Sound Recoveries of Structural Workflows with Synchronization

Piotr Chrząstowski-Wachtel, Paweł Gołąb, and Bartosz Lewiński

Institute of Informatics, Warsaw University,
Banacha 2, PL 02-097 Warszawa, Poland
pch@mimuw.edu.pl, pawel.golab@mimuw.edu.pl, bartosz.lewinski@mimuw.edu.pl

Abstract. We consider communication places in workflow nets, where the connected transitions lie in parallel threads. When a workflow Petri net is constructed structurally, by means of refinements, such places cannot be modeled structurally. They are added after the net is constructed. Workflow nets constructed in a structural manner are sound, but addition of such communication places can result in losing the desired *soundness* property. However, there is a method to avoid such misplacement of communication places. We should limit the pairs of connected transitions to the ones that lie in truly parallel threads and to avoid cycles introduced by communication places.
Recovery transitions — special kind of transitions used as a powerful tool in dynamic workflow modeling — allow the manager to move tokens arbitrarily, when some unexpected situation happens. They were extensively studied and proved to be a useful tool in the workflow management [HA00]. They can be modeled as a kind of *reset transitions* with additional feature of depositing tokens taken from a specified region to particular places in this region, like it was proposed in [Ch06]. Moving tokens arbitrarily by the manager requires a lot of attention, since soundness of the net can easily be destroyed. In this paper we present a sufficient and necessary condition of soundness for a marking in a structured net with communication places. Verifying the condition turns out to be fast. The cost is linear with respect to the total number of places and transitions.

1 Introduction

Workflow management is an area, where workflow designers can prove correctness and flexibility of their models. It has been studied in numerous papers, like [WS09], [BPS09]. One of the major problems is how to organize communication between parallel tasks performed by communicating agents [BPM05]. Making communication pattern wrongly can easily lead to deadlocks, mutual waiting or leaving messages as trash, when they are deposited somewhere, and not consumed by anyone. The danger of bad design increases, when we talk about management that lasts in time and needs rearrangement due to some unexpected situations.

Composing workflow nets in a structural way was proposed in [ChBHO03]. A number of basic node refinement rules has been introduced. These rules include sequential split, parallel split, choice and loop splits. They reflect typical programming constructs like sequence of actions, an invocation of parallel threads, instruction of choice and a loop statement. The control of the workflow runs can be hence guarded by these constructs. Restricting the nets to the nets obtained from a single node by these structural constructs was proven in to guarantee soundness, as defined by [vdAtH00].

As it was already recognized in [ChBHO03], these constructs are not sufficient for typical needs of a workflow designer. In the cited paper a number of non-structural synchronization rules were proposed. By non-structural we mean here adding of new nodes and edges, which do not result as a refinement of existing nodes. Among them the most important was the synchronization of two parallel actions. When in two parallel threads A and B we want an action b from B to wait until an action a from A has been executed, we can model it by introducing a new communication place s with arcs leading from a to s and from s to b. Such construct we call a *synchronization* or *communication*, depending on whether we emphasize the fact that b must wait for a or that a has something important to communicate to b. Introducing such synchronization places can result in a possible deadlock or other unsound constructs, like trash tokens generation. In order to avoid misintroduction of such places, a criterion was proposed, which is a sufficient condition for soundness. The condition was based on the idea of the *refinement tree*, reflecting the history of refinements. It has been proven that refinement trees are unique up to an isomorphism for a given structural WF-net. In other words, if a WF-net is constructed structurally, then all the histories creating this net differ only in unimportant details (like the order of refinements in disjoint areas), resulting in the same refinement tree.

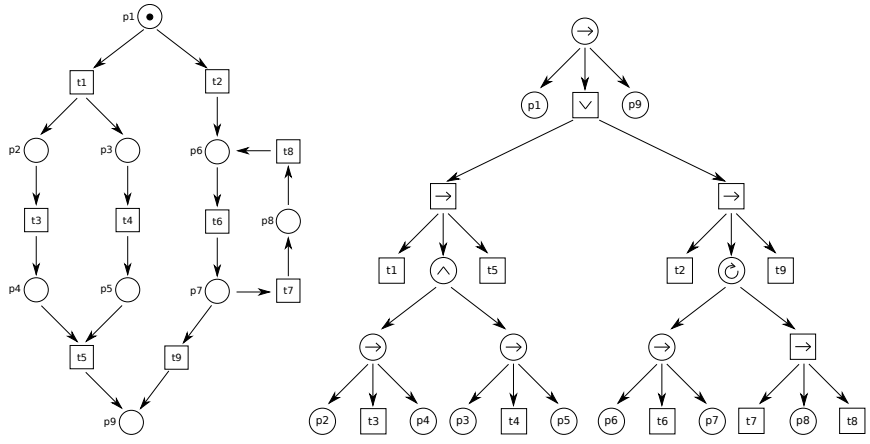

Fig. 1. Example *WF*-net and an corresponding refinement tree

In dynamical workflows it is often desired that the control is being changed during the lifetime of the workflow execution. Such situations are quite normal, especially when workflows describe processes that last for a long time (months or even years). Sometimes the manager decides to detour from the anticipated control flow and would like to "correct" the flow manually moving tokens around. Situations of this kind can happen in particular, when for instance under some time pressure we decide to skip several actions or when we decide that some part of the workflow should be repeated due to unexpected failures, which were not foreseen during the design. In such cases we would like to support the workflow management by allowing the manager to perform only sound rearrangements of the flow. When no such restriction would be set, the manager, quite possibly without understanding side effects, could make undesirable changes. This can lead to unwanted behavior, making the net unsound.

