International Journal of Cooperative Information Systems
© World Scientific Publishing Company

SYNCHRONISATION AND CANCELLATION IN WORKFLOWS BASED ON RESET NETS

M.T. WYNN, W.M.P. VAN DER AALST, A.H.M. TER HOFSTEDE AND D. EDMOND

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

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 branches that cannot complete will not be waited for. Many systems and languages struggle with the semantics and implementation of the OR-join because its non-local semantics require a synchronisation depending on an analysis of future execution paths. The presence of cancellation features and other OR-joins in a workflow further complicates the formal semantics of the OR-join.

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 paper provides a suitable (non-local) semantics for an OR-join and gives a concrete algorithm with two optimisation techniques to support the implementation. 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 when its transition fires. Through the behaviour of reset arcs, the behaviour of cancellation regions can be captured in a natural manner.

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

1. Introduction

Workflow systems aim to provide automated support for the conduct of certain business processes. Workflow systems are driven by specifications which among others, capture the execution interdependencies between various activities. These interdependencies are modelled by means of different control flow constructors, e.g., sequence, choice, parallelism and synchronisation. It is shown in the workflow patterns research that the support for and the interpretation of various control flow constructs varies substantially across workflow systems [14]. Two of the most problematic patterns relate to the *OR-join* and to *cancellation*.

Typically, synchronisation of parallel activities in workflows can be achieved using one of three join constructs: AND-join, XOR-join and OR-join. The AND-join construct requires strict synchronisation. All paths must be completed before the task following the AND-join can be started. As a result, the entire workflow can deadlock if not every path can be completed. On the other hand, the XOR-join can be started when one path is completed and hence, the task following the XOR-join may be completed multiple times. In some cases, this may be undesirable or expensive. An OR-join provides a middle ground between these two join structures by allowing a workflow to continue when only certain paths are completed and it is certain that other paths can never complete.

The presence of cancellation feature further complicates the formal semantics of the OR-join. The cancellation feature is commonly used to model external events that can change the behaviour of a running workflow. It can be used to either disable activities in certain parts of a workflow or to stop currently running activities. Even though it is possible to cancel activities in workflow systems using some sort of abort function, many workflow systems do not provide direct support for this feature in the workflow language. Sometimes, cancellation affects only a selected part of a workflow and other activities can continue after performing a cancellation action. In those cases, an OR-join is the only synchronisation construct flexible enough to ensure that the process is completed correctly. As cancellation occurs naturally in business scenarios, comprehensive support and a corresponding implementation in workflow systems is required.

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 (e.g., IBM MQSeries, InConcert, eProcess, WebSphere MQ Workflow, Eastman, Domino) and business process modelling tools support OR-join-like constructs (e.g., Event Process Chains (EPCs) and Business Process Modelling Notation (BPMN)). Support for cancellation features are also provided in the Business Process Modelling Notation (BPMN), the Business Process Execution Langauge (BPEL) and the Unified Modelling Language (UML).

Even though the OR-join construct is useful in process modelling, its formal semantics is difficult to capture and to implement. 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 the active paths that are being executed in a given workflow instance. The difficult question arises as to when an OR-join should wait and when it should go ahead. This decision *cannot be made locally*, that is, just by evaluating the current state of the workflow. The decision requires the awareness of the current state as well as possible future states of the workflow. State space analysis could be used to determine all future states of the workflow. However, this analysis becomes much more complicated when there are multiple OR-joins in the workflow or when other complex constructs such as cancellation and loops are present in the workflow. Defining the non-local semantics of an OR-join is not trivial even when a workflow language does not support complex constructs (e.g., cancellation) and/or puts certain restrictions on the models (e.g., no loops or only allow structures where an OR-join is preceded by an OR-split).

In a workflow language that supports all these constructs without restrictions, there are a number of complicating factors when it comes to defining a *general approach* to OR-join semantics. Firstly, for workflows with multiple OR-joins, it is an open issue how a state space analysis for a certain OR-join should treat other OR-joins. Secondly, for workflows with infinite loops (e.g., a continuous monitoring activity), the state space could be infinite. Thirdly, cancellation regions complicate the computation of future states. A task that an OR-join is waiting for that is in the cancellation region of some other task may or may not be disabled. Such considerations make state space analysis computationally expensive. In this paper, we take on this challenge and propose a general approach to OR-join semantics in the presence of cancellation features and without imposing extra structural restrictions for OR-joins. The OR-join semantics is defined and presented in terms of the workflow language YAWL that provides support for cancellation regions [2].

The **contributions** of this paper are threefold. Firstly, the OR-join semantics as proposed by van der Aalst and ter Hofstede [2] is re-examined. We will argue that its behaviour does not match the informal semantics in the context of other OR-joins. Secondly, the mapping of YAWL nets to reset nets is exploited to find an algorithmic solution to the non-trivial problem of OR-join enablement. Thirdly, two restriction techniques are proposed to make the OR-join algorithm more efficient.

The rest of the paper is organised as follows. In Section 2, various attempts at defining and supporting OR-join semantics from the literature are presented, including the problems associated with the original OR-join semantics in YAWL [2] as well as possible improvements. Section 3 formally defines a new semantics for the OR-join in YAWL. Section 4 demonstrates an algorithm to determine when an OR-join is enabled. This algorithm is based on backwards search techniques drawn from the area of Well-Structured Transition systems [8,13]. Section 5 presents two restriction techniques to improve the efficiency of the analysis. Section 6 describes the implementation in the context of the open source system YAWL and provides a detailed analysis of its performance. Section 7 describes a realistic YAWL model for the general skill migration visa application to Australia, which contains multiple cancellation regions and multiple OR-joins. Execution times for analysing OR-joins in this visa application model are also presented. Section 8 discusses related work and Section 9 concludes the paper. Appendix A contains background definitions for Well-Structured Transition Systems and Appendix B contains proofs for restriction techniques.

This paper extends work by the authors previously reported in [23] in the following ways. More examples have been added to illustrate the OR-join semantics in Section 2. Structural restriction and active projection techniques have been proposed to improve the performance in Section 5. A number of experiments have been carried out to determine the correctness of the OR-join enablement algorithm in various settings. The results from these experiments have been reported in Section 6. Section 7 shows how the OR-join analysis is carried out for the visa application example.

2. OR-join semantics

In this section, we first present the different variants of an OR-join semantics from the literature. The informal semantics of an OR-join in the YAWL language is then explained using a number of examples. Problems associated with the original OR-join semantics in YAWL for multiple OR-joins are then discussed. Two alternative treatments for dealing with multiple OR-joins are then proposed.

2.1. *Different interpretations and implementations of an OR-join*

Several variants and interpretations of the OR-join have been proposed in the literature. In Rittgen's report [19], 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". The presence of a matching split makes the process "structured". 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" [19]. The Business Process Modelling Notation (BPMN) also contains an OR-join like construct called inclusive OR-join gateway [20]. The semantics of an OR-gateway in BPMN 1.0 is given Sepcification as "it will wait for (synchronize) all Tokens that have been produced upstream. If an upstream Inclusive OR produces two out of a possible three Tokens, then a downstream Inclusive OR will synchronize those two Tokens and not wait for another Token, even though there are three incoming Sequence Flow". However, the OR-join gateway does not capture the correct behaviour for unstructured BPMN models [21]. It seems to be challenging to select a suitable OR-join semantics and to implement it efficiently. Van der Aalst et al [1] highlight the technical, conceptual and practical problems with the formal semantics of the OR-join in EPCs. 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. It was suggested 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.

Many workflow systems and languages also struggle with the semantics and implementation of the OR-join. This is because its non-local semantics requires a synchronisation depending on an analysis of future execution paths, which 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 [18] (used as a basis for the BPEL standard) appears to offer full support for the OR-join for acyclic workflows [1]. As a consequence of the requirement for workflows to be acyclic, loops are disallowed and the only way to introduce loops is by specifying a postcondition for a subprocess; the subprocess is then repeated until the postcondition evaluates to true. Other systems like Eastman and Domino Workflow also support an OR-join concept with non-local semantics. The use of the non-local semantics may result in poor performance as is stated in the manual of Eastman and the recommendation to avoid this type of routing [10]. Even the OR-join definition from the Workflow Management Coalition does not support non-local semantics. An OR-join is defined as "a point within the workflow where two or more alternative activity(s) workflow branches re-converge to a single common activity as the next step within the workflow. (As no parallel activity execution has occurred at the join point, no synchronisation is required.)". For a more complete discussion on OR-join semantics, we refer the reader elsewhere [1,3,14,15,16].

In the collection of workflow patterns, the synchronising merge pattern captures the essence of an OR-join [14]. The OR-join construct in YAWL, is intended to provide direct support for OR-join semantics while imposing *no syntactical restrictions*. However, the original OR-join semantics as defined in [2] could yield counter-intuitive results such as e.g., an OR-join firing prematurely for workflows with multiple OR-joins and OR-joins in sub-processes. Hence, we believe that there is scope for further improving the semantics of an OR-join concept in YAWL.

2.2. *OR-join semantics 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. Tasks are active components in a YAWL model and when a task fires, tokens are consumed from its input conditions and tokens are generated for its output conditions depending on its split and join behaviours. There are three kinds of split and three corresponding kinds of join; they are AND, XOR and OR. The splits, joins, conditions and cancellation symbols for YAWL are shown in Figure 1. Each YAWL model has one start condition and one end condition. 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 its input conditions. 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. In YAWL, the semantics of an XOR-join is considered to be local. In [15,5], the XOR-join is also assumed to have non-local semantics. When a task is executed, it takes tokens out of its 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.

In general, an OR-join task is enabled if there is at least one token in one of its input conditions and it is not possible for more tokens to arrive in other (currently empty) input conditions in the future states (i.e, there is no need to wait for synchronisation). If it is possible for tokens to arrive in currently empty input conditions in the future states, then the OR-join task should wait before proceeding. This is the desired behaviour of an OR-join and we will refer to this as the *informal semantics* of an OR-join.

A more technical explanation of the OR-join semantics is as follows: an OR-join is

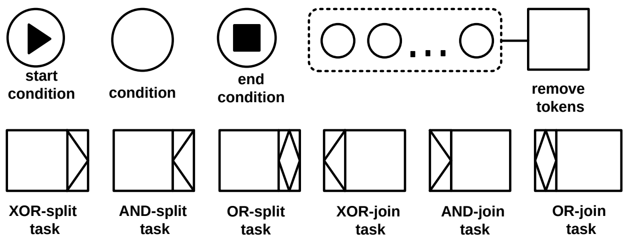

Fig. 1. Splits, joins, conditions and cancellation in YAWL

enabled at a marking if and only if 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.

Next, a series of YAWL nets are presented to establish the need for complex analysis to determine whether an OR-join is enabled. For simplicity, the YAWL nets use short labels (e.g., A, c1, etc.) to identify tasks and conditions. This allows us to focus on the control flow requirements for a particular net. We also give here an informal explanation of some Petri net terminology, such as marking and reachability, which is also used in YAWL. As with Petri nets, the term *marking*, is used to describe the state of a YAWL model and is represented as the number of tokens in certain conditions of a net (e.g., $M = c1 + c5$ represents a state of the workflow where there is one token each in conditions $c1$ and $c5$). A marking is *reachable* from another marking, if there is a sequence of tasks that can fired from the first marking to arrive at the second marking.

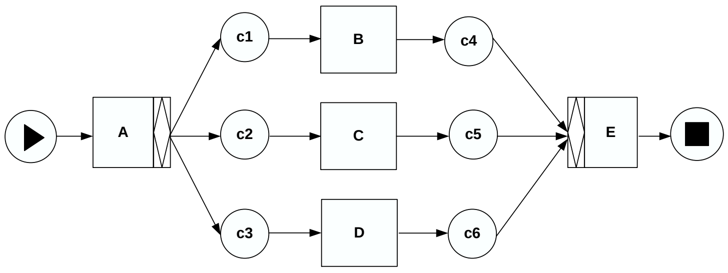

Fig. 2. A structured YAWL net with an OR-split task A and an OR-join task E

Figure 2 is a net where A is an OR-split task and E is an OR-join task. The initial marking for the net has exactly one token in the start condition. At the initial marking, task A is enabled and can be fired. After A is executed, tokens are put into conditions $c1$, $c2$, and $c3$ according to OR-split behaviour. Note that the OR-split allows one or more paths to be selected after executing the task. Consider a marking $M = c1 + c5$, which 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 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, it is not possible for $c6$ to be marked in the reachable markings of M'. Task E is enabled at M'. As this is a "structured" net, task E is not enabled until the tokens from all the active threads from task A reach the input conditions of E.

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. Figure 3 represents a slight modification to the YAWL net of Figure 2 and it shows that this notion is false. In Figure 3,

$c4$ 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 has any tokens in $c6$ and therefore, E is 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 are on the path to an OR-join task.

