ implies that t cannot be enabled in any reachable marking from M_1 and therefore, $t \notin \sigma$. So σ only contains transitions $t \in T'$. The enabling of $t \in T'$ only depends on places in P' (i.e. $\overset{N}{\bullet} t = \overset{N'}{\bullet} t \subseteq P'$). Figure 26 gives the states in both models. Assume that $M_{t-pre} \restriction P' = M_{t-pre}^{'}$ and $t \in T'$. Firing t only affect the output places and they are all in P' (i.e. $t \overset{N}{\bullet} = t \overset{N'}{\bullet} \subseteq P'$). As $\overset{N'}{\bullet} t = {}^{N}\bullet t$, $t \overset{N'}{\bullet} = t \overset{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post} \restriction P' = M_{t-post}^{'}$. This can be repeated for all $t \in \sigma$, hence: $M_2' = M_2 \restriction P'$.

Lemma 5 will demonstrate that for any marking $M_2^{'} \in \mathbb{M}(N')$ reachable from $M_1 \restriction P'$, there is a corresponding marking $M_2 \in \mathbb{M}(N)$ reachable from M_1 such that $M_2' = M_2 \restriction P'$.

Lemma 5. *Let* $(N, M_1) = ((P, T, F, R), M_1)$ *be a marked reset net and* $N' = ap(N, M_1) = (P', T', F', R')$ *its active projection.*

$\forall_{M_2' \in \mathbb{M}(N')}(M_1 \restriction P' \overset{N',*}{\rightarrow} M_2^{'} \Rightarrow \exists_{M_2 \in \mathbb{M}(N)} M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_2^{'} = M_2 \restriction P')$

Proof. Consider a firing sequence $\sigma : M_1 \restriction P' \overset{N',\sigma}{\rightarrow} M_2^{'}$. We will prove that σ is enabled in (N, M_1). As σ is enabled in $(N', M_1 \restriction P')$ and $\forall_{t \in \sigma} \overset{N}{\bullet} t = \overset{N'}{\bullet} t \subseteq P'$, this implies that σ is also enabled in (N, M_1). From Definition 21, $t \in T \setminus T'$ implies that t cannot be

enabled in any reachable marking from M_1 and therefore, $t \notin \sigma$. So σ only contains transitions $t \in T'$. The enabling of $t \in T'$ only depends on places in P' (i.e. ${}^{N}\bullet t = {}^{N'}\bullet t \subseteq P'$). Figure 26 gives the states in both models. Assume that $M_{t-pre}{\restriction}P' = M'_{t-pre}$ and $t \in T'$. Firing t only affect the output places and they are all in P' (i.e. $t{\bullet}^{N} = t{\bullet}^{N'} \subseteq P'$). As ${}^{N'}\bullet t = {}^{N}\bullet t$, $t{\bullet}^{N'} = t{\bullet}^{N} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post}{\restriction}P' = M'_{t-post}$. This can be repeated for all $t \in \sigma$, hence: $M'_2 = M_2{\restriction}P'$.

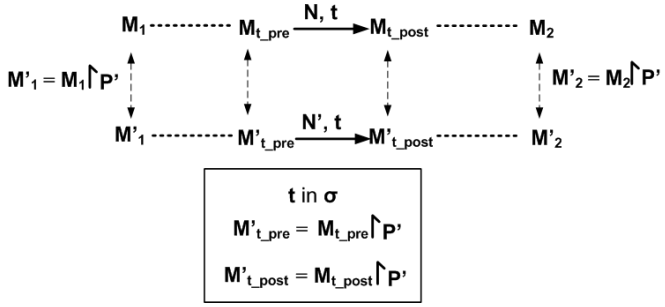

Fig. 26. Transition firing in a restricted net (active projection)

Corollary 2. *Let* $(N, M_1) = ((P, T, F, R), M_1)$ *be a marked reset net,* $G \subseteq P$*, and* $N' = res(ap(N, M_1), G) = (P', T', F', R')$.