One of the main problems with such on-the-fly changes of the markings is to determine the *impact area*, which is the least part of the net, called *region*, which is affected by the rearrangement of tokens. The refinement tree gives us precise information — in order to define the impact area caused by any changes in the net, it suffices to find the latest common predecessor of the nodes, where the changes were made. The nodes which are not descendants of this node are not affected by these changes.

The changes we consider are of two kinds. First of them is the addition of places or transitions in an unstructured manner. An example of such useful addition is the introduction of a place joining two transitions, which are in (different) parallel threads. If such a place connects transition t with transition r, then the intention is to suspend the execution of r until t is executed. Clearly the introduction of such a place can result in a deadlock. In [ChBHO03] a strong sufficient condition was presented guaranteeing net soundness after such insertion of a communication place. It turns out that if an inserted place connects such two transittion-type leaves t and r in the refinement tree that no choice-split or loop-split node is found on the path from t to r and if no cycle can be detected in the net after such insertion, then the resulting net is sound.

The second kind of change is of dynamical matter. We allow the manager to modify the marking by arbitrary moving tokens around some *region*. A region is understood as the net unfolded from a single place-type node in the refinement tree. Inside a region we consider the so-called *recovery transitions*, which remove tokens from the whole area and restore them in arbitrarily chosen places. Our goal is to find conditions, which would protect the manager from depositing tokens on such places, that the resulting marking would be unsound, hence not properly terminating.

2 Refinement Rules

This section is a short reminder of *WF*-nets refinement rules introduced in [ChBHO03]. The idea behind is that having defined the refinement rules preserving certain network properties, we are able to analyse *WF*-networks that were

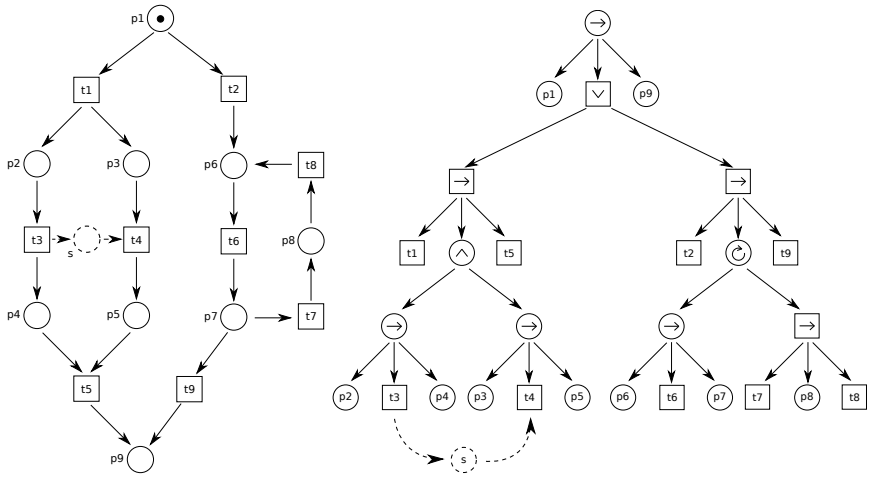

Fig. 2. Example of net synchronization with corresponding refinement tree

constructed by using those rules. To construct such network one starts with a single place, and then repeatedly applies context-free rules on network elements. One of biggest advantages of using refinement rules approach is construction trace called *refinement tree* that contains much information about the network structure. An example of such tree and corresponding network can be found on figure 1. The rest of this section covers basic rules presented in [ChBHO03]. All described rules are depicted on figure 3.

The first two rules are called *sequential splits*. They are used to create linear sequence of places and transitions, like $p2 \rightarrow t3 \rightarrow p4$ on figure 1. It's an example of splitting a single place. There are two kinds of sequential splits depending on the node type they are applied to. We call them *sequential place split* and *sequential transition split* respectively. Splits of this type introduce partial order of transitions in sound transition firing sequences. Splits replace the chosen node with three other nodes: the first and the last are of the same type and have either the same inputs or outputs as the original node, respectively. The third node is the one in the middle that connects two other nodes, so is of opposite type.

The next two rules are equivalent to logical AND and OR gates respectively. The first of them applies to places and the second one to transitions and both are replacing node with two copies of it.

The first split called *parallel split* introduces two parallel threads that will be executed simultaneously. In sound transition firing sequences transitions from different parallel paths can be safely swapped (if partial order defined by other splits is preserved). Examples of such paths are $p2 \rightarrow t3 \rightarrow p4$ and $p3 \rightarrow t4 \rightarrow p5$ on figure 1.