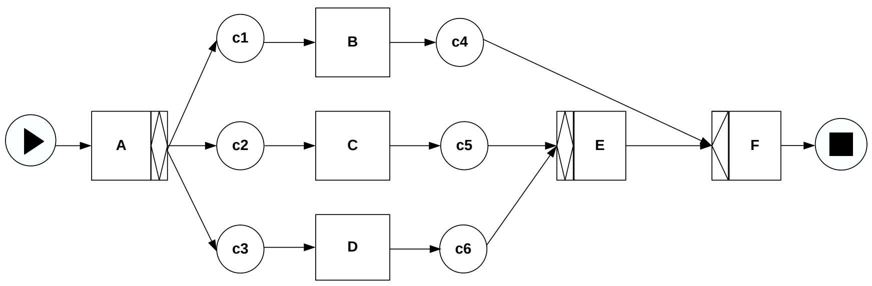

Fig. 3. A YAWL net modified from Figure 2

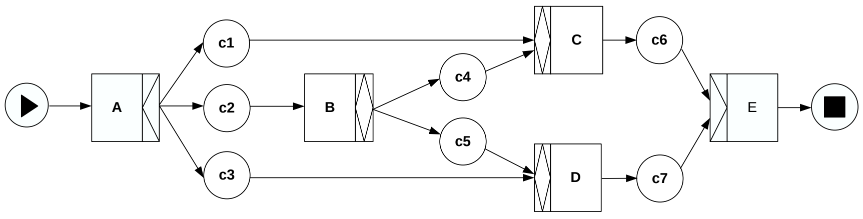

Fig. 4. A YAWL net with two OR-join tasks C and D

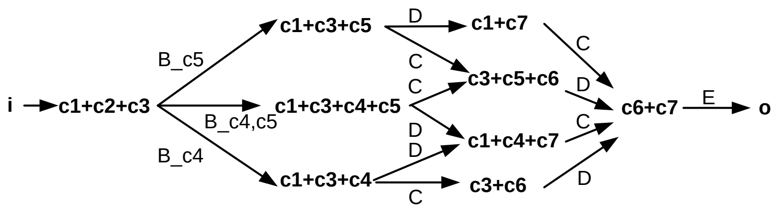

Fig. 5. The reachability graph of the YAWL net in Figure 4

Next, the behaviour of OR-join is described using an example with one OR-split and two OR-joins. The example in Figure 4 demonstrates a net with AND-split task A, AND-join task E, OR-split task B and OR-join tasks C and D. The graph of Figure 5 shows the reachable markings from the initial marking i to the end marking 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. 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 and D should be enabled at M, we need to find out whether either condition $c4$ or $c5$ is marked 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 nor D is 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 is 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.

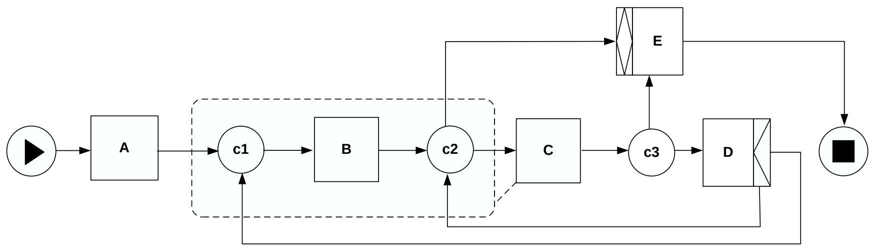

Fig. 6. A YAWL net with a cancellation task C and an infinite loop

Now, let us consider OR-joins in the context of cancellation. Figure 6 describes a 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 $c3$ are marked in reachable markings of M. The following sequence of reachable markings from M can be observed: $c2 \xrightarrow{C} c3 \xrightarrow{D} c1 + c2 \xrightarrow{B} 2c2 \xrightarrow{C} c3$. Similarly, there is another sequence: $c2 \xrightarrow{C} c3 \xrightarrow{D} c1 + c2 \xrightarrow{C} c3$, 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 $c2$, a marking $c3$ is reached and not a marking $c2 + c3$ (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 is enabled at M. Suppose now that task C no longer has a cancellation set associated with it in Figure 6. From the marking $M = c2$, the following sequence of reachable markings can be observed: $c2 \xrightarrow{C} c3 \xrightarrow{D} c1 + c2 \xrightarrow{B} 2c2 \xrightarrow{C} c2 + c3$. 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. One possible technique is to perform the state space analysis of the entire workflow model. An OR-join algorithm can evaluate possible reachable markings from a given 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.

2.3. *Problems with the original semantics*

Two problems may be identified with the original OR-join semantics of YAWL [2]. The first problem is related to the treatment of other OR-joins preceding an OR-join under consideration. The OR-join semantics ignores other OR-joins when analysing whether a particular OR-join should be enabled at a given marking. In Figure 7, there are two OR-join tasks, E and F in the 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 $c1+c3 \xrightarrow{C} c3+c4+c5 \xrightarrow{D} c3+c4+c6 \xrightarrow{E} c3 + c7$. Hence, $M' = c3 + c7$ is a reachable marking from M. However, the original OR-join semantics 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 o. A net which can produce a token for the output condition o while still having tokens in the other conditions is considered as not having proper completion and is therefore not sound. 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.

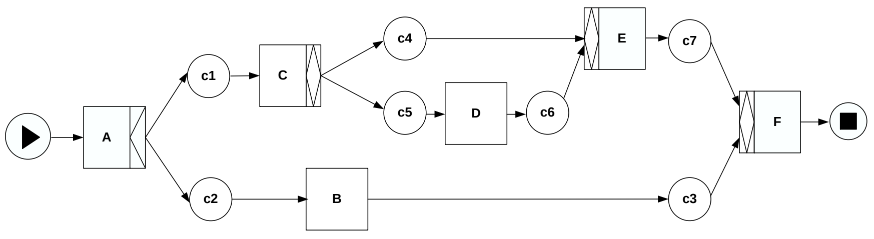

Fig. 7. A YAWL net with an OR-join task E preceding another OR-join task F

The second (related) problem is due to unfolding of composite tasks during an OR-join analysis. This implies that a net at a lower level cannot be considered as a black box. If the lower level net contains OR-joins, it will impact on the OR-join analysis at a higher level net. Consider a specification where task B in Figure 7 is a composite task that is unfolded into a net with an OR-split and an OR-join task as shown in Figure 2. The composite task B will be unfolded to the net in Figure 2 (including the OR-join task E at lower level). The composite task 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 will then conclude incorrectly that F should be enabled at M because condition $c3$. The analysis in the original semantics takes into account the net at the lower level and composite tasks have not been treated as black boxes. We propose that it is possible to abstract from constructs that exist in a lower level net (including OR-joins).

2.4. *Optimistic and pessimistic approaches*

The informal semantics of an OR-join can be supported quite well when there is only one OR-join in a given net. However, when dealing with multiple OR-joins where one precedes the other, the semantics is not well-defined. The question arises as to “how to treat other OR-joins in the workflow while we try to decide whether one OR-join should be enabled?”. Next, we propose to solve this issue by considering one OR-join at a time during the analysis. Instead of ignoring other OR-join tasks during the analysis, two alternative treatments have been proposed for those OR-joins: either as XOR-joins (*optimistic*) or as AND-joins (*pessimistic*). We believe this strategy to be better than ignoring these OR-joins completely during the analysis (as used in the original semantics proposed for YAWL). 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. 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 van der Aalst et al. [1]).

The treatment of an OR-join as an XOR-join is an *optimistic* approach. It is considered *optimistic* as the analysis waits for synchronisation if the resulting XOR-join can be enabled. The term “optimistic” refers to the expectation that the preceding OR-join can be enabled when treated as an XOR-join. Consider a marking $M = c1 + c3$ in Figure 7 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 closely with the informal semantics of an OR-join.

The treatment of an 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 7 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.

In Figure 8, we have an unusual situation described as a vicious circle by Kindler [15] where the two OR-join tasks B and C could be waiting for each other to be fired first and it is unclear what the informal semantics of the model should be. 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, task C is treated as an XOR-join task during the analysis for B. As a result, a reachable marking $c1 + c3 + c6$, which marks both input conditions of B can be found. Therefore, B should not be enabled at $c1 + c2$. Similarly, we 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 is not enabled at $c1 + c2$. As a result of this *optimistic* approach, the net is in *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 $c2 + c4 + c5$. This will enable task C and after firing C, the marking $c3 + c5 + c6$ 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. It can be seen 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. It is clear that in this particular case, there is no formal semantics that can exactly match the informal semantics.

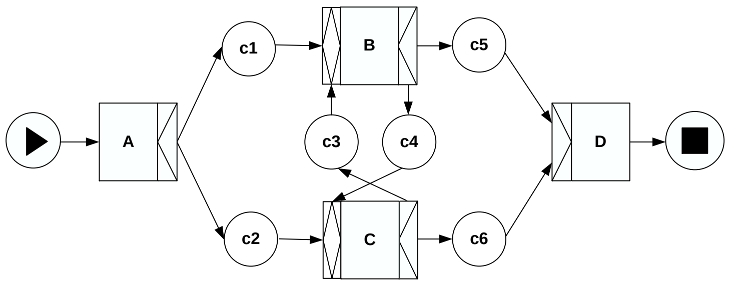

Fig. 8. OR-join tasks B and C in a vicious circle

From the above discussions, it is evident that there is no ideal treatment for multiple OR-joins to support the non-local semantics. Any formal semantics imposes some restrictions or it deviates from the informal semantics to some extent. We have shown that both approaches are valid and can be used during the analysis. The motivation behind our semantics and the use of the optimistic approach is to postpone execution of an OR-join for as long as possible. For this reason, the optimistic approach (XOR-join treatment) is chosen during the analysis. The resulting OR-join semantics is well-defined in every circumstance. However, the interpretation of this semantics can sometime lead to a deadlock in the presence of vicious circles as both OR-joins will wait for each other to fire first.

3. Proposed OR-join semantics

This section proposes new OR-join semantics for YAWL that exploits mappings from YAWL to reset nets. First, for convenience, some background definitions for reset nets and YAWL are presented. Next, a function to transform a YAWL net with OR-joins into a reset net is given. Finally, a new OR-join semantics for YAWL is defined formally.

3.1. *Background definitions*

3.1.1. *Reset nets*

This subsection presents a number of definitions and notations for reset nets, Petri nets with reset arcs([6,8,9,11,13]). The concepts are similar to those defined for Petri nets except that they have been defined for reset nets. A reset net is a Petri net with special *reset arcs*, that can clear the tokens in selected places when its transition fires. A *reset arc* connects a place and a transition and graphically, a reset arc is modelled as a doubled-headed arrow (see Figure 9). The nature of reset arcs matches closely with the concept of cancellation in workflow modelling and reset nets are used as a formalism for modelling workflows with cancellation. This approach allows us to leverage existing literature and techniques in the area of Petri nets and reset nets in particular [4,6,8,9,11,12,13].

Definition 3.1. (Reset net [8]) A reset net is a tuple (P, T, F, R) where (P, T, F) is a Petri net and $R : T \to \mathbb{P}(P)$ assigning a set of (possibly empty) places to every transition.

The complexity introduced by a reset arc (when compared with Petri nets in general) is threefold: 1) as the transition removes *all* tokens and not just one when it fires, place invariants do not hold for such nets, 2) the reset action can be *ineffective* if a place does not contain any tokens at the exact time when the transition fires and the reset action is carried out, and 3) a reset arc can affect any place in the entire net (i.e., its effect is global), unlike normal arcs of a transition which can only influence their input and output places (i.e., their effect is local). As a result, the notion of reachability is undecidable for reset nets with more than two reset arcs [9].

We now fix some additional notations that will be used throughout the paper. Let N be a reset net and $x \in P \cup T$, $\bullet x$ and $x\bullet$ denote the set of inputs to x (preset) and the set of outputs of x (postset). If the net N involved cannot be understood from the context, it is explicitly included, and written as $\overset{N}{\bullet} x$ and $x \overset{N}{\bullet}$. We write F^+ for the transitive closure of the flow relation F and F^* for the reflexive transitive closure of F. When we write $F(x, y)$, this evaluates to 1 if $(x, y) \in F$ and to 0 if $(x, y) \notin F$.

We will use Figure 9 to explain the concepts and notations for reset nets. In Figure 9, we have $\bullet t = \{p1, p2\}$, $t\bullet = \{p3, p4, p5, p6\}$, and $R(t) = \{p3\}$.

Places can contain one or more *tokens* represented by black dots. The *state* of a reset net is represented by a *marking*, that describes the number of tokens in each place of a net.

Definition 3.2. (Marking) Let P be a set of places. A marking M is defined as $M : P \to \mathbb{N}$.

A marking can be interpreted as a vector, a function, and a multiset just as with ordinary

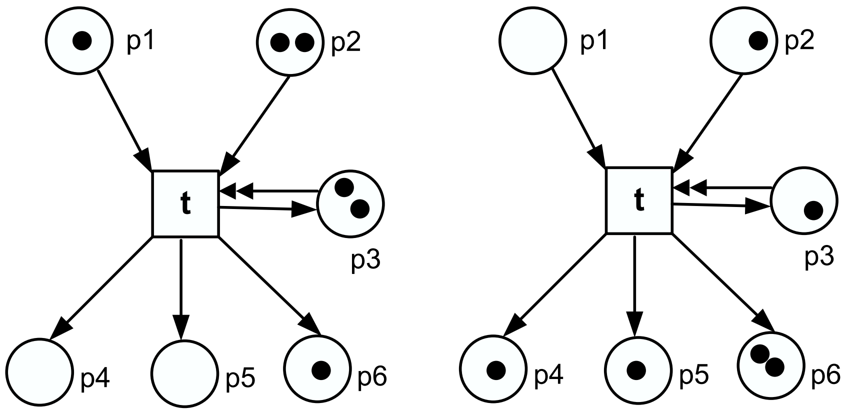

Fig. 9. An example reset net before and after firing a transition

Petri nets. If X is a set over Y, it could also be interpreted as a bag where each input place occurs once (E.g., $\bullet t$ can be interpreted both as a set of places $\{p1, p2\}$and as a bag where each place has one token each $1p1 + 1p2$). In the latter case, this bag can be straightforwardly interpreted as a marking with one token for each place. $M(p)$ returns the number of tokens in a place p if p in the domain of $M(p \in dom(M))$ and we define M(p)=0 if $p \notin dom(M)$. For a reset net N, $\mathbb{M}(N)$ is used to represent a set of all possible markings of N. In Figure 9, the left net shows a marking M where there is a token in $p1$, two tokens in $p2$, two tokens in $p3$, and one token in $p6$ (denoted as a multiset $p1 + 2p2 + 2p3 + p6$). $M(p1)$ returns 1 where as $M(p4)$ returns 0.