$\exists_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G])$ *if and only if* $\exists_{M'_2 \in \mathbb{M}(N')}(M_1{\restriction}P' \overset{N',*}{\rightarrow} M'_2 \wedge M_1[G] \sqsubset M'_2[G])$

Proof. ($\Rightarrow$) First, we will prove that $\exists_{M_2 \in \mathbb{M}(N)}(M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G])$ implies that $\exists_{M'_2 \in \mathbb{M}(N')}(M_1{\restriction}P' \overset{N',*}{\rightarrow} M'_2 \wedge M_1[G] \sqsubset M'_2[G])$. Using Lemma 4, we can show that $M_1 \overset{N,*}{\rightarrow} M_2$ implies $M_1{\restriction}P' \overset{N',*}{\rightarrow} M_2{\restriction}P'$. Hence, there exists an $M'_2 = M_2{\restriction}P'$ such that $M_1[G] \sqsubset M'_2[G]$.

($\Leftarrow$) Second, we will prove that $\exists_{M'_2 \in \mathbb{M}(N')} M_1 \restriction P' \overset{N',*}{\rightarrow} M'_2 \wedge M_1[G] \sqsubset M'_2[G]$ implies that $\exists_{M_2 \in \mathbb{M}(N)} M_1 \overset{N,*}{\rightarrow} M_2 \wedge M_1[G] \sqsubset M_2[G]$. Using Lemma 5 and assuming $M_1 \restriction P' \overset{N',*}{\rightarrow} M'_2$, there exists a M_2 such that $M_1 \overset{N,*}{\rightarrow} M_2$ and $M_2 \restriction P' = M'_2$. Since $G \subseteq P$, $M_1[G] \sqsubset M'_2[G]$ implies $M_1[G] \sqsubset M_2[G]$.

An OR-join task *o-j* is enabled at a marking M of an E2WF-net N, if it is not possible to reach a marking M_n such that $M \overset{*}{\rightarrow} M_n$ and $M[\bullet o\text{-}j] \sqsubset M_n[\bullet o\text{-}j]$.To determine whether *o-j* should be enabled at M, we proposed to perform the following analysis. Let $N_R = \text{transE2WFOJ}(N, o\text{-}j)$ be the rest net, $G = \bullet o\text{-}j$ and M_R be the corresponding marking of M in the reset net. Instead of using N_R to perform the analysis, we can reduce the search space by first applying the structural restriction and active projection techniques so that $N'_R = \text{res}(\text{ap}(N_R, M_R), G) = (P', T', F', R')$. Using Corollary 2,

we can determine whether there is a marking $M_R^{'} \in \mathbb{M}(N_R)$ such that $M_R \overset{N_{R,*}}{\rightarrow}$ and $M_R[G] \sqsubset M_R^{'}[G]$. If it does, this implies that more tokens can be placed into the input places of *o-j* in the reachable markings from M_R. Hence, the OR-join analysis can take place in the restricted net N_R and *o-j* should not be enabled at M.

7 Implementation

The OR-join analysis algorithm as described in Section 5 together with the structural restriction and active-projection techniques from Section 6 have been implemented in the YAWL engine[6]. The algorithm uses the OR-join semantics described in Section 4 to detect when an OR-join should be enabled. The algorithm still uses an enumerative approach for storage of markings but we have implemented a number of intuitive optimisation techniques. A number of YAWL models have been tested and OR-join enabling results are as expected. The observations also indicate that restriction techniques significantly reduce the execution times for OR-join analysis.