The second split, called *choice split*, defines two alternative paths that the process can follow. During a single process run, transitions of only one of the

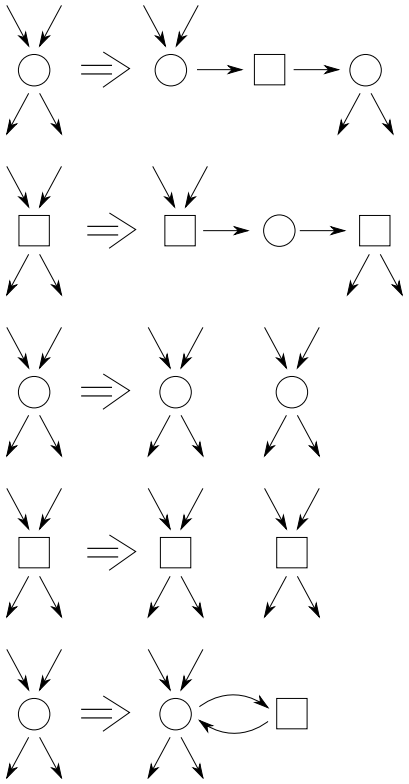

Fig. 3. Basic *WF*-nets refinement rules. Starting from top: *sequential place split*, *sequential transition split*, *parallel split*, *choice split* and *loop split*.

paths can be enabled. Examples of such paths are $t1 \to \ldots \to t5$ and $t2 \to \ldots \to t9$ on figure 1.

The last split type introduces loops and therefore is called a *loop split*. A loop example with nodes later splitted by sequential split rules can be found on figure 1.

3 Definitions

In this section we'll present some definitions and notation conventions that will be used in the rest of the article.

Siblings. For the node v that is a child of sequential split type node p, let right siblings be defined as siblings that occur after v in p's children order. The definition of left siblings of v is analogous.

Prenodes. We define *Prenodes*(v) set as follows. Let p_i be the i-th node on the path from *root* to v. Then for each p_i of sequential split type, *Prenodes*(v) includes p_i, left siblings of p_{i+1} and their subtrees.

This definition corresponds to the original refinement tree, without using additional edges provided by synchronisations.

Existential marking function. Below we present notation and definition for existential marking function which returns information if in a given set of nodes any places are marked.

$$\mathrm{M}^{?}(V) = \begin{cases} 0 & \text{if } \mathrm{M}(v) = 0 \text{ for each } v \in V \\ 1 & \text{otherwise} \end{cases}$$

Structured *SWF*-net (Synchronised WorkFlow network) is an extension of *WF*-net defined as a 7-tuple $\langle P, T, F, s, e, S, C \rangle$, where

1. $\langle P, T, F, s, e \rangle$ is a standard structured *WF*-net with the set of places P, transitions T, flow function F the source place s, and the exit place e.
2. S — the set of synchronisation places (semaphores)
3. C — the set of edges joining semaphores and synchronised transitions. It is easy to see that $C \cap F = \emptyset$

When two transitions t_1 and t_2 are synchronised via place s, we denote t_1 as $in(s)$ and t_2 as $out(s)$.

4 Soundness Characterisation

The main goal of this chapter is to find properties of marking M in *SWF*-net that guarantee soundness. Before introducing these properties we'll define two auxiliary sets *Before* and *After*, that will help us in those properties formulation and we'll explore some important properties of them.

4.1 *Before* and *After* sets

Intuitively, *Before*(v) and *After*(v) sets include places and transitions, that are over or under vertex v in *WF*-net graph respectively. In the case of *SWF*-nets we consider only such nodes that are not synchronisation places during *Before* and *After* sets construction.

We define *Before*(v) set as follows. A leaf l is in *Before*(v) if and only if there exists a predecessor q of l being a left-sibling of some node lying on the path from the root to v inclusively. Similarly for $After(v)$ we consider right siblings instead. We ignore the synchronisation places.

An important feature of *Before* and *After* sets is that *Before*(v) and *After*(v) sets are not containing v itself, so immediate conclusion from *Before* and *After* sets definitions is that *Before*$(v) \cap$ *After*$(v) = \emptyset$ for each node v.

It is worth to explore, how *Before* and *After* sets are constructed in the case of loops. We can distinguish two cases depending on whether loop contains the node for which these sets are constructed or not.

Let us consider loops from the first case. Let l be the loop that was created by splitting place p_l and transition t_l. In this case *Before* and *After* sets will either contain leaves only from p_l subtree or only from t_l subtree.

In the second case the *Before* and *After* sets can either contain all the leaves of the given loop or none of its nodes.

In the forthcoming text it is important to have clarity about *Before* and *After* sets containment rules. Let v, v_b, v_a be the vertices such that $v_b \in \mathit{Before}(v)$ and $v_a \in \mathit{After}(v)$. Clearly, $\mathit{Before}(v_b) \subset \mathit{Before}(v)$ and $\mathit{After}(v_a) \subset \mathit{After}(v)$. We also have $v \in \mathit{After}(v_b)$ and $v \in \mathit{Before}(v_a)$. And so, finally, $\mathit{After}(v) \subset \mathit{After}(v_b)$ and $\mathit{Before}(v) \subset \mathit{Before}(v_a)$.