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

Definition 3.3. (Marked) Let $N = (P, T, F, R)$ be a reset net and $M \in \mathbb{M}(N)$: marked$(M) = \{p \in dom(M) \mid M(p) > 0\}$.

We use notations such as $M \leq M'$, $M > M'$, $M + M'$, and $M \dot{-} M'$ for comparison and operations on markings. $M \leq M'$ iff $\forall_{p \in P} M(p) \leq M'(p)$. $M > M'$ iff $\forall_{p \in P} M(p) \geq M'(p) \wedge \exists_{p \in P} M(p) > M'(p)$. $M + M'$ are multisets such that $\forall_{p \in P} : (M + M')(p) = M(p) + M'(p)$. Similarly, $M \dot{-} M'$ are multisets such that $\forall_{p \in P} : (M \dot{-} M')(p) = M(p) \dot{-} M'(p)$ where for any natural numbers a, b: $a \dot{-} b$ is defined as max$(a - b, 0)$. The use of $\dot{-}$ instead of $-$ ensures that the number of tokens can never be a negative number.

The $\sqsubseteq$ relation indicates that M marks fewer or the same places as M'.

Definition 3.4. ($\sqsubseteq$) 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 not $M' \sqsubseteq M$.

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 fewer places than M'.

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

Definition 3.5. (Enabling rule) Let N be a reset net, $t \in T$, and $M \in \mathbb{M}(N)$. Transition t is *enabled* at M, denoted as $M[t\rangle$, if and only if $\forall p \in \bullet t : M(p) \geq 1$.

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

Definition 3.6. (Forward firing) Let $N = (P, T, F, R)$ be a reset net, $t \in T$ and $M, M' \in \mathbb{M}(N)$.

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

In Figure 9, transition t is enabled at marking $p1 + 2p2 + 2p3 + p6$ as $\bullet t = p1 + p2$ and t may fire. When transition t fires, it removes a token each from its input places $p1$ and $p2$, removes all tokens from its reset place $p3$, and puts one token each in its output places $p3$, $p4$, $p5$, $p6$, resulting in the marking $p2 + p3 + p4 + p5 + 2p6$.

We now define the concepts of reachability and coverability of markings from a given marking in a reset net. It is possible to fire a sequence of transitions from a given marking in a reset net resulting in a new marking using the forward firing rule defined above. This sequence of transitions is represented as an occurrence sequence σ. A marking M' is reachable from another marking M in a reset net, if there is an occurrence sequence leading from M to M'.

Definition 3.7. (Reachability) Let $N = (P, T, F, R)$ be a reset net and $M, M' \in \mathbb{M}(N)$. M' is reachable in N from M, denoted $M \overset{N,*}{\to} M'$, if there exists an occurrence sequence $\sigma \in T$ such that $M \overset{\sigma}{\to} M'$.

The *reachability set* is the set of markings that can be reached from a given marking M in a reset net after firing all possible occurrence sequences and denoted as $N[M\rangle$. If all places in a reset net are bounded, the reset net is also bounded and hence, it is possible to generate a finite reachability set. If a place is unbounded, the reachability set contains an infinite number of states (*an infinite state space*). In such cases, reachability of a marking cannot be determined but coverability can be determined.

For reset nets, reachability is undecidable for nets with more than two reset arcs but coverability is decidable using a backward firing algorithm [9]. A marking M_2 is said to be *coverable* from another marking M_1 in a reset net if there is a reachable marking M' from M_1 such that M' is bigger than or equal to M_2.

Definition 3.8. (Coverability) Let $N = (P, T, F, R)$ be a reset net and $M_1, M_2 \in \mathbb{M}(N)$. M_2 is coverable from M_1 in N, if there exists a marking M' such that $M' \in N[M_1\rangle$ and $M' \geq M_2$.

Next, two notations: *projection* and *filtering* are presented to allow operations on selected places of a marking in a reset net. 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.

Definition 3.9. (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

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

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 the set of places in P' and M_2 has more tokens in $p2$.

The notation $M \upharpoonright P'$ is used to alter a marking based on a set of places P', i.e., unlike $M[P']$ the domain may be modified (extend or reduce the set of places).

Definition 3.10. (Filtering $\upharpoonright$) Let $N = (P, T, F, R)$ be a reset net, $M \in \mathbb{M}(N)$ and $P \subseteq P'$. $M \upharpoonright P'$ returns a function such that dom$(M \upharpoonright P') = P'$ and

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

Let $M = p1 + p2 + p3$ and $P' = \{p1, p2\}$. $M \upharpoonright P' = p1 + p2$ and dom$(M \upharpoonright P') = \{p1, p2\}$. If $P' = \{p1, p2, p3, p4\}$, $M \upharpoonright P' = p1 + p2 + p3 + 0p4$ and dom$(M \upharpoonright P') = \{p1, p2, p3, p4\}$.

Next, we define the notion of *Backward firing* that is used to generate coverable markings for a reset net by firing transitions backwards. We denote $M' \dashrightarrow^t M$ if it is possible to fire a transition t backwards starting from a marking M and resulting in another marking M'.

Definition 3.11. (Backward firing) Let $N = (P, T, F, R)$ be a reset net and $M, M' \in \mathbb{M}(N)$.

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

For places that are not reset places, the number of tokens in M' is determined by the number of tokens in M for p and the production and consumption of tokens. If a place is an output place of t and not a reset place, one token is removed from $M(p)$ if $M(p) > 0$. If a place is an input place of t and not a reset place, one token is added to $M(p)$. For any reset place p, $M(p) \leq F(t, p)$ because it is emptied when firing and then $F(t, p)$ tokens are added. We do not require $M(p) = F(t, p)$ for a reset place p because the aim is coverability and not reachability. M', i.e., the marking before (forward) firing t, should *at least* contain the *minimal* number of tokens required for enabling t and resulting in a marking of *at least* M. Therefore, only $F(p, t)$ tokens are assumed to be present in a reset place p.

In Figure 9, it is possible to fire transition t backwards in the right net at marking $M = p2 + p3 + p4 + p5 + 2p6$ as there is exactly one token in place $p3$ that is a reset place of t as well as one of its output places (i.e., $M[R(t)] = p3 = t \bullet [R(t)]$). This results in a marking M' where one extra token is put into all input places of t and one token is removed from all output places of t that are not reset places of t. Finally, one token is put into all reset places of t. This results in a marking $M' = p1 + 2p2 + p3 + p6$. Note that this marking M' has only one token in $p3$ whereas the marking shown in the left net has two tokens in $p3$. This is because when firing backwards, it is impossible to know how many tokens were originally present in a reset place. Hence, the backwards firing rule returns a *coverable* marking and not necessarily a reachable marking.

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

Definition 3.12. (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 $M \overset{*}{\rightarrow} M'$ and $M[P'] \sqsubset M'[P']$.

This *superM* predicate is used to decide whether an OR-join should be enabled. For the set of input places of an OR-join P' and the current state of the workflow represented as marking M, the *superM* predicate holds if it is possible to mark more input places of the OR-join and hence, the OR-join should wait for synchronisation.

All the definitions for reset nets presented in this subsection will be used to formally define the YAWL OR-joins semantics in subsection 3.3. First, we present how a YAWL net can be formally defined and then show how a YAWL net can be mapped to a reset net.

3.1.2. *Formalisation of YAWL models*

A YAWL specification is formally defined as a nested collection of Extended Workflow Nets (EWF-nets) by van der Aalst and ter Hofstede [2]. A YAWL specification supports hierarchy and a composite task unfolds into another EWF-net. For our purposes, it suffices to consider only one net in isolation. A YAWL net formally corresponds to what was termed an “EWF-net” and we present here the definition of a YAWL net and refer the reader elsewhere [2] for a formal definition of a YAWL specification.

Definition 3.13. (YAWL net [2]) An YAWL net N is a tuple $(C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ such that[a]

- 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,

[a]Note that we are using basic mathematical notations such as $\not\rightarrow$ for a partial function, $\mathbb{N}$ for natural numbers, and $\mathbb{N}^{inf}$ for $\mathbb{N} \cup \{inf\}$.

- split: $T \to \{AND, XOR, OR\}$ specifies the split behaviour of each task and join: $T \to \{AND, XOR, OR\}$ specifies the join behaviour of each task,
- rem: $T \nrightarrow \mathbb{P}(T \cup C \setminus \{i, o\})$ specifies the cancellation region for a task,
- nofi: $T \nrightarrow \mathbb{N} \times \mathbb{N}^{\text{inf}} \times \mathbb{N}^{\text{inf}} \times \{\text{dynamic, static}\}$ specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).

While firing rules exist for YAWL nets [2], in this paper these are not needed as state analysis is relegated to the reset net level. In a YAWL net, tasks can be connected directly to other tasks but conditions cannot be connected directly to other conditions. To enable a mapping to reset net, an implicit condition is introduced between two tasks if there is a direct connection between them. We call these nets where all implicit conditions are made explicit, explicit YAWL nets or (eYAWL-nets) [22]. All YAWL nets are assumed to be first transformed into eYAWL-nets for OR-join analysis.

Definition 3.14. (eYAWL-net) Let $N = (C, \mathbf{i}, \mathbf{o}, T, F, \mathit{split}, \mathit{join}, \mathit{rem}, \mathit{nofi})$ be a YAWL net, the corresponding eYAWL-net is defined as $(C^{ext}, \mathbf{i}, \mathbf{o}, T, F^{ext}, \mathit{split}, \mathit{join}, \mathit{rem}, \mathit{nofi})$ where

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

Let N be an eYAWL-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 $x \bullet = \{y | (x, y) \in F^{ext}\}$ as before.

3.2. *Mapping from YAWL with OR-joins to reset nets*

This subsection describes how a YAWL net with OR-joins can be transformed into a reset net. But first, a number of abstractions from YAWL nets are proposed thus enabling a mapping to reset nets.

Even though YAWL is based on Petri nets, the YAWL language supports complex constructs such as multiple instances, hierarchy, cancellation, OR-joins that are not easy to model in Petri nets. For OR-join analysis, cancellation plays a very important role and it is not possible to abstract from cancellation regions. However, other constructs such as multiple instances and hierarchy do not affect the OR-join analysis and hence, it is possible to abstract from them.

- *composite tasks and hierarchy*: A YAWL specification could contain multiple YAWL nets with hierarchical structure and a composite task is used to unfold these nets. We propose to treat a net as a *flat* net, and ignore the hierarchical structure. That is, composite tasks will be treated as black boxes. 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 in the composite task, it will not influence the decision to enable another OR-join at a higher level. Hence, composite tasks can be abstracted and the hierarchical structure of a YAWL specification is ignored.
- *multiple instances*: A multiple instances task can be used to execute a particular task a number of times in parallel. For this abstraction, it is assumed that the engine is capable of keeping the multiple instances apart, and that it will synchronise them at the end. Therefore, for the purposes of OR-join analysis the execution of a multiple instances task is the same as the execution of an atomic task.
- *internal conditions of a task*: The YAWL semantics [2] defines a task as having internal conditions and state transitions. As they represent intermediate states, it is possible to consider only one internal state together with the input and output conditions of a task during OR-join analysis.
- *other perspectives*: We focus our attention on the control flow perspective only. We propose to abstract from the data perspective. In particular, branching conditions of XOR-split and OR-split tasks are not taken into account when considering the execution flow. We also abstract from the resource perspective, the operational perspective and exception handling considerations.

After abstractions from the features mentioned above, a net is considered as having tasks with various split and join behaviours, possible cancellation sets and explicit and implicit conditions. For a net without OR-joins, there is then a straight-forward mapping into a reset net. Figure 10 illustrates the approach taken in the transformation for a net *without* OR-joins. This is made possible by the fact that some concepts of YAWL such as multiple instances, composite tasks and internal state transitions of a task can be abstracted. In general, a condition is mapped onto a place, and a task onto two sets of transitions and an intermediate place. The transitions in the first set start the task (modelling the join behaviour), whereas the transitions in the second set complete it (modelling the split behaviour). In Figure 10, labels S and E are used to denote start transitions and end transitions. Condition names are also used to differentiate transitions within a particular set (e.g., transition $t_S^{p_1}$ represents the start transition for task t that has p_1 as its input).