We present here execution times of OR-join enabling algorithm for a number of YAWL models. Five different execution times for each OR-join evaluation call will be presented for comparison. **SRestrict+AProject** indicates that structural restriction is applied first and then, active projection is applied before OR-Join call. **AProject+SRestrict** indicates that active projection is applied first and then, structural restriction is applied before OR-Join call. **SRestrict** indicates that only structural restriction has been applied. **AProject** indicates that only active projection has been applied. **NoRestrict** indicates that no restriction technique has been applied. To minimise the effects of variations, each method is called 100 consecutively. Furthermore, this process has been repeated ten times for sampling. We will provide average execution times with confidence intervals (95%). All the figures are in milliseconds and are rounded to one decimal point.

7.1 A structured YAWL net with an OR-split and an OR-join

The YAWL net in Figure 8 represents a small structured net with an OR-split task A and an OR-join task E. At a marking $M = c1 + c2 + c6$, OR-join evaluation for E returns FALSE. A new marking $M_1 = c1 + c5 + c6$ is reached after executing task C at M. The execution times for the analysis are shown in Table 1. Even from a small example, we can see that the combined restriction techniques can reduce the time it takes to perform the OR-join evaluation.

7.2 A YAWL net with loop and cancellation

Figure 7 represents a YAWL net with a loop and cancellation on the path to OR-join task E. At a marking $M = c2$, OR-join evaluation for task E returns TRUE as it is not possible to reach a bigger marking from M. The execution times are shown in Table 2. Again, we can see that the combined restriction techniques can reduce the

[6] (http://sourceforge.net/projects/yawl/)

Table 1. Execution times for the YAWL model in Figure 8

OR-join: E	Marking: $c1 + c2 + c6$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	335.9	10.0
AProject+SRestrict	328.0	11.3
AProject	306.4	22.9
SRestrict	790.4	21.1
NoRestrict	790.5	24.8
OR-join: E	Marking: $c1 + c5 + c6$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	130.2	2.3
AProject+SRestrict	126.6	3.1
AProject	107.6	4.6
SRestrict	3126.6	114.8
NoRestrict	3172.0	84.8

evaluation time. The difference between the execution times for structural restriction and no restriction calls is minimal as most tasks and conditions in this YAWL net will be in the restricted net as well.

Table 2. Execution times for the YAWL model in Figure 7

OR-join: E	Marking: $c2$ returns TRUE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	685.9	16.9
AProject+SRestrict	676.8	6.9
AProject	654.7	16.9
SRestrict	2365.5	81.9
NoRestrict	2348.4	17.5

7.3 A larger YAWL net with loop and cancellation

We have presented in Table 3 the execution times for an OR-join evaluation call for OR-join task G with two markings $c1 + c7$ and $c_{BB} + c3 + c7$ in Figure 18. OR-join evaluation for both markings returns FALSE. This YAWL net also contains a loop and cancellation on the path to G. In this case, the restriction techniques reduce the execution time by a significant amount. The difference between structural restriction and no restriction calls is minimal in this example as most tasks and conditions in the YAWL net are also in the structurally restricted net.

7.4 A large YAWL net with an OR-join task

To demonstrate the impact on structural restriction on OR-join analysis, we present a YAWL net in Figure 27 that contains a number of tasks which have no impact on the

Table 3. Execution times for the YAWL model in Figure 18

OR-join: G	Marking: $c1 + c7$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	1032.8	12.4
AProject+SRestrict	1032.8	10.5
AProject	1003.1	5.8
SRestrict	11664.0	16.5
NoRestrict	11654.9	33.9

OR-join: G	Marking: $c_{BB} + c3 + c7$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	587.4	7.2
AProject+SRestrict	585.9	9.9
AProject	568.7	9.8
SRestrict	11195.3	12.1
NoRestrict	11198	21.9