The *Before* and *After* sets have some interesting properties in the context of sound marking, that we will formulate in the following proposition. We say that a node (place or transition) x in a Petri net is reachable from a given marking M if it is not dead in the case x is a transition or it can be marked by some marking reachable from M, in the case x is a place.

Proposition 1 *Let M be a sound marking of WF-net with synchronisations. For each place or transition x we have:*

1. $M(x) \geq 1$ *implies* $M(\mathit{After}(x)) = 0$
2. $M(\mathit{Before}(x)) > 0$ *implies* $M(\mathit{After}(x)) = 0$
3. $M(\mathit{Before}(x)) > 0$ *implies* $M(x) = 0$ *and x is reachable.*

Proof. Let $\langle P, T, F, s, e, S, C\rangle$ be a structured SWF-net and M a sound marking on this network.

We begin the proof with some observations. When constructing the *Before* and *After* sets, we take into account only subtrees related with nodes that are of sequential split type. The sequential split type nodes determine the partial order of transitions in possible transition sequences transforming any sound marking M (in particular M_s^1) into M_e^1. Some transitions are incomparable in this order because of different types of nodes, for example AND nodes that introduce concurrency in nets. This partial order results in important properties of *Before* and *After* sets.

For a transition $t \in T$ that is not a part of any loop, the set $\mathit{After}(t)$ contains all transitions that can fire after t occurs in a sequence transforming a sound marking M into M_e^1 and that this firing is directly dependent on t. If the transition t is in a loop, the same condition holds, except for some other transitions from this loop — not all of them are included in $\mathit{After}(t)$ set. But still, all transitions from the loop that belong to the $\mathit{After}(t)$ set in order to fire, need transition t to fire before them.

It is important to stress out here, that we only consider transition sequences that contain t when writing about firing dependences. In the case of transitions that come after AND nodes, there are at least two independent paths which can lead to those transitions firing, so there are possible situations, when t won't

occur in such sequences but the considered transitions will still fire. Nevertheless, if t occurs in such a sequence, it determines the path that the process went through and we know that t firing is necessary in order to make the next transitions firings possible. We have a similar situation in the case when t results from loop main transition fragmentation (as $t7$ and $t8$ on figure 1).

With this observation, we can move to the main part of the proof.

1. The first step is a direct result of our observation. If $x = e$ then this point is obvious. If $x \neq e$ then $\mathrm{M}(x) \geq 1$ means that x is a place and x is an input for some transitions $T_x \subset T$.

Firstly, let's consider nets that contain no loops. Each transition in any sequence transforming M_s^1 into M can occur only once. Since x isn't empty, we know that none of the T_x transitions will occur in possible transition sequences transforming M_s^1 into M. Taking into account our observation this also means that none of the transitions from *After* sets of T_x transitions will occur in such sequences. It means that none of outputs of T_x transitions or outputs of transitions from their *After* sets is marked. We also know, that all the transitions from T_x, all their outputs and their *After* sets are in $After(x)$. Moreover it is easy to find that these are the only items in $After(x)$. So we have $\mathrm{M}(After(x)) = 0$.

The case of loops is very similar. The only problem is that x can be a loop element. It is possible in this case, that not all of T_x transitions will be in $After(x)$ set — some of them can be transitions starting new loop iteration. However, it is easy to recognize that this makes no problem, and reasoning for transitions from T_x that are in $After(x)$ is still valid.

2. This property is a simple consequence of 1. We know that for all places from $Before(x)$ that marking M is greater than zero and their *After* sets markings is zero too. We also know that $After(x)$ set is a subset of those *After* sets, so $\mathrm{M}(After(n)) = 0$.

3. First part of this property is also a consequence of point 1. We know, that for all places from $Before(x)$ for which the marking M is greater than zero, their *After* sets markings equal zero. We also know that x is an element of those places *After* sets, so $\mathrm{M}(x) = 0$.

The fact that x is reachable is a result of our observation. Let t_y be such a transition that is in *After* set of some place from $Before(x)$ that the marking M is greater than zero, and for which this place in an input. We know that x is in t_y's *After* set, so there exists a transition t_x for which x is an output and either $t_x \in After(t_y)$ or simply $t_x = t_y$. Because marking M is sound, it is possible for t_y to fire. In the case when $t_x \neq t_y$, we know from the observation and from the fact that the marking M is sound [ChBHO03], that it is also possible for t_x to fire, so x is reachable.

4.2 Properties of sound *SWF*-net markings

In this section we'll give a characterisation of *SWF*-net sound markings in the form of a short lemma. Before formulating the lemma, it is worth noticing, that as in the case of standard *WF*-nets, if M is a sound marking of *WF*-net with synchronisations, then M is 1-bounded (there can't be two tokens at a one place).