For a YAWL net with OR-joins to be converted into a reset net, it is necessary to remove the OR-joins first as they have non-local semantics. As mentioned in subsection 2.4, we propose to define the formal semantics of a general OR-join in YAWL by treating other OR-joins in the net as XOR-joins. A net *with* OR-joins can be transformed into a reset net by first singling out one OR-join (the one that we would like to decide whether it can be enabled), removing it from the net, and then changing other OR-joins in the net to XOR-joins. A transformation function **transE2WF** converts a net without OR-joins into the corresponding reset net. Function R stores all transitions and its associated reset places. As a task in a YAWL net is now split into a number of t_S and t_E transitions depending on the split and join behaviour, a place p_t is introduced for each task t to represent an internal place between t_S and t_E. The flow relation F' is also modified so that the newly introduced places in P' and transitions T' are properly connected. As we abstract from

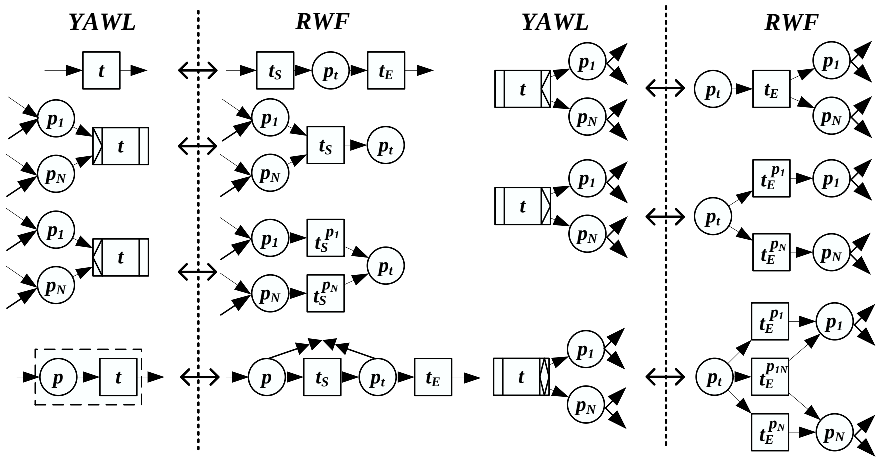

Fig. 10. Reset net transformations for YAWL split and join behaviours

multiple instances tasks and the function $nofi$ is not considered during the transformation.

Definition 3.15. (transE2WF) Let $N = (C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ be an eYAWL-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

$$\begin{aligned}
T_{start} = \{&t_S | t \in T \ \wedge \ join(t) = AND\} \\
&\cup \{t_S^p | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}, \\
T_{end} =\{&t_E | t \in T \ \wedge \ split(t) = AND\} \\
&\cup \{t_E^p | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\} \\
&\cup \{t_E^x | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing\}, \\
F' =\{&(p, t_S) | t \in T \ \wedge \ join(t) = AND \ \wedge \ p \in \bullet t\} \\
&\cup \{(t_S, p_t) | t \in T \ \wedge \ join(t) = AND\} \\
&\cup \{(p_t, t_E) | t \in T \ \wedge \ split(t) = AND\} \\
&\cup \{(t_E, p) | t \in T \ \wedge \ split(t) = AND \ \wedge \ p \in t\bullet\} \\
&\cup \{(p, t_S^p) | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\} \\
&\cup \{(t_S^p, p_t) | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\} \\
&\cup \{(p_t, t_E^p) | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\} \\
&\cup \{(t_E^p, p) | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\} \\
&\cup \{(p_t, t_E^x) | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing\} \\
&\cup \{(t_E^x, p) | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \varnothing \wedge \ p \in x\}, \\
R =\{&(t_E, \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C)) | t \in T \ \wedge \ split(t) = AND\} \\
&\cup \{(t_E^p, \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C)) | t \in T \ \wedge split(t) = XOR \\
&\wedge p \in t\bullet\} \\
&\cup \{(t_E^x, \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C)) | t \in T \ \wedge split(t) = OR \\
&\wedge x \subseteq t\bullet \ \wedge x \neq \varnothing\}
\end{aligned}$$

$$\begin{aligned}
&\cup\{(t_E,\varnothing)|t\in T\setminus dom\,rem\ \wedge\ split(t)=AND\}\\
&\cup\{(t_E^p,\varnothing)|t\in T\setminus dom\,rem\ \wedge\ split(t)=XOR\wedge p\in t\bullet\}\\
&\cup\{(t_E^x,\varnothing)|t\in T\setminus dom\,rem\ \wedge\ split(t)=OR\wedge\ x\subseteq t\bullet\ \wedge x\neq\varnothing\}\\
&\cup\{(t,\varnothing)|t\in T_{start}\}.
\end{aligned}$$

The transformation rule defined for a YAWL net without OR-joins can be used for all tasks that are not OR-joins. For OR-join tasks, all except the one under consideration are transformed as if they are XOR-joins and the OR-join under consideration is removed. The reason that this OR-join can be removed is because only reachable markings that mark the input places of the OR-join are relevant when deciding whether the OR-join should be enabled.

Definition 3.16. (transE2WFOJ) Let N be a YAWL net with OR-joins and N^{ext} be the eYAWL-net of N and *o-j* be an OR-join task under consideration. The function transE2WFOJ$(N, o\text{-}j)$ returns $N' = (P, T'', F'', R)$ such that P, T', T_{start}, T_{end}, F', and R are as defined in Definition 3.15 and T'' and F'' are defined as follows:

$$\begin{aligned}
&T''=T'_{start}\cup T_{end},\\
&T'_{start}=T_{start}\ \cup\ \{t^p_{start}|t\in T\wedge\ join(o\text{-}j)=OR\wedge t\neq o\text{-}j\ \wedge\ p\in\overset{N}{\bullet}t\},\text{ and}\\
&F''=F'\cup\{(p,t^p_{start})|p\in\overset{N}{\bullet}t\ \wedge\ t\in T\wedge join(t)=OR\wedge t\neq o\text{-}j\}\\
&\qquad\cup\{(t^p_{start},p_t)|p\in\overset{N}{\bullet}t\ \wedge\ t\in T\wedge join(t)=OR\wedge t\neq o\text{-}j\}.
\end{aligned}$$

Naturally, a given marking M in an eYAWL-net can be linked to a marking M_R in the corresponding reset net for a particular OR-join in consideration. For all the conditions that exist in an eYAWL-net, they will be marked exactly the same as in the corresponding marking and the newly introduced places in the reset net have zero tokens. The marking marks all the places in the reset net which correspond to conditions in N with the same number of tokens. This marking is referred to as the corresponding marking and is denoted as M_R.

3.3. *Definition and illustration of OR-join semantics*

You may recall that informally an OR-join task is 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 waits for synchronisation. The following steps are proposed to decide whether an OR-join task *o-j* should be enabled at a marking M of a given net.

(1) translate the YAWL net into a reset net for a given *o-j*,
(2) apply *superM* predicate to determine whether it is possible to mark more input places of *o-j* in the reachable markings from M, and
(3) if at least one of the input places of *o-j* is marked at M and *superM* evaluates to *FALSE*, *o-j* is enabled at M. Otherwise, *o-j* is not enabled at M.

Definition 3.17. (OR-join semantics) Let $N = (C, T, F, R)$ be an eYAWL-net, M be a marking of N, *o-j* be the OR-join task under consideration, N_R = transE2WFOJ$(N, \textit{o-j})$ be the corresponding reset net and $M_R \in \mathbb{M}(N_R)$. *o-j* is enabled at M iff $\exists p \in \bullet\textit{o-j} : M(p) \geq 1$ and $\neg$ superM$(N_R, M_R, \bullet\textit{o-j})$.

We now describe how the transformations will be performed for a net with two OR-join tasks C and D as shown in Figure 11. Note that an explicit condition c_{BD} has been added for the implicit condition between tasks B and D. Consider a marking $M = c1 + c_{BD}$ where the OR-join analysis for task D is performed as there is a token in c_{BD}, one of the input places of D. As the two 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} from M. Figure 12 shows an equivalent reset net for the eYAWL-net in Figure 11 for the OR-join analysis of D. Note that the other OR-join task in the net, C, is treated as an XOR-join task and modelled with two start transitions, one for $c1$ and one for $c3$. Also note that D has been removed from the net. There is a corresponding marking for the reset net, $M_R = c1 + c_{BD}$. The sequence $c1 + c_{BD} \overset{C^{c1}_{start}}{\rightarrow} p_C + c_{BD} \overset{C_{end}}{\rightarrow} c4 + c_{BD}$ exists and hence, it is possible to reach $M'' = c4 + c_{BD}$ from M. Recall that the *superM*(N, M, P') predicate returns true if it is possible to reach a marking from M which marks more places in a set of places P'. Therefore, superM(transE2WFOJ$(N, \textit{o-j}), M_R, \bullet\textit{o-j})$ returns true as $M_R \overset{*}{\rightarrow} M''$ and $M_R[\{c4, c_{BD}\}] \sqsubset M''[\{c4, c_{BD}\}]$. As it is possible to reach a marking that marks more input places of the OR-join, then D is not enabled at M.

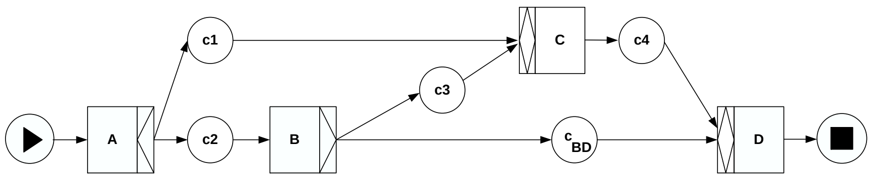

Fig. 11. An eYAWL-net N with OR-join tasks C and D

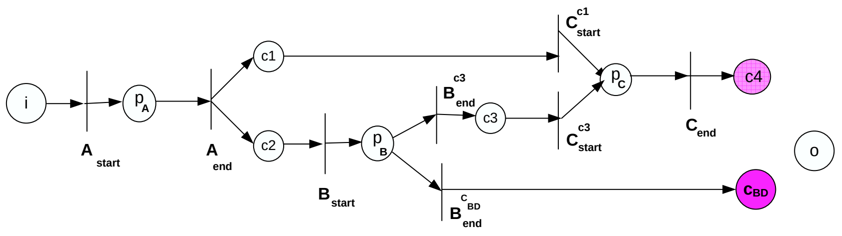

Fig. 12. A reset net for OR-join analysis of task D in Figure 11

Next, we look at how the new OR-join semantics can be operationalised and an algorithmic approach towards determining OR-join enablement is examined.

4. Operationalising the OR-join

The main objective of the OR-join algorithm is to determine, for a given OR-join, whether there is a reachable marking M' from M such that more input places of that OR-join could be marked at M'. This analysis is performed by first transforming an eYAWL-net (with OR-joins) into a reset net for a given OR-join task using the function transE2WFOJ and then calling the proposed OR-join algorithm. The algorithm works backwards by computing the predecessor coverable markings for a given marking, as opposed to the forward approach used in coverability tree algorithms. The algorithm is based on backward search techniques for Well-Structured Transition Systems (WSTSs) [4,8,11,12,13].

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 [8,17]. 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 [17]. In the context of reset nets, the backward firing rule (cf. Definition 3.12) is used to define $pb(M)$ for a given marking. The backwards reachability analysis can be performed to decide the coverability [8,11,17] provided that $\leq$ is decidable and $pb(y)$ exists and can be effectively computed [13]. In Appendix A, we present some background definitions on WSTSs and demonstrate that $pb(y)$ can be computed.

We now present the various procedures that operationalise the coverability question for reset nets using the backwards algorithm for WSTSs. We then demonstrate how to perform OR-join enablement analysis using the coverability results.

4.1. *Procedures*

The procedure $\mathbf{Coverable}$ returns a Boolean value to indicate whether a marking y is coverable from a marking x of a reset net.

```
PROCEDURE $\mathbf{Coverable}$ (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 by Leuschel and Lehmann [17].

```
PROCEDURE $\mathbf{FiniteBasisPred}^*$ (SET Marking $I$): SET Marking
SET Marking $K, K_{next}$;
BEGIN
  $K := I$; $K_{next} := K \cup \mathbf{pb}(K)$;
  while not $\mathbf{IsUpwardEqual}(K, K_{next})$ 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. [12].

```
**PROCEDURE** $\mathbf{IsUpwardEqual}$ (SET Marking $K$,SET Marking $K_{next}$): Boolean
 **BEGIN**
    return $K = K_{next}$;
**END**
```

The procedure $\mathbf{pb}(I)$ returns $pb(I)$ such that $pb(I) = \bigcup_{x \in I} pb(x)$ [17].

```
**PROCEDURE** $\mathbf{pb}$ (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" [17].[b] Note that, in our case, this minimum is one as there are no weighted arcs. We will make use of backward firing rule. For each transition $t \in T$, it is possible to determine whether an M' exists such that $M' \overset{t}{\dashrightarrow} M$. Hence, $pb(M) = \{M' | \exists_{t \in T}\, M' \overset{t}{\dashrightarrow} M\}$.

```
**PROCEDURE** $\mathbf{pb}$ (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**
```

The coverability findings of a reset net are then applied to the OR-join analysis. At the current marking M, we know that one or more of its input places are marked. For each

[b] Note that the algorithm described by Leuschel and Lehmann [17] is incorrect. $pb(M)$ is defined in a rather naive way by Leuschel and Lehmann [17] . Applying $pb(M)$ to the empty marking yields a counter example, since it is not a finite basis for $\uparrow Pred^*(\uparrow \{M\})$.

of the unmarked input places, we need to test whether there is a coverable marking from M that marks that place. The test marking is constructed by marking each input places of the OR-join that is marked in the current state with one token each and adding one token also for an unmarked input place. A test marking is created for each of the unmarked input places. If none of these markings are coverable from M, then the OR-join is enabled at M.

Let (N, M) be a marked eYAWL-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 $M' \in \mathit{Pred}^*(M_w)$ 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 $\mathbf{OrJoinEnabled}$ (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
```

4.2. *Worked example*

Throughout this paper, several examples have been presented which indicate that it is a non-trivial task to decide if an OR-join is enabled or not. Clearly, the algorithm can be applied successfully to these situations. To illustrate its inner working in some detail we use one last example.