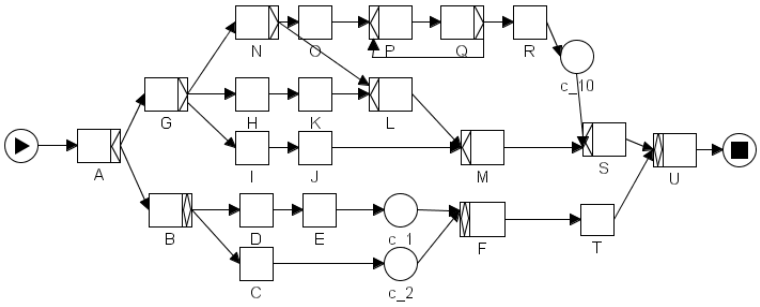

Fig. 27. A YAWL net with an OR-join tasks F and U

OR-join task F. For instance, all tasks and conditions on the path between tasks G to S could not influence the OR-join analysis for task F. Average execution times for a marking $M = c_{AG} + c_{BD} + c_2$ are given in Table 4. Average execution times for OR-join analysis of F with a marking $M = c_{BD} + c_2 + c_{10}$ are also given. The figures show a huge difference in execution times between different restriction techniques. Average execution times for OR-join analysis of U with a marking $M = c_{AG} + c_{TU}$ are given in Table 5. In this case, structural restriction alone does not reduce the execution time as most tasks in the YAWL net are also part of the structurally restricted net. However, the combination of structural restriction and active projection reduces the execution time significantly (2148.5 milliseconds cf. 84863.9 milliseconds). From these tests, it is evident that performing structural restriction and active projection on a YAWL net before an OR-join analysis could reduce execution time of an OR-join evaluation significantly.

Table 4. Execution times for Task F from the YAWL model in Figure 27

OR-join: F	Marking: $c_{AG} + c_{BD} + c_2$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	465.8	24.5
AProject+SRestrict	529.7	10.3
AProject	796.7	6.7
SRestrict	1479.8	14.7
NoRestrict	3681.3	9.5

OR-join: F	Marking: $c_{BD} + c_2 + c_{10}$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
AProject+SRestrict	275.0	7.1
SRestrict+AProject	304.7	86.5
AProject	276.5	9.3
SRestrict	1198.4	9.4
NoRestrict	3492.2	23.8

Table 5. Execution times for Task U from the YAWL model in Figure 27

OR-join: U	Marking: $c_{AG} + c_{TU}$ returns FALSE	
Duration (100 calls)	OJ	Confidence Interval
SRestrict+AProject	2148.5	60.6
AProject+SRestrict	2123.5	20.4
AProject	2014.0	20.3
SRestrict	85404.8	208.6
NoRestrict	84863.9	144.6

8 Epilogue

This paper focuses on the OR-join construct in YAWL and proposes a new semantics. The decision to enable an OR-join task cannot be made locally: an OR-join task should only be enabled when there is at least one token in one of the input conditions and there is no possibility of a token arriving at one of the yet unmarked input conditions of the OR-join. Otherwise, the OR-join task should wait for synchronisation. Instead of ignoring other OR-joins on the path, we propose two alternative approaches (optimistic or pessimistic) for OR-joins which are on the path of other OR-joins. Reset nets are used as formal basis for OR-join analysis to support cancellation feature. This is made possible by the fact that we can abstract from the concepts of YAWL such as multiple instances, composite tasks and internal state transitions of a task. We present transformation rules from a YAWL model with OR-joins to a reset net for a specific OR-join analysis. We then propose an OR-join evaluation algorithm which is based on the backward search techniques for Well-Structured Transition Systems. We also present structural restriction and active projection techniques for optimisation together with the findings from the implementation in the YAWL engine.

Other optimisation techniques can also be applied to improve the performance of OR-join analysis. Our algorithm does not yet exploit these potential optimisation tech-

niques. For instance, well-known Petri net reduction techniques as described in [21] can be applied to the reset net. The use of more efficient data structures for storage of markings could also be explored [7, 8, 12]. In [7, 8], the authors propose a new symbolic representation for upward-closed sets based on the sharing trees, called Covering Sharing Trees (CSTs) to compactly represent upward-closed sets of markings. In [12], the authors present the symbolic algorithms for forward and backward search techniques. Incremental methods can also be used so that OR-join analysis does not need to be performed for every marking change. By keeping track of the mapping used for the reductions and a record of the relationship between the original reset net and the reduced reset net, together with a set of markings that do not enable the OR-join task, incremental techniques can be used to perform a more efficient OR-join evaluation.