Lemma 1 *Let* $\mathcal{N} = \langle P, T, F, f, e\rangle$ *be an* WF*-net and* $\mathcal{N}_s = \langle P, T, F, f, e, \{s\}, C\rangle$ *be the same net with added synchronisation place* s*. Given* $\mathcal{N}_s$ *marking* M *is sound if and only if:*

1. $M|_{\mathcal{N}}$ *is sound in* $\mathcal{N}$
2. *Exactly one of following holds:*
   (a) $M(Before(in(s))) > 0$
   (b) s *is marked*
   (c) $M(After(out(s))) > 0$
   (d) *Synchronisation was skipped, so* $M(Before(in(s)) \cup After(in(s))) = 0$ *and* $M(Before(out(s)) \cup After(out(s))) = 0$

Basically, the lemma describes intuitions about how synchronisation place should work. It says, that we have four different states, that our process can be in. First three cases are straightforward: it can be either before synchronisation, during synchronisation or after synchronisation. The last one is the case, when the synchronisation occurs in one of the branches that resulted from an choice split, and an active branch is not the synchronised one. An example of such situation is depicted below on figure 4.

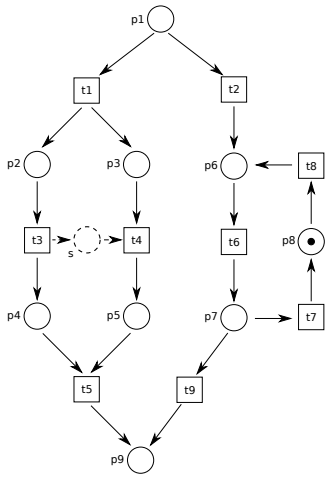

Fig. 4. Example of synchronisation skipping marking

Therefore, for the sake of precision, we can describe the first three states as: synchronisation place is active, process is during synchronisation and synchronisation is inactive, but either synchronisation has occurred, or the process branch without synchronisation is at the evaluation point, when marked places are in $After(out(s))$ set. The last one describes situation when process finished considered fragment with alternative). Nevertheless it is more convenient to use definitions presented in the previous paragraph.

To show that the above intuitions are right, we will now prove our lemma.

Proof. It is important to remember that synchronization of items in loops is not allowed, so in every valid transition sequence $in(s)$ and $out(s)$ can occur only once.

Firstly let us assume that a marking M is sound in $\mathcal{N}_s = \langle P, T, F, f, e, \{s\}, C \rangle$ net.

What synchronisation point does, is prohibiting certain transition sequences that allow obtaining marking $\mathrm{M}|_{\mathcal{N}}$ from M_s^1 in $\mathcal{N}$, by enforcing firings of one transition group before another. It is important that synchronisation is not adding any new possible transition sequences. This means, that any transition sequence that leads to the marking M in $\mathcal{N}_s$ is also valid in $\mathcal{N}$, leading to the marking $\mathrm{M}|_{\mathcal{N}}$. This means that $\mathrm{M}|_{\mathcal{N}}$ is sound in $\mathcal{N}$, so 1. holds.

For the case when there exists such OR place that allows to skip synchronisation, let p_{in} be the place before such fork, and p_{out} first place after it. From the rules of synchronization we know that in this case p_{in} is in $\mathit{Before}(in(s))$ and p_{out} is in $\mathit{After}(out(s))$.

It is easy to see, that each of the conditions a), b) and c) exclude d) and vice versa. Now we'll show that a), b) and c) exclude each other. If $\mathrm{M}(\mathit{Before}(in(s))) > 0$, then we know from the proven proposition, that there is no transition sequence that leads to a marking M containing $in(s)$, so s will be empty for each valid subsequence of those transitions, which means that there was no possibility for $out(s)$ to fire, so again from the proposition, $\mathrm{M}(\mathit{After}(out(s))) = 0$.

If s is marked, than $in(s)$ has already fired. We know that $\mathrm{M}|_{\mathcal{N}}$ is valid so, $\mathit{Before}(in(s))$ marking must be empty. Again from the proposition we have $\mathrm{M}(\mathit{After}(out(s))) = 0$, because there was no possibility for $out(s)$ to fire.

If $\mathrm{M}(\mathit{After}(out(s))) > 0$ then it means that transitions $in(s)$ and $out(s)$ have already fired, so s is empty and from the validity of $\mathrm{M}|_{\mathcal{N}}$, either $\mathrm{M}(\mathit{Before}(in(s)))$ is empty or the process was run following a path where no synchronisation occurs. In the second case we have $\mathrm{M}(p_{out} \cup \mathit{After}(p_{out})) > 0$, we have that $\mathrm{M}(s) = 0$ and $\mathrm{M}(\mathit{Before}(in(s))) = 0$, because $\mathrm{M}(p_{in} \cup \mathit{Before}(p_{in})) = 0$ according to the proposition.

If none of the conditions a), b) or c) holds, we have $\mathrm{M}(\mathit{Before}(in(s))) = 0$, $\mathrm{M}(s) = 0$ and $\mathrm{M}(\mathit{After}(out(s))) = 0$. There are two possibilities. Firstly, we may consider situation when $\mathrm{M}(\mathit{After}(in(s))) > 0$ and $\mathrm{M}(\mathit{Before}(out(s))) > 0$, but in this case s can't be empty, because it would make impossible for $out(s)$ to fire and we know from the proposition that it has to fire in order to clean up tokens from $\mathit{Before}(out(s))$.