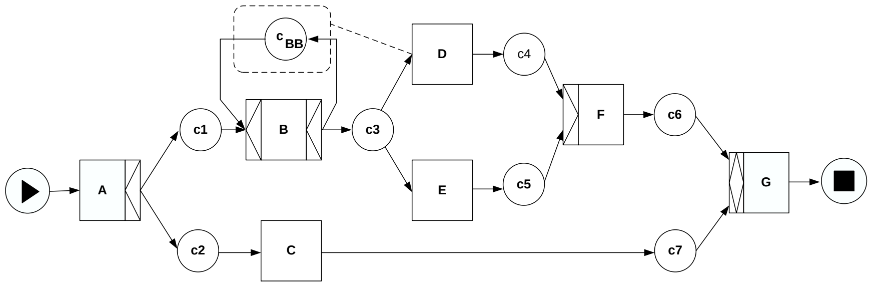

Fig. 13. A YAWL net with an OR-join task G and cancellation

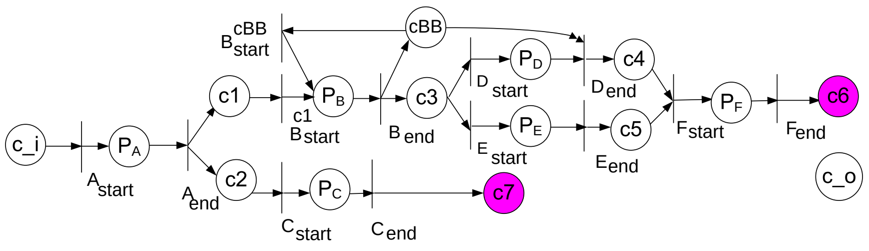

Fig. 14. A corresponding reset net for Figure 13 (note the double-headed arrow denoting the reset arc from C_{BB} to D_{end})

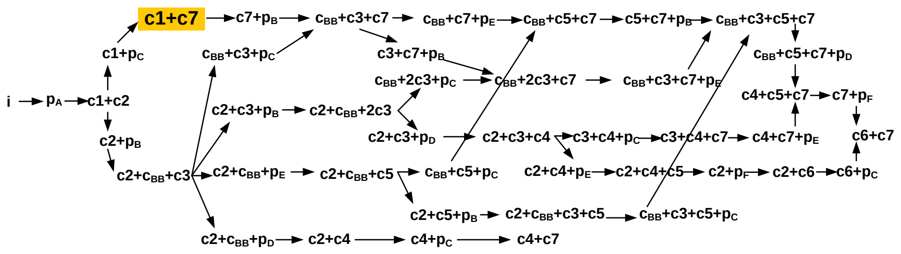

Fig. 15. Illustration of backwards reachability analysis

Consider a marking $M = c1 + c7$ in Figure 13 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 starts with a procedure call like this: $\mathbf{OrJoinEnabled}(c1 + c7, \{c6, c7\})$. $Y := \{c6 + c7\}$ and for $M_w = c6 + c7$, a finite basis of all the predecessors of $c6 + c7$ is obtained. Figure 15 illustrates the backwards reachability analysis [12], 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.

In the previous example, we have seen that even for an OR-join with two input conditions, a number of iterations are needed to generate the finite set of coverable markings. When there are many input conditions to an OR-join, the process needs to be repeated for each unmarked input condition. Furthermore, the analysis needs to be carried out every time the workflow changes its state. Hence, it is easy to see that the algorithm can become quite expensive when we have a large net with many tasks and conditions. Therefore, one potential drawback of such a generic approach to an OR-join semantics without structural restrictions is an efficient implementation. To achieve our combined objective of a generic formal OR-join definition with an efficient implementation, we propose two restriction techniques in the next section.

5. Restriction techniques

For an OR-join analysis, it is possible to consider only a portion of the net that is relevant to the analysis and refrain from exploring those paths that do not affect the OR-join enabling behaviour. This would correspond to the notion of slicing in program analysis [?]. To improve the performance of the OR-join evaluation algorithm, two forms of restriction are proposed: *structural restriction* and *active projection*. *Structural restriction* involves removing from a net tasks and conditions 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 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 make use of the reset net mappings and define how restriction operations are applied to a reset net.

5.1. *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. We will call them *goal places*. Function *res* describes how a reset net could be constructed so that only the transitions and places that are on the path to goal places are included in the restricted net.

Definition 5.1. (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 $(N' = \text{res}(N, G))$ 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
$R' = \{(t, R(t) \cap P') | t \in T'\}$.

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.

Figure 16 describes how function *res* of N 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, and 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 as input place of 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'. 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. the reset arcs connecting p_4 and t_7).

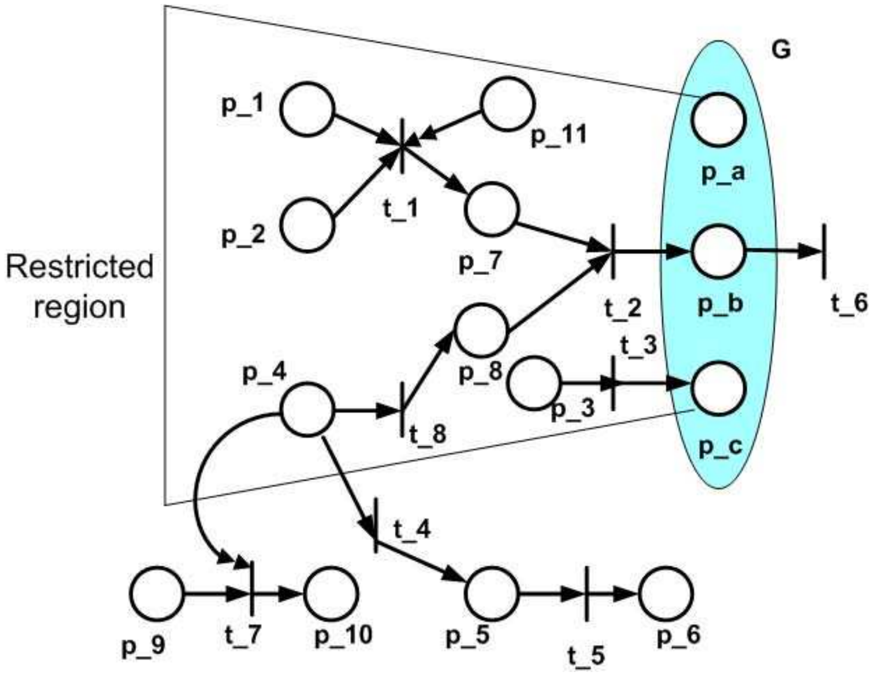

Fig. 16. Restriction diagram

5.2. *Active projection*

In addition to applying structural restriction to a 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, this transition can be safely excluded 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 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 *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 17 shows the effect of the active projection function *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'.

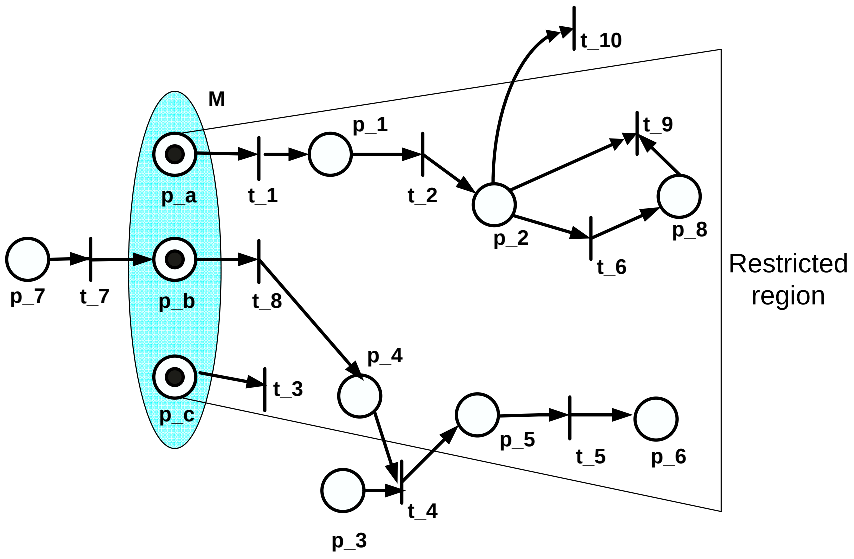

Fig. 17. Active projection diagram

Definition 5.2. (ap(N,M)) Let $(N,M) = ((P,T,F,R),M)$ be a marked reset net. $N' = \text{ap}(N,M) = (P',T',F',R')$ is the *active projection* of (N,M) where

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

An OR-join task *o-j* is enabled at a marking M of an eYAWL-net N, if it is not possible to reach a marking M_n such that $M \xrightarrow{*} M_n$ and $M[\bullet o\text{-}j] \sqsubset M_n[\bullet o\text{-}j]$. To determine whether *o-j* should be enabled at M, the following analysis is carried out. Let $N_R =$ transE2WFOJ$(N, o\text{-}j)$ be the reset 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, the search space can be reduced by first applying the structural restriction and active projection techniques so that $N_R' = \text{res}(\text{ap}(N_R, M_R), G) = (P',T',F',R')$. It is then possible to 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.

Complete proofs for these two restriction techniques are provided in Appendix B.

6. Implementation

The OR-join analysis algorithm as described in Section 4 together with the structural restriction and active-projection techniques from Section 5 have been implemented in the

YAWL engine[c]. A number of YAWL nets have been tested and OR-join enabling results are as expected. The observations also indicate that the two restriction techniques significantly reduce the execution times for OR-join analysis.

The execution times of the OR-join enabling algorithm for a number of YAWL nets are presented. 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 the OR-Join call. **AProject+SRestrict** indicates that active projection is applied first and then, structural restriction is applied before the 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 times consecutively. Furthermore, this process has been repeated ten times for sampling. Average execution times with confidence intervals (95%) are provided. All the figures are in **milliseconds** and are rounded to one decimal place.

6.1. *Matching OR-split and an OR-join*

The net in Figure 7 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. We can see that by utilising the restriction techniques, it is possible to reduce the execution times. In this case, structural restriction does not influence the execution time as we are dealing with a small net. The active projection technique, on the other hand, has significant effects on the execution time as all possible combinations of an OR-split do not need to be considered. Even for a small net, it can be seen that restriction techniques can reduce the time it takes to perform the OR-join evaluation.

6.2. *Loop and cancellation*

Figure 6 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, it is clear that the combined restriction techniques significantly reduce the evaluation time. The difference between the execution times for OR-join analysis with structural restriction and without any restrictions is minimal as most tasks and conditions in this YAWL net will be in the restricted net as well.

6.3. *Larger loop and cancellation*

Table 3 presents 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 13. OR-join evaluation for both markings returns FALSE. This YAWL net also contains a loop and cancellation on the path to G. In

[c]http://sourceforge.net/projects/yawl/

OR-join: E	Marking: $c1 + c2 + c6$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
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
Restriction techniques	Duration in millisecs (100 calls)
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

Table 1. Execution times for the OR-join analysis of the net in Figure 7

OR-join: E	Marking: $c2$ returns TRUE
Restriction techniques	Duration in millisecs (100 calls)
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

Table 2. Execution times for the OR-join analysis of the net in Figure 6

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.

6.4. *Multiple OR-joins*

To demonstrate the impact of structural restriction on OR-join analysis, the YAWL net in Figure 18 that contains a number of tasks which have no impact on the OR-join task F will be used. 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 5. 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 considerable differences 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

OR-join: G	Marking: $c1 + c7$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
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
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	587.4 ± 7.2
AProject+SRestrict	585.9 ± 9.9
AProject	568.7 ± 9.8
SRestrict	11195.3 ± 12.1
NoRestrict	11198.0 ± 21.9

Table 3. Execution times for the OR-join analysis of the YAWL net in Figure 13

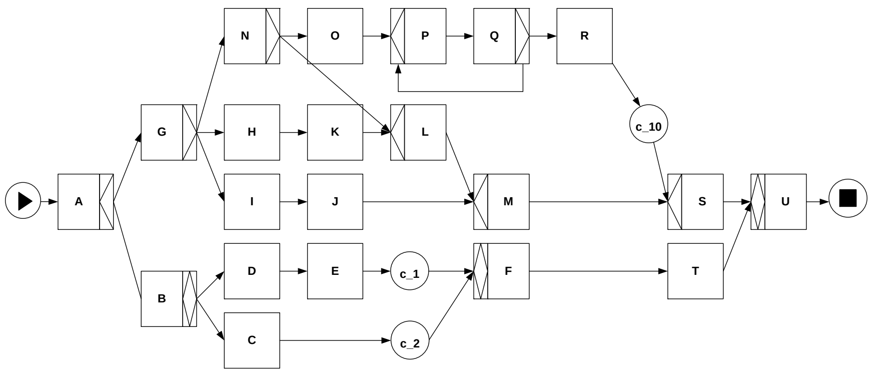

Fig. 18. A YAWL net with OR-join tasks F and U

given in Table 4. 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 techniques 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 significantly reduce the execution time of an OR-join evaluation.