To conclude the paper, we would like to emphasise that the results reported in this paper are not limited to YAWL. As is indicated in the introduction, many workflow management systems, but also other process-aware information systems (e.g., ERP, CRM, and PDM systems), have problems dealing with the OR-join. In fact, the problem surfaces in many other domains [24].

Acknowledgements. We would like to especially thank Philippe Schnoebelen and Jerome Leroux for their valuable input on the issue of decidability of OR-join algorithm and for many useful references provided in the area of reset nets.

References

1. 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.
2. W.M.P van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Rump and F.J. Nüttgens, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, 2002. Gesellschaft für Informatik, Bonn.
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. P.A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (27 - 30 July)*, pages 313–321, New Brunswick, NJ, July 1996. IEEE Computer Society.
6. P. Darondeau. Unbounded Petri net Synthesis. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–428, Eichstätt,Germany, 2003. Springer-Verlag.
7. G. Delzanno and J.-F. Raskin. Symbolic Representation of Upward-Closed Sets. In S.Graf and M.Schwartzbach, editors, *TACAS/ETAPS*, volume 1785 of *Lecture Notes in Computer Science*, pages 426–441. Springer-Verlag, 2000.
8. G. Delzanno, J.-F. Raskin, and L. van Begin. Attacking Symbolic State Explosion. In H. Common J. Berry and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 298–310. Springer-Verlag, 2001.
9. 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 Inter-*

national Colloquium on Automata, Languages and Programming, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.
10. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T Nets. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Lectures on Concurrency and Petri Nets*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer-Verlag.
11. Eastman Software. *RouteBuilder Tool User's Guide*. Eastman Software, Inc, Billerica, MA, USA, 1998.
12. A. Finkel, J.-F. Raskin, M. Samuelides, and L. van Begin. Monotonic Extensions of Petri Nets: Forward and Backward Search Revisited. *Electronic Notes in Theoretical Computer Science*, 68(6):1–22, 2002.
13. A. Finkel and Ph. Schnoebelen. Fundamental Structures in Well-Structured Infinite Transition Systems. In C.L. Lucchesi and A.V. Moura, editors, *Theoretical Informatics: Third Latin American Symposium, Campinas, LATIN'98 (20 - 24 April)*, volume 1380 of *Lecture Notes in Computer Science*, pages 102–118, Campinas, Brazil, 1998. Springer-Verlag.
14. A. Finkel and Ph. Schnoebelen. Well-structured Transition Systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, April 2001.
15. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
16. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. Phd thesis, Queensland University of Technology, Brisbane, Australia, 2003.
17. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In J. Desel, B. Pernici, and M. Weske, editors, *Proceedings of 2nd International Conference on Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97, Potsdam, Germany, 2004. Springer–Verlag.
18. E. Kindler. On the Semantics of EPCs: Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2005.
19. M. Leuschel and H. Lehmann. Coverability of Reset Petri Nets and other Well-Structured Transition Systems by Partial Deduction. In J. Lloyd et al., editors, *Proceedings of Computational Logic 2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 101–115, London, UK, 2000. Springer-Verlag.
20. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
21. T. Murata. Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.
22. P. Rittgen. Modified EPCs and their Formal Semantics. Technical Report 99/19, Institute of Information Systems, University Koblenz-Landau, Koblenz, Germany, 1999.
23. 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 (20 - 25 June)*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443, Miami, USA, June 2005. Springer-Verlag.
24. A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the Models for Asynchronous Circuit Behaviour with OR Causality. *Formal Methods in System Design*, 9(3):189–233, 1996.