The only valid option is $\mathrm{M}(\mathit{After}(in(s))) = 0$ and $\mathrm{M}(\mathit{Before}(out(s))) = 0$, which means that all tokens are in $\mathit{After}(p_{in}) \cap \mathit{Before}(p_{out})$ on process path disjoint with the synchronisation. There are no other options because of the synchronisation rules, which allow to synchronize only these transitions, which are not separated by choice or loop split nodes in the refinement tree. So we know now that a), b), c) and d) exclude each other and in each case one of them must hold, so soundness of $\mathcal{N}_s$ means that 1. and 2. hold.

Now lets assume that 1. and 2. from the lemma hold.

$\mathrm{M}|_{\mathcal{N}}$ is sound, so let $\langle x_i \rangle$ be a sequence of events leading to M_e^1 in $\mathcal{N}$. It's easy to see, that in case when one of b), c) or d) holds, the same sequence can be fired in $\mathcal{N}_s$. In the case when s is not empty, this sequence will clean s. A bit harder is the case when a) holds, but one can easily see, that it is possible to rearrange $\langle x_i \rangle$ so that $in(s)$ will occur before $out(s)$ and still all transitions from $\mathit{Before}(in(s))$ will be before $in(s)$ and all transitions from $\mathit{After}(out(s))$ will be after $out(s)$. Rules of synchronisation guarantee that such operations will lead to valid transition sequence, which will also be valid in $\mathcal{N}_s$ and will lead to the same result as the original sequence.

This proves that if 1. and 2. from the lemma hold for a given marking M that the marking is sound.

We have proved implications in both directions, so the lemma is valid.

5 Soundness Checking

The algorithm for checking the soundness of a marking will be a modification of the soundness checking algorithm (theorem) from presented in previous work [Ch06].

As a reminder, two functions w and W were defined as follows:

$$w(v) = \begin{cases} 1 & \text{if } v = \mathit{root} \\ w(\mathit{parent}(v)) & \text{if } v \text{ is a child of a node which is not a parallel split node} \\ \frac{w'(\mathit{parent}(v))}{c} & \text{if } v \text{ is a child of a node of the parallel split node, with } c \text{ children} \end{cases}$$

$$W(v) = \begin{cases} \mathrm{M}(v)\, w'(v) & \text{for each place-leaf } v \\ \sum_{y \in Ch(v)} W(y) & \text{for all internal nodes } x \\ 0 & \text{for each transition-leaf } t \end{cases}$$

The theorem stated that a marking is sound if and only if for each tree node x either $W^s(w)(x) = 0$ or $W^s(w)(x) = w(s)$. We will now define new functions w^s, W^s and S such that W^s and S will have the same signature as W and w^s will have the same signature as w. Note that the values in W and S are determined only by leaves — for the internal nodes we take sum of the values of their children. For both $in(s)$ and $out(s)$, we will add additional weight to the path from the node to the root, and propagate this change downwards for the pre-nodes (nodes occurring to the left of it). Let c_s be a constant such that for every node x the condition $w(x) > c_s$ is satisfied.

We will apply the following transformation two times to W presented in [Ch06] for both $in(s)$, $out(s)$, with c equal to c_s, $-c_s$ respectively.

For every node v on the path from t to root we add the weight c

$$w'(v) = w(v) + c$$

For every node $v \in Prenode(in(s))$ (nodes on the left of the path from $in(s)$ to root)

$$w^s(v) = \begin{cases} 1 & \text{if } v = root \\ w^s(parent(v)) & \text{if } v \text{ is a child of a node which is not a parallel split node} \\ \frac{w^s(parent(v))}{c} & \text{if } v \text{ is a child of a node being a parallel split node, with } c \text{ children} \end{cases}$$

For the remaining nodes $w^s(v) = w(v)$. The function W remains unchanged, except for the fact that now it depends on w^s.

$$S(M)(t) = \begin{cases} c_s \cdot (\mathrm{M}(s) + \mathrm{M}^?(After(out(s)))) & \text{if } t = in(s) \\ -c_s \cdot (\mathrm{M}^?(After(out(s)))) & \text{if } t = out(s) \\ 0 & \text{for every other case} \end{cases}$$

We will use this transformation two times to nodes $in(s)$, $out(s)$ with c being inverses of each other. Notice that this pair of transformations together changes nodes only inside the smallest subtree containing them.

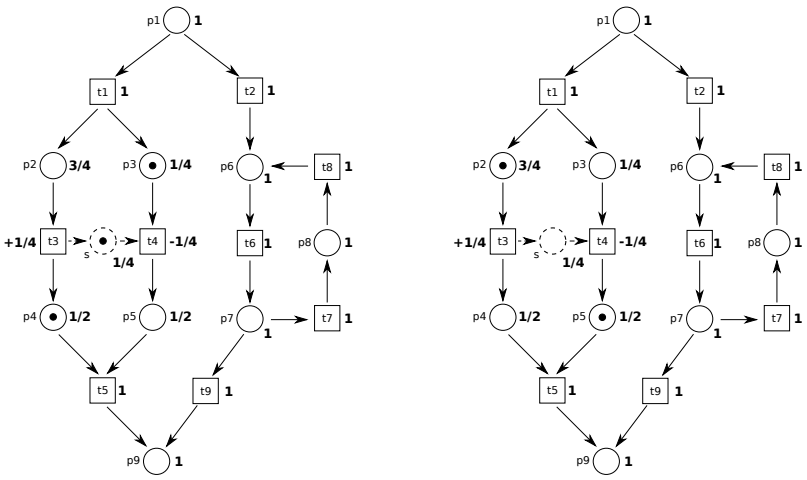

Fig. 5. Example of *SWF*-net with marked weights defined by functions from the theorem. The marking on the left is sound, while the marking on the right isn't, which results in weight explosion in $t1$.