OR-join: F	Marking: $c_{AG} + c_{BD} + c_2$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
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
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	275.0 ± 7.1
AProject+SRestrict	304.7 ± 86.5
AProject	276.5 ± 9.3
SRestrict	1198.4 ± 9.4
NoRestrict	3492.2 ± 23.8

Table 4. Execution times for Task U from the net in Figure 18

OR-join: U	Marking: $c_{AG} + c_{TU}$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
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

Table 5. Execution times for Task F from the YAWL net in Figure 18

7. Visa application example - A YAWL workflow with cancellation regions and OR-joins

In the previous section, a number of small YAWL nets are presented to illustrate the various features of the OR-join algorithm. We now demonstrate the effectiveness of the proposed algorithm using a real-life process model: **visa application for general skilled migration to Australia**. This process is modelled “as is” using publicly available information from Department of Immigration and Multicultural Affairs website[d]. The process starts when a visa application is received by the immigration department and ends when a decision is made to grant or to deny the visa. The model represents the process from the viewpoint of a case officer who handles the visa application. The resulting YAWL workflow contains four nets *Overview*, *Perform main assessment*, *Check basic requirements*, and *Allocate marks*.

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

Figure 19 shows the *Overview* net and the typical process flow is explained first. When an application is received, the case officer opens a file for the applicant, processes visa application fees and performs an initial assessment. If the application is found to be *complete*, the officer continues with the main assessment. If the application is *incomplete*, he/she sends an acknowledgement letter to the applicant requesting further documentation. This is modelled as an XOR-split task after the task *Perform initial assessment*. The *Perform main assessment* task is modelled as a composite task and the internal working of this task is captured in another net. After completing the main assessment, the case officer might request more information, or he/she is ready to make a decision. This is modelled as an XOR-split task. Condition $c9$ represents a state where the officer is waiting for further documentation from the applicant. If he/she receives the requested information, the main assessment task is performed again. On the other hand, the designated time period could have expired, and the officer decides to perform the main assessment again if possible to stop processing the application if it cannot be processed further with existing documentation. Before the officer makes a decision, he/she checks to see if there is any change in circumstances that need to be considered. The *Check circumstances changes* task has a cancellation region containing condition $c2$. Removing a token from $c2$ indicates that there is no need to wait for further circumstances changes. The officer then makes a decision to either grant or deny the visa after taking into account any changed circumstances. The *Make decision* task is an OR-join task with two inputs $c5$ and $c7$. A token in $c5$ indicates that there are changes that need to be considered. If a decision is made to deny the visa, the applicant is then notified. Otherwise, the visa is granted. The process ends when the *Finalise application* task is executed.

While an application is being processed, it is possible for two events to occur. First, an applicant can decide to withdraw his/her application and secondly, an applicant can notify the immigration department of changes in his/her circumstances - such as change of address, correction of errors, etc. Hence, the task *Open applicant file* is modelled as an AND-split to indicate that two tasks (*Wait for possible withdrawal request* and *Monitor circumstances changes*) could occur in addition to the main flow starting with *Process application fees* task. These two tasks represent *external triggers* that can be enabled when a notification is received from the applicant. These triggers affect the main flow of the process and are also captured in the model. Note that there is no YAWL notation to represent external triggers. As a result, these two tasks are represented as normal tasks. A token in $c6$ indicates that there is some circumstances change that needs to be taken into account. Similarly, a token in $c4$ indicates that a request has been received for withdrawal. The *Cancel application* task is modelled as an OR-join and when it fires, it removes tokens from conditions and tasks in the net before the *Make decision* task. The application can be withdrawn until a decision is made. The *Make decision* task removes tokens from conditions and tasks associated with the trigger for application withdrawal.

In the *Overview* net, the *Perform main assessment* is represented as a composite task and it is unfolded into the YAWL net with the same name. Similarly, there are two composite tasks: *Check basic requirements* and *Allocate marks* in the *Perform main assessment* net and they are also unfolded into two YAWL nets with the corresponding names. Figure 20 shows the three subnets in the process. More details about this process model can be found

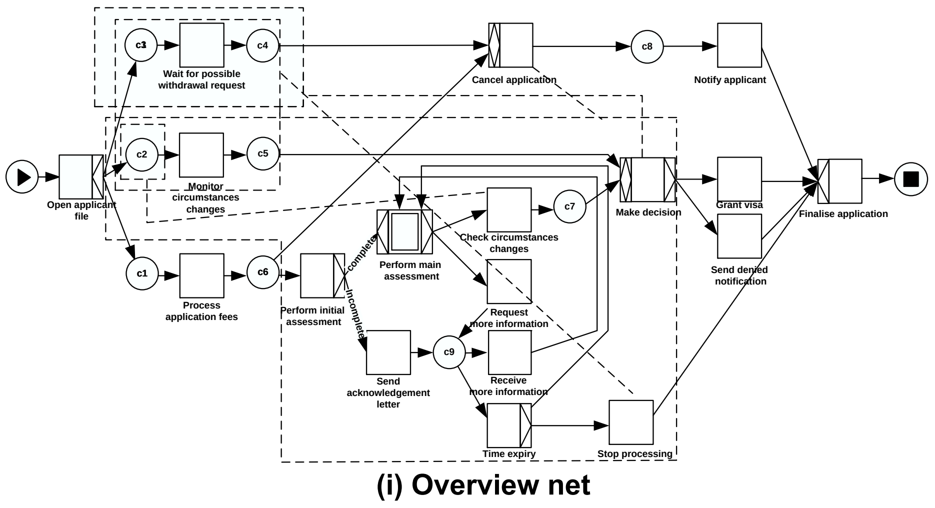

Fig. 19. *Overview*: the main YAWL net in the visa application process

in [22]. As the *Perform main assessment* net and the *Check basic requirements* net do not contain OR-join, we focus our attention on the *Allocate marks* net.

The *Allocate marks* net represents the process for calculating the marks received by each applicant. This visa class uses a points system where marks are given based on the applicant's circumstances assessed on several criteria. The total mark is then compared against the current pass mark for the visa class (110 points) to decide whether the visa will be granted. The net models how these points are allocated for 11 criteria to calculate the total points. Some criteria such as points for age, skills and English ability are relevant to all applicants, while others such as points for Australian qualifications and spouse skills are relevant to some applicants only. The *Decidable applicable categories* task is modelled as an OR-split where a decision is made regarding the relevance of a particular criterion. The net completes with an OR-join task that waits for synchronisation of all active paths before calculating the total points allocated to the applicant.

7.1. *Enabling the cancel application task*

The *Cancel application* task is within the main net *Overview* (Figure 19). It is modelled as an OR-join with two inputs $c4$ and $c6$. A token in $c4$ indicates that the case officer has received a request for withdrawal. A token in $c6$ indicates that the application fees have been processed. The *Cancel application* task is enabled when there are tokens in both inputs ($c4$ and $c6$) or a token in either $c4$ or $c6$ and it is not possible for a token to arrive at the other empty input place. Consider a marking $c3+c6$ where payment has been processed but there is no request for withdrawal. At marking $c3+c6$, the OR-join behaves as expected (returns FALSE) and the *Cancel application* task is not enabled. Now, consider a marking $c1 + c4$ where the request for withdrawal has been received but payment has not been processed. In this case, the *Cancel application* task is not enabled until payment has been processed. That is, a marking $c4+c6$ enables the task. This is used to model business logic

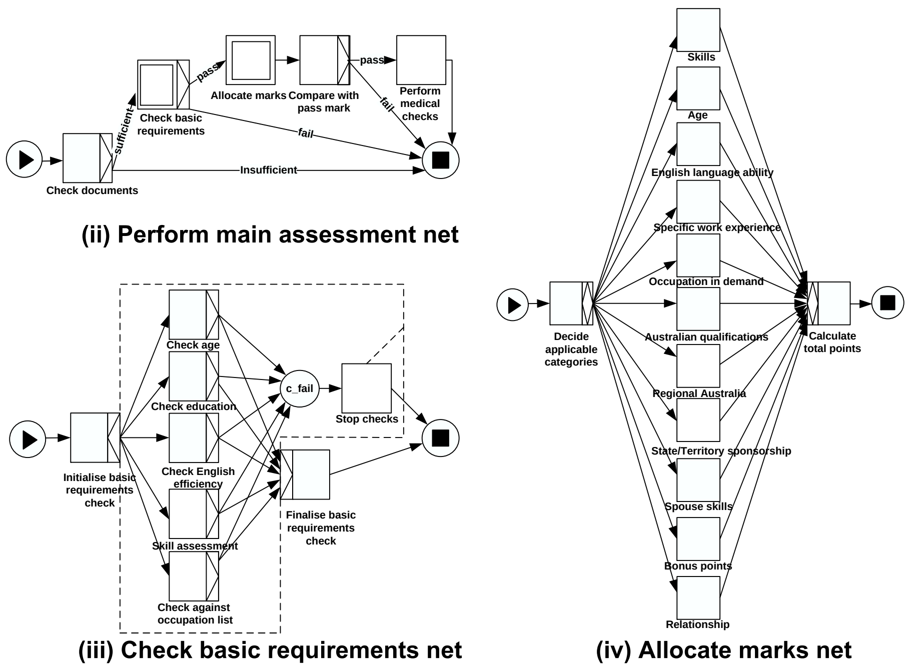

Fig. 20. YAWL nets: *Perform main assessment*,*Check basic requirements*,*Allocate marks*

stated as “You can withdraw your application by advising the Adelaide Skilled Processing Centre in writing at any time before a decision is made. Any charges that you paid at the time of application are usually not refunded.” [7]. Perhaps, a more common scenario is where an applicant decides to withdraw the application while it is being processed. For instance, marking $c4 + c7$ represents a state where the request for withdrawal has been received and the application is awaiting a decision. In this case, the *Cancel application* should go ahead and it is enabled at marking $c4 + c7$. The analysis returns TRUE as it is not possible to receive a token in $c6$ from reachable markings of $c4 + c7$. Table 6 shows the execution times for *Cancel application* task in Figure 19 for these markings. In general, we can see that execution times with optimisation techniques are much faster than the ones without optimisation. For marking $c3 + c6$, the execution times for OR-join algorithm with both types of restrictions are at least four times faster when compared to the execution times without any restrictions. Similarly, for marking $c1 + c4$, the execution times are at least six times faster when compared to the ones without any restrictions. Also, for marking $c4+c7$, the execution times are at least ten times faster when compared to the ones without any restrictions.

7.2. *Enabling the make decision task*

The *Make decision* task is within the main net *Overview* (see Figure 19). It is modelled as an OR-join with two inputs $c5$ and $c7$. A token in $c5$ indicates that there are some circumstances changes that must be taken into account. A token in $c7$ indicates that the

OR-join: Cancel application	Marking: $c3 + c6$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	401.3 ± 45.6
AProject+SRestrict	489.7 ± 50.3
AProject	614.7 ± 8.9
SRestrict	609.7 ± 0.6
NoRestrict	2114.3 ± 9.2

OR-join: Cancel application	Marking: $c1 + c4$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	213.3 ± 8.9
AProject+SRestrict	296.7 ± 0.6
AProject	437.3 ± 15.5
SRestrict	437.3 ± 0.6
NoRestrict	1833.3 ± 23.4

OR-join: Cancel application	Marking: $c4 + c7$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	177.3 ± 9.2
AProject+SRestrict	208.3 ± 9.2
AProject	443.0 ± 8.7
SRestrict	182.3 ± 8.9
NoRestrict	1937.7 ± 27.1

Table 6. Execution times for enabling analysis of the *Cancel application* task in Figure 19

case officer is ready to make a decision. The *Make decision* task is enabled when there are tokens in both inputs ($c5$ and $c7$) or a token in either $c5$ or $c7$ and it is not possible for a token to arrive at the other empty input place. Consider a marking $c5 + c6$ where the applicant has reported a change in circumstances such as a change of residential address and the case officer has not finished processing the application. The OR-join behaves as expected and the *Make decision* task is not enabled (returns FALSE). Consider another marking $c7$ where the case officer is ready to make the decision and there are no circumstance changes to consider. In this case, the *Make decision* task should go ahead (returns TRUE) and it is enabled at marking $c7$. Table 7 shows the execution times for enabling analysis of the *Make decision* task in Figure 19 for these markings. For marking $c5 + c6$, we can see that the execution times for the algorithm with both types of restrictions are nearly nine times faster when compared to the execution times without any restrictions. This is because the *Make decision* task is located in the middle of the net and a number of tasks and conditions that follow the OR-join can be removed before the OR-join call. Similarly, for marking $c7$, the execution times are over 40 times faster when compared to the ones without any restrictions. This is because in addition to the abstractions using structural restriction, active projection removes most tasks and conditions as they cannot be enabled from current marking $c7$.

OR-join: Make decision	Marking: $c5 + c6$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	942.7 ± 17.9
AProject+SRestrict	989.7 ± 35.8
AProject	5442.7 ± 38.8
SRestrict	1255.3 ± 9.2
NoRestrict	8422.0 ± 87.1

OR-join: Make decision	Marking: $c7$ returns TRUE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	198.0 ± 8.6
AProject+SRestrict	161.3 ± 8.4
AProject	5192.7 ± 24.0
SRestrict	233.0 ± 5.2
NoRestrict	8244.7 ± 32.4

Table 7. Execution times for enabling analysis of the *Make decision* task in Figure 19

7.3. *Enabling the calculate total points task*

The *Calculate total points* task is within the *Allocate marks* net (see Figure 20). The net is modelled as a structured net with an OR-split (*Decide applicable categories*) and an OR-join (*Calculate total points*). The *Calculate total points* task waits to synchronise until all active paths leading out of the *Decide applicable categories* task are completed. Consider a scenario where marks are to be allocated for four criteria: skills, age, English ability and work experience. Consider a marking $c_{SC} + c_{AC} + c_{EC} + c_{DW}$ where marks for skills, age and English ability have been allocated but marks for work experience have not been processed. The *Calculate total points* task is not enabled (returns FALSE) at that marking as it needs to wait. It is only enabled when marks have been allocated for all four criteria (i.e., at the marking $c_{SC} + c_{AC} + c_{EC} + c_{WC}$). Table 8 shows the execution times for the OR-join enabling analysis of the *Calculate total points* task for these markings.

This example highlights the need for the optimisation techniques. The net contains an OR-split with 11 possible paths resulting in $2^{11} - 1 = 2047$ possible combinations and hence, resulting in a large state space. As a result, OR-join analysis (without optimisation) does not complete after letting it process for several hours. We can observe that active projection makes a huge difference as it is only necessary to consider active paths and not all possible combinations from the OR-split, thus significantly reducing the state space. Structural restriction does not have an effect here as the OR-join is the last task in the net.

Calculate total points	Marking: $c_{SC} + c_{AC} + c_{EC} + c_{DW}$ returns FALSE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	30088.3 ± 76.9
AProject+SRestrict	28255.0 ± 609.3
AProject	21463.0 ± 86.1
SRestrict	N/A
NoRestrict	N/A

Calculate total points	Marking: $c_{SC} + c_{AC} + c_{EC} + c_{WC}$ returns TRUE
Restriction techniques	Duration in millisecs (100 calls)
SRestrict+AProject	29864.7 ± 140.2
AProject+SRestrict	22552.0 ± 474.6
AProject	21036.7 ± 726.4
SRestrict	N/A
NoRestrict	N/A

Table 8. Execution times for the OR-join analysis of the *Calculate total points* task

8. Related work

In van der Aalst et al. [1], the authors summarise the problems associated with capturing the non-local semantics of an OR-join connector in EPCs. Kindler proposes a semantic framework for formally defining the non-local semantics of EPCs including the OR-join [15]. The author states that “a single transition relation cannot precisely capture the informal semantics of EPCs”. It is proposed to define the non-local semantics in terms of a pair of transition relations and a semantic definition using techniques from fixed point theory is presented [15,16]. The paper by Cuntz et al. describes how to “calculate this semantics of an EPC in an efficient way by employing Kleene’s fixed-point theorem and different techniques from symbolic model checking” [5]. Kindler shows how a pair of transition relations can be calculated to determine the non-local semantics and proposes the use of reduced ordered binary decision diagrams (ROBDDs) to represent huge sets of states and huge transition relations for optimisation [16]. Their motivation is similar to ours in the sense that the author also attempts to “define a mathematically sound semantics that comes as close as possible to the informal semantics”. This approach represents an alternative approach to defining non-local semantics of the OR-join in the absence of cancellation [16].

9. Conclusion

Many workflow management systems and other process-aware information systems (e.g., ERP, CRM, and PDM systems), have problems supporting the OR-join semantics without restrictions. In this paper, we formally defined the semantics of an OR-join in the presence of cancellation regions, other OR-joins and (infinite) loops without adding structural restrictions. In addition, we operationalised this formal semantics and presented an efficient algorithm and implementation in the workflow language YAWL. To the best of our knowledge, no other semantics or workflow system implementation come close to supporting such a general OR-join, especially in the presence of arbitrary cancellation regions.

In our approach, reset nets are used as a formal basis for OR-join analysis to support workflows with cancellation. A transformation function to map a YAWL net with OR-joins into a reset net is provided. An OR-join evaluation algorithm which is based on the backward search techniques for Well-Structured Transition Systems is then proposed. The proposed semantics upholds the notion that an OR-join waits for synchronisation when necessary and continue when appropriate. Two optimisation techniques, structural restriction and active projection, are presented together with the findings from the implementation in the YAWL engine. A realistic process model with multiple cancellation regions and multiple OR-joins in a non-structured setting is presented to highlight the need for a general approach to OR-join semantics. To conclude the paper, we would like to emphasise that the results reported in this paper are not limited to YAWL. These results are equally applicable to any process modelling language that wishes to support advanced synchronisation constructs such as the OR-join and cancellation.

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. We also like to thank the three anonymous reviewers for their valuable feedback.

References

1. 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.
2. 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.
3. 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.
4. 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*, pages 313–321, New Brunswick, NJ, July 1996. IEEE Computer Society.
5. N. Cuntz, J. Freiheit, and E. Kindler. On the semantics of EPCs: Faster calculation for EPCs with small state spaces. In F.J. Nüttgens and M. Rump, editors, *Proceedings of EPK 2005*, pages 7–23, Hamburg, December 2005.
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. Department of Immigration and Multicultural Affairs. Processing your application. `http://www.immi.gov.au/migration/skilled/post_lodgmnt/proc_timefrms.htm` accessed on 21 April 2006, 2006.
8. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.
9. 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.

10. Eastman Software. *RouteBuilder Tool User's Guide*. Eastman Software, Inc, Billerica, MA, USA, 1998.
11. 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.
12. 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*, volume 1380 of *Lecture Notes in Computer Science*, pages 102–118, Campinas, Brazil, 1998. Springer-Verlag.
13. A. Finkel and Ph. Schnoebelen. Well-structured Transition Systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, April 2001.
14. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003.
15. 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.
16. E. Kindler. On the Semantics of EPCs: Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.
17. 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.
18. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
19. P. Rittgen. Modified EPCs and their Formal Semantics. Technical Report 99/19, Institute of Information Systems, University Koblenz-Landau, Koblenz, Germany, 1999.
20. Mark Weiser. Program slicing. In *ICSE '81: Proceedings of the 5th international conference on Software engineering*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.
21. S. White. Process Modeling Notations and Workflow Patterns. In L. Fischer, editor, *Workflow Handbook 2004*, pages 265–294, FL, USA, 2004. Future Strategies Inc.
22. P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-Based Analysis of the Control-Flow Perspective of UML Activity Diagrams. In L. Delcambre, C. Kop, H. Mayr, J. Mylopoulos, and O. Pastor, editors, *Proceedings of 24th International Conference on Conceptual Modeling ER 2005*, volume 3716 of *Lecture Notes in Computer Science*, pages 63–78, Klagenfurt, Austria, 2005. Springer-Verlag.
23. M.T. Wynn. *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*. PhD Thesis, Faculty of Information Technology, Queensland University of Technology, Nov 2006.
24. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International conference on Application and Theory of Petri nets and Other Models of Concurrency*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443, Miami, USA, June 2005. Springer-Verlag.

Appendix A. Well-structured transition systems and Reset nets

A.1. *Well structured transition systems (WSTS)*

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" [13]. The existence of a well-quasi-ordering over an infinite set of states ensures the decidability of termination and coverability properties [8,13].

Definition Appendix A.1. (Well-Structured Transition System [8]) 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) [17].

Definition Appendix A.2. (Upward-closed set [13]) 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 $I \subseteq Q$, $\mathit{Pred}(I)$, $pb(I)$ and $\mathit{Pred}^*(I)$ can be defined [17]. The immediate predecessors of I: $\mathit{Pred}(I) = \{x | x \rightarrow y \ \wedge \ y \in I\}$, all predecessor states of I, $\mathit{Pred}^*(I) = \{x | x \xrightarrow{*} y \ \wedge \ y \in I\}$ and $pb(I) = \bigcup_{y \in I} pb(y)$ where $pb(y)$ yields a finite basis of $\uparrow \mathit{Pred}(\uparrow \{y\})$ (i.e., $pb(y)$ yields a finite set such that $\uparrow pb(y) = \uparrow \mathit{Pred}(\uparrow \{y\})$) [17].

A finite basis of $\mathit{Pred}^*(\uparrow \{y\})$ is computed as the limit of the sequence $I_0 \subseteq I_1 \subseteq ...$ where $I_0 =^{\text{def}} \{y\}$ and $I_{n+1} =^{\text{def}} I_n \cup pb(I_n)$ [17]. The sequence eventually stabilises at some I_n when $\uparrow I_{n+1} = \uparrow I_n$ and a stabilisation point is reached that has the property $\uparrow I_n = \mathit{Pred}^*(\uparrow \{y\})$ [17]. As $\uparrow \{y\}$ is upward-closed, $\mathit{Pred}^*(\uparrow \{y\})$ is upward-closed [13].

A.2. *Linking WSTSs and reset nets*

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 $x' \in \mathit{Pred}^*(\uparrow \{y\})$ such that $x' \leq x$ (because $\leq$ is a wqo). We now show that $pb(M)$ can be effectively computed and that the property $\uparrow pb(M) = \uparrow \mathit{Pred}(\uparrow \{M\})$ holds.

Lemma Appendix A.1. *Let (N, M) be a marked reset net. $pb(M) = \{M' | \exists_{t \in T} M' \dashrightarrow^t M\}$ where $\uparrow pb(M) = \uparrow \mathit{Pred}(\uparrow \{M\})$.*

Proof. First, we will prove that $\uparrow pb(M) \subseteq \uparrow \mathit{Pred}(\uparrow \{M\})$.
Let $M_1 \in \uparrow pb(M)$, we need to show that $M_1 \in \uparrow \mathit{Pred}(\uparrow \{M\})$. There is an $M_2 \leq M_1$

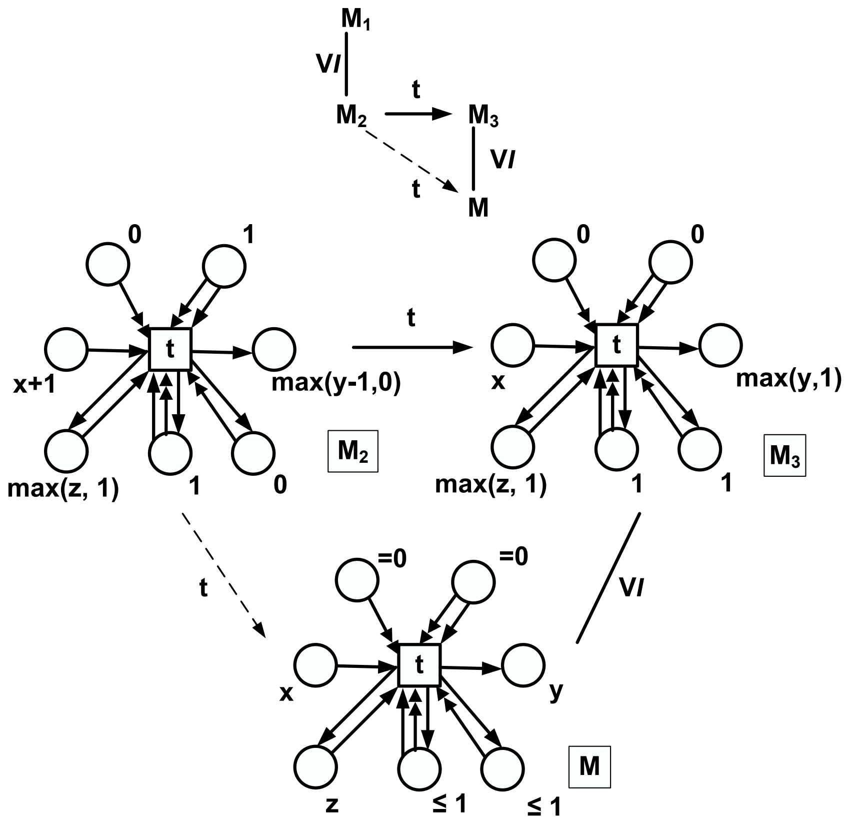

Fig. 21. Sketch of the first part of the proof

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 21. 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 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 tokens. 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 $M_1 \in \uparrow Pred(\uparrow \{M\})$.

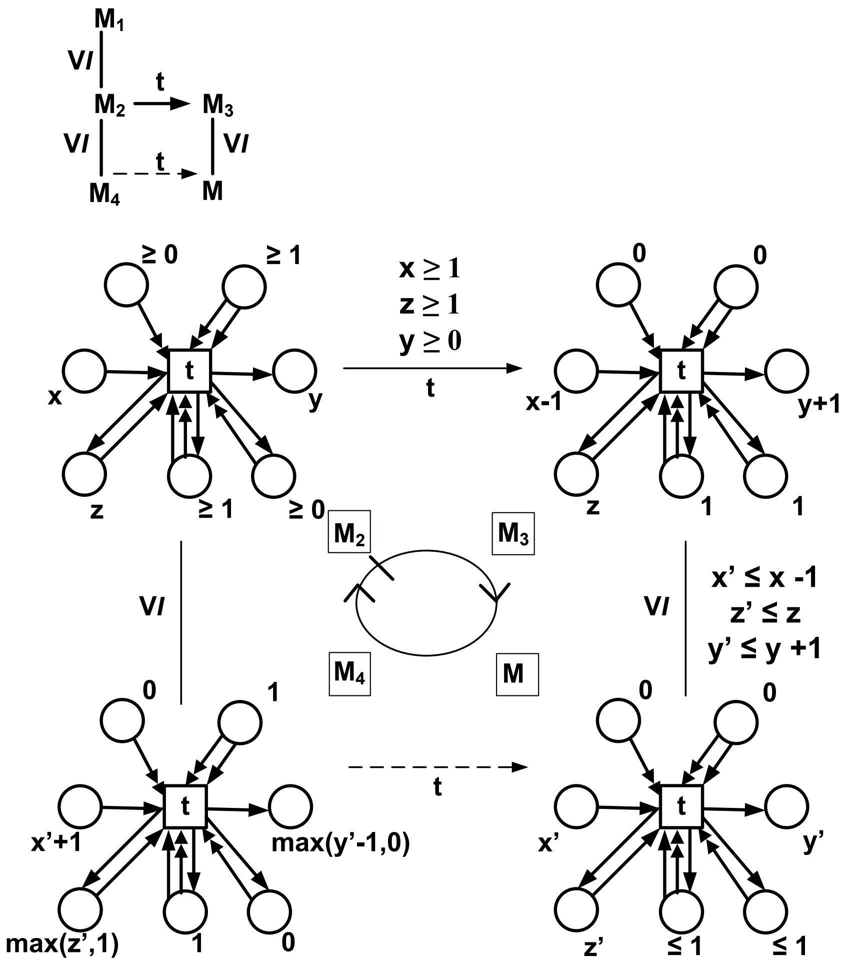

Fig. 22. Sketch of the second part of the proof

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 22. 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 22. 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 $M_1 \in \uparrow pb(M)$. □

Appendix B. Proofs for the restriction techniques

B.1. *Structural restriction*

Let N, N' be two reset nets such that $N' = \mathrm{res}(N, G)$, the structurally restricted net w.r.t. G. The following lemma will show that for any marking M_2 of N such that $M_1 \overset{N,*}{\rightarrow} M_2$, there is a corresponding marking $M_2^{'}$ of 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 Appendix B.1. *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 an occurrence 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 B.1, $t \notin T'$ implies that $t \bullet \cap P' = \emptyset$. Transitions that are not in the restricted net but in the occurrence 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 $\forall_{t \notin \sigma'} t \bullet \cap P' = \emptyset$, 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 $(N', M_1 \restriction P')$.