Theorem 1 *A marking M of the* SWF*-net $\mathcal{N}$ is sound if and only if for each node v in the refinement tree either $W^s_M(v) + S(v) = w(v)$ or $W^s_M(v) = 0$*

Proof. We will prove it by using the sound characterisation lemma.

Let M be a sound marking. By the above lemma, exactly one of following holds:

1. $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) > 0$
2. s is marked.
3. $\mathrm{M}\,(\mathit{After}\,(\mathit{out}(s))) > 0$
4. Synchronisation was skipped, so $\mathrm{M}\,(\mathit{After}\,(\mathit{in}(s)) \cup \mathit{Before}\,(\mathit{in}(s))) = 0$ and $\mathrm{M}\,(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}\,(\mathit{out}(s)))$

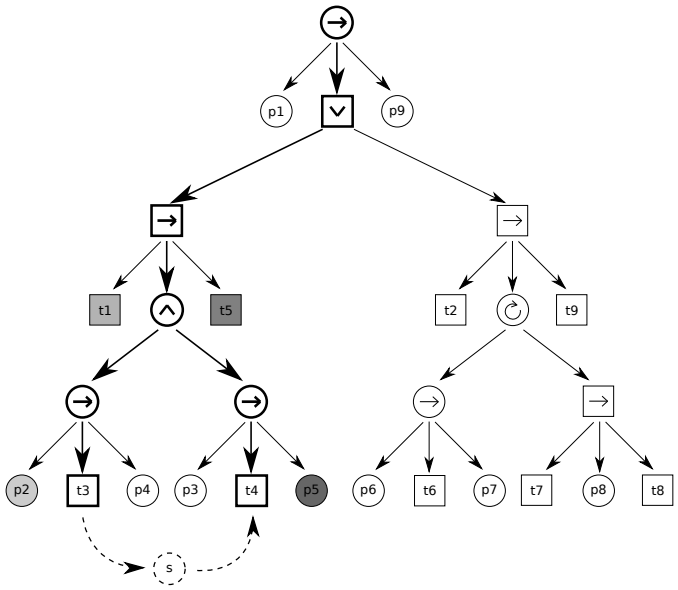

Fig. 6. Derivation Tree with paths highlighted

Lets reason case by case, we will prove that in each of these cases our algorithm will correctly verify the marking:

1. If $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) > 0$: Both $\mathit{in}(s)$ and $\mathit{out}(s)$ are inactive (which means $S(\mathit{in}(s)) = S(\mathit{out}(s)) = 0$). The marking M is sound, so $M(\mathit{After}\,(\mathit{out}(s))) = 0$ and $M(\mathit{After}\,(\mathit{in}(s))) = 0$. This means that all the marked nodes are in modified left sides ("Before") of trees, so the verification behaves just as in the W without synchronisation, because all the weights were just decreased by some amount.

2. If synchronization place s is marked: $\mathit{in}(s)$ is active and $\mathit{out}(s)$ is inactive. Since $\mathrm{M}(\mathit{After}\,(\mathit{in}(s)) \cup \mathit{Before}\,(\mathit{in}(s))) > 0$ and $\mathrm{M}\,(\mathit{Before}\,(\mathit{in}(s))) = 0$, we have $\mathrm{M}\,(\mathit{After}\,(\mathit{in}(s))) > 0$ and because the marking is sound, it meets $S(\mathit{in}(s))$ in some node in the path to root, and then $S(\mathit{in})$ modifies it to the correct amount

3. If $\mathrm{M}(\mathit{After}(\mathit{out}(s))) > 0$: both $\mathit{in}(s)$ and $\mathit{out}(s)$ are active. Of course in this situation $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) > 0$ and $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) = 0$, so we have $\mathrm{M}(\mathit{After}(\mathit{in}(s)))) > 0$ The marking is sound, so it meets the $S(\mathit{in}(s)$ in some node in the path to root, and then $S(\mathit{in}(s))$ modifies it to the correct amount. Similarly, since $\mathrm{M}(\mathit{After}(\mathit{out}(s))) > 0$ and marking is sound, the verification succeeds until it meets at the path from in to root and here it is corrected by S from $\mathit{in}(s)$

4. If synchronisation was skipped, so $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) = 0$ and $\mathrm{M}(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}(\mathit{out}(s)))$: The modifications to weights we made apply only to nodes in the path from $\mathit{in}(s)$ to $\mathit{out}(s)$ and they are all empty in this case.

This proves that in every case, our verification succeeds.