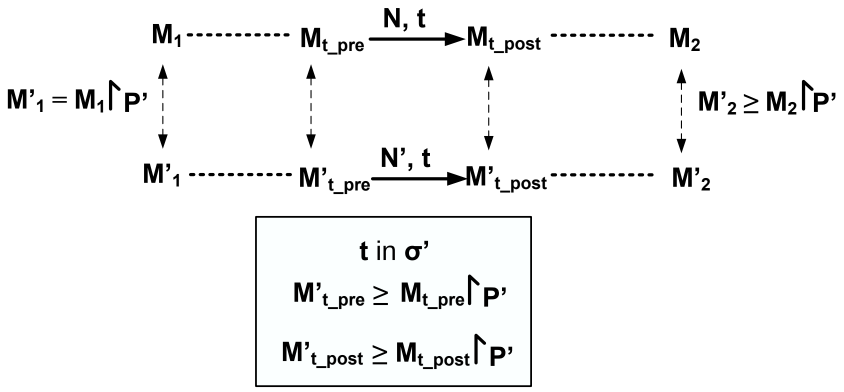

Fig. 23. Transition firings in both the original net and the restricted net

Next, we will prove that there exists $M_2^{''} \in \mathbb{M}(N)$ and $M_2^{'} \in \mathbb{M}(N')$ such that $M_1 \overset{N,\sigma'}{\rightarrow} M_2^{''} : (M_1 \restriction P' \overset{N',\sigma'}{\rightarrow} M_2^{'}) \wedge (M_2^{''} \restriction P' = M_2^{'})$. As shown before, $\sigma^{'}$ is enabled in $(N', M_1 \restriction P')$.

Figure 23 gives the states in both models for transitions that can be fired in both nets N and N'. Assume that $M_{t-pre} \restriction P' \geq M_{t-pre}^{'}$ and $t \in T'$. As $\overset{N'}{\bullet} t = \overset{N}{\bullet} t$, $t \overset{N'}{\bullet} = t \overset{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post} \restriction P' = M_{t-post}^{'}$. 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 24 gives the states in both models for transitions that can only be fired in the net

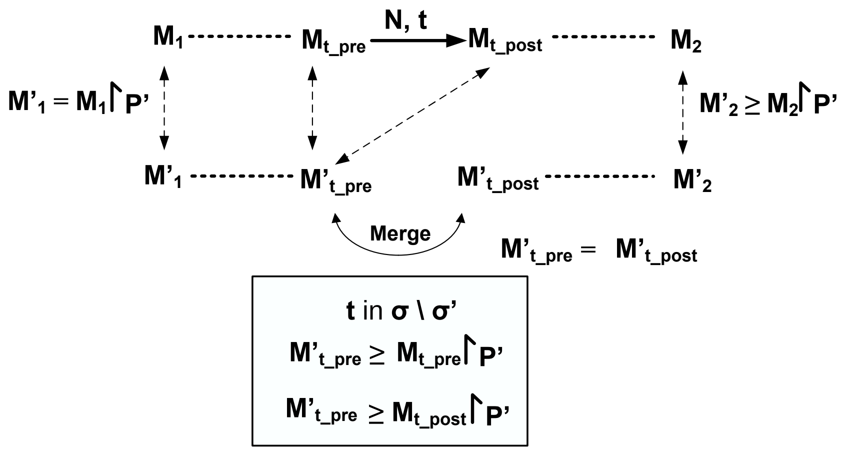

Fig. 24. Transition firings in the original net only

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 $M'_{t-pre} \geq M'_{t-post} \restriction P'$. $\square$

Now we will show 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 Appendix B.2. *Let $N = (P, T, F, R)$ be a reset net and $N' = res(N, G) = (P', T', F', R')$ is the restriction on G.*

$\forall_{M_1 \in \mathbb{M}(N), M'_2 \in \mathbb{M}(N')} (M_1 \restriction P' \overset{N',*}{\to} M'_2 \Rightarrow \exists_{M_2 \in \mathbb{M}(N)} M_1 \overset{N,*}{\to} M_2 \wedge M_2 \restriction P' = M'_2)$

Proof. Consider an occurrence sequence $\sigma : M_1 \restriction P' \overset{N',\sigma}{\to} M'_2$. We first show that σ is enabled in (N, M_1) and then that there is marking $M_2 : M_1 \overset{N,*}{\to} M_2 \wedge M_2 \restriction P' = M'_2$. 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).

Figure 25 gives the states in both models for transitions that can be fired in both nets N and 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 transitions $t \in \sigma$, hence: $M'_2 = M_2 \restriction P'$. $\square$

Corollary Appendix B.1. Let $(N, M_1) = ((P, T, F, R), M_1)$ be a marked reset net and $N' = \text{res}(N, G) = (P', T', F', R')$ its restriction on G.

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

Proof. ($\Rightarrow$) First, we will prove that $\exists_{M_2 \in \mathbb{M}(N)} (M_1 \overset{N,*}{\to} M_2 \wedge M_1[G] \sqsubset M_2[G])$ implies

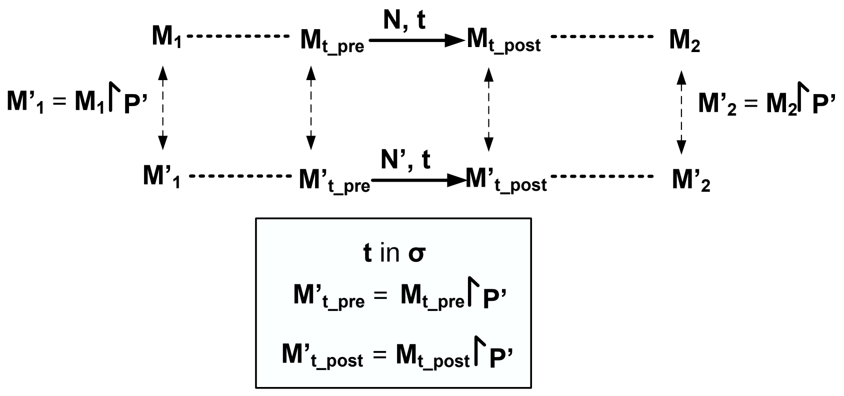

Fig. 25. Transition firings in both the original net and the restricted net

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 B.1, 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 B.2, 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 $M_1[G] \sqsubset M_2[G]$. $\square$

An OR-join task *o-j* is enabled at a marking M of an eYAWL-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 propose to perform the following analysis. Let N_R = transE2WFOJ$(N, o\text{-}j)$ be the reset 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, the search space can be reduced by applying the structural restriction so that $N_R' = \text{res}(N, G)$. Using Corollary B.1, it is possible to determine whether there is a marking $M_R' \in \mathbb{M}(N_R)$ such that $M_R \overset{N_R,*}{\rightarrow} M_R'$ 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.

B.2. *Active projection*

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 B.3 will demonstrate that for any marking $M_2 \in \mathbb{M}(N)$ reachable from M_1, there is a corresponding marking $M_2' \in \mathbb{M}(N')$ 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 Appendix B.3. *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.*

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

Proof. Consider an occurrence sequence $\sigma : M_1 \overset{N,\sigma}{\rightarrow} M_2$. First, we will prove that σ is enabled in $(N', M_1 \upharpoonright P')$. From Definition B.2, $t \in T \setminus T'$ 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., $\overset{N}{\bullet} t = \overset{N'}{\bullet} t \subseteq P'$). Figure 26 gives the states in both models. Assume that $M_{t-pre} \upharpoonright P' = M'_{t-pre}$ 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 $\overset{N'}{\bullet} t = \overset{N}{\bullet} t$, $t \overset{N'}{\bullet} = t \overset{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post} \upharpoonright P' = M'_{t-post}$. This can be repeated for all $t \in \sigma$, hence: $M'_2 = M_2 \upharpoonright P'$. □

Lemma B.4 will demonstrate that for any marking $M'_2 \in \mathbb{M}(N')$ reachable from $M_1 \upharpoonright P'$, there is a corresponding marking $M_2 \in \mathbb{M}(N)$ reachable from M_1 such that $M'_2 = M_2 \upharpoonright P'$.

Lemma Appendix B.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.*

$\forall_{M'_2 \in \mathbb{M}(N')}(M_1 \upharpoonright P' \overset{N',*}{\rightarrow} M'_2 \Rightarrow \exists_{M_2 \in \mathbb{M}(N)} M_1 \overset{N,*}{\rightarrow} M_2 \wedge M'_2 = M_2 \upharpoonright P')$

Proof. Consider an occurrence sequence $\sigma : M_1 \upharpoonright P' \overset{N',\sigma}{\rightarrow} M'_2$. We will prove that σ is enabled in (N, M_1). As σ is enabled in $(N', M_1 \upharpoonright P')$ 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 B.2, $t \in T \setminus T'$ 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., $\overset{N}{\bullet} t = \overset{N'}{\bullet} t \subseteq P'$). Figure 26 gives the states in both models. Assume that $M_{t-pre} \upharpoonright P' = M'_{t-pre}$ 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 $\overset{N'}{\bullet} t = \overset{N}{\bullet} t$, $t \overset{N'}{\bullet} = t \overset{N}{\bullet} \cap P'$ and $R'(t) = R(t) \cap P'$, we deduce: $M_{t-post} \upharpoonright P' = M'_{t-post}$. This can be repeated for all $t \in \sigma$, hence: $M'_2 = M_2 \upharpoonright P'$. □

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

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

Proof. 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 $\exists_{M'_2 \in \mathbb{M}(N')}(M_1 \upharpoonright P' \overset{N',*}{\rightarrow} M'_2 \wedge M_1[G] \sqsubset M'_2[G])$. Using Lemma B.3, we can show that $M_1 \overset{N,*}{\rightarrow} M_2$ implies $M_1 \upharpoonright P' \overset{N',*}{\rightarrow} M_2 \upharpoonright P'$. Hence, there exists an $M'_2 = M_2 \upharpoonright P'$ such that $M_1[G] \sqsubset M'_2[G]$.

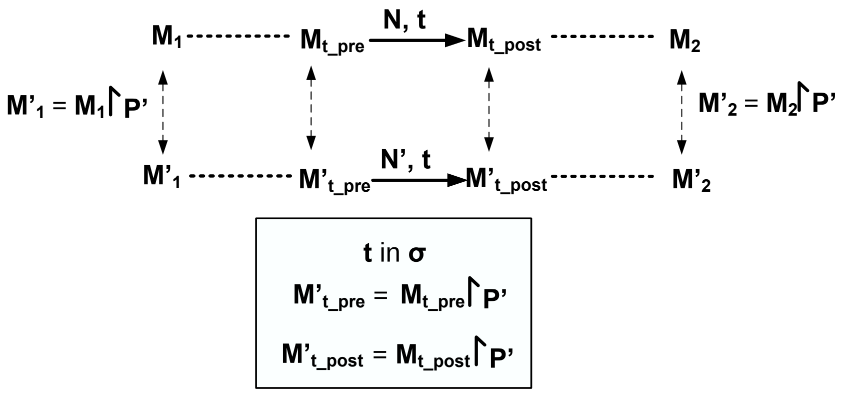

Fig. 26. Transition firing in a restricted net (active projection)

Second, we will prove that $\exists_{M_2' \in \mathbb{M}(N')} M_1 \upharpoonright P' \overset{N',*}{\rightarrow} M_2' \wedge M_1[G] \sqsubset M_2'[G]$ 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 B.4 and assuming $M_1 \upharpoonright P' \overset{N',*}{\rightarrow} M_2'$, there exists a M_2 such that $M_1 \overset{N,*}{\rightarrow} M_2$ and $M_2 \upharpoonright P' = M_2'$. Since $G \subseteq P$, $M_1[G] \sqsubset M_2'[G]$ implies $M_1[G] \sqsubset M_2[G]$. □