Let us prove the opposite implication now. Let M be such a marking that for each node v in the refinement tree either $W^s_M(v) + S(v) = w(v)$ or $W^s_M(v) = 0$. We will show that the cases from the lemma are exclusive:

1. excludes (2. or 3.) Let's assume $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) > 0$. Then if 2. or 3. then $\mathit{in}(s)$ is active, but since $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) > 0$, there is an active node that is on the left hand side from the path from $\mathit{in}(s)$ to root, but it was already modified by c, so when S will meets, there will be c added two times.

2. and 3. Easy case, $W(\mathit{in}(s)) = w^s(\mathit{in}(s)) \cdot \left(\mathrm{M}(s) + \mathrm{M}^?(\mathit{After}(\mathit{out}(s)))\right) = 2w^s(\mathit{in}(s)) > w^s(\mathit{in}(s))$

It is easy to see that 4. is disjoint from the other cases too. We need only to distinguish 4. from 2.: If $\mathrm{M}(s) > 0$ then $W(\mathit{in}(s)) > 0$, and since 4. holds, $W(\mathit{out}(s)) = 0$, so in the place where paths from $\mathit{in}(s)$ to root and $\mathit{out}(s)$ to root meet there will be an unbalance.

Hence the cases from the lemma are disjoint. We need to prove now that at least one of them holds. Let's assume none of them holds. Then $\mathit{in}(s)$ and $\mathit{out}(s)$ are not active. Assume that $\mathrm{M}(\mathit{Before}(\mathit{out}(s)))$ isn't empty. Because of 1., $\mathrm{M}(\mathit{Before}(\mathit{in}(s))) = 0$, so there are nodes enabled to the right from the path from $\mathit{out}(s)$ to the root node, that are below least common ancestor of $\mathit{in}(s)$ and $\mathit{out}(s)$. When the weight of those active nodes gets passed to the path, they need S to have $W^s_M(v) + S(v) = w(v)$, but $\mathit{in}(s)$ is inactive, so we came to a contradiction.

Similarly, assume $\mathrm{M}(\mathit{After}(\mathit{in}(s))) > 0$. Because of 1., $\mathrm{M}(\mathit{After}(\mathit{out}(s))) = 0$, so there are enabled nodes to the left from path from $\mathit{out}(s)$ to the root node, that are below least common ancestor of $\mathit{in}(s)$ and $\mathit{out}(s)$. When the weight of those enabled nodes get passed to the path, they need S to have $W^s_M(v) + S(v) = w(v)$, but $\mathit{in}(s)$ is disabled, so we came to a contradiction.

These two arguments imply that 4., $\mathrm{M}(\mathit{After}(\mathit{in}(s)) \cup \mathit{Before}(\mathit{in}(s))) = 0$ and $\mathrm{M}(\mathit{Before}(\mathit{out}(s) = 0) \cup \mathit{After}(\mathit{out}(s)))$, so at least one of the cases from the lemma holds.

Now we need to prove the first part of the lemma, that $\mathrm{M}|_{\mathcal{N}}$ is sound in $\mathcal{N}$. The proof of this fact is identical to first part of this proof. We show that if one of 1., 2. or 3. occurs, the rest of checking behaves identical to checking done in w.

6 Conclusion

We have proven that the important construction of creating a channel between two transitions (like links in BPEL4WS) can be done in a semi-structured manner with the preservation of soundness. We have discovered a condition that is sufficient and necessary for a marking to preserve soundness. The condition, based on the structure of the refinement tree is fast to verify; in fact it is linear with respect to the number of nodes of the net (so even better than the size of the net: the edges, with possible quadratic number of them, do not count). This condition allows us to determine soundness of an arbitrary marking and allow on-the fly changes of markings during the execution of a workflow. Such changes are considered a powerful tool for a manager to change a marking in an arbitrary manner in case of an unexpected detour from the normal workflow run. Support by automatic verification, if such changes can cause an undesired behavior (like deadlock or creation of trash tokens) is an important improvement of the technology.

References

[vdAtH00] W.M.P.van der Aalst, A.H.M. ter Hofstede, *Verification of Workflow Task Structures: A Petri-net-based Approach*, Information Systems, 25(1): 43-69, March 2000.

[ChBHO03] Piotr Chrząstowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, Adi Susanto, *Top-down Petri Net Based Approach to Dynamic Workflow Modelling*, Lecture Note in Computer Science. v2678. 336-353., 2003.

[Ch06] P.Chrząstowski-Wachtel, *Determining Sound Markings in Structured Nets*, Fundamenta Informaticae, 72, 2006.

[BPM05] M. Laugna, J. Marklund. *Business Process Modeling, Simulation, and Design.* Prentice Hall, Upper Saddle River, New Jersey, 2005.

[HA00] C.Hagen, G.Alonso, *Exception Handling in Workflow Management Systems*, IEEE Trans. of Soft. Eng. vol. 26 No 10, Oct 2000.

[BPS09] W.M.P. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. *Business Process Simulation: How to get it right?* In J. vom Brocke and M. Rosemann, editors, International Handbook on Business Process Management, Springer-Verlag, Berlin, 2009.

[WS09] A. Rozinat, M. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge., *Workflow Simulation for Operational Decision Support.*, Data and Knowledge Engineering, 68(9):834-850, 2009.