Determining Sound Markings in Structured Nets

Piotr Chrząstowski-Wachtel†
Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warszawa, Poland
pch@mimuw.edu.pl

Abstract. When we model workflows with Petri nets, we call a workflow net sound, if it neither makes any transition dead nor it produces trash tokens in such a way that for every input token exactly one token appears eventually on the output place. We assume that the initial marking consists always of one token on the input place. However, sometimes it is necessary to take into account arbitrary markings, for instance when we make a recovery from an unexpected situation during the workflow execution. An arbitrary marking is sound if it eventually produces exactly one output token without the possibility to leave any trash tokens. The paper addresses the problem of determining the proper control recovery, when unexpected situation arises, and we must detour from the normal execution. When we create an arbitrary control state during the recovery, it is easy to overlook some consequences and create either trash tokens or a deadlock in the future. The problem of determining if a given marking is sound is addressed in the paper in the context of structured nets. The presented linear solution is a necessary and sufficient condition for soundness of markings in structured nets.

Recoveries are caused by unexpected events, sometimes external, which can occur during the execution of a workflow. They were extensively studied in the workflow area (see e.g. [13], [6]). In particular the idea to use a kind of structured nets for the management of rollbacks was addressed in [5]. The recovery problem for workflows is to model them inside the design framework. When a recovery takes place, we are supposed to recover the workflow, and reinitialize the workflow control from some arbitrary state. This usually requires removing all control tokens from the affected area and putting some of them on some specific places.

*This work is supported by internal research grant No.BW/ALG/01/2004 of PJWSTK, financially supported by KBN (State Committee for Scientific Research) in Poland
†Address for correspondence: Institute of Informatics, Warsaw University, Banacha 2, PL 02-097 Warszawa, Poland
We do not restrict ourselves to any specific areas of the regions where we would like to recover. It is not only rolling back and re-doing some actions, but also advancing the workflow execution forward (like in [15]) or making an incomparable entry (in the sense of a partial ordering of progress states), so an entry to the point that has not been yet considered in the normal workflow runs, leading to an entirely new state.

It is quite easy to make an error concerning the recovery distribution of tokens. We consider recovery situations as unpredictable, hence the token distribution can be considered more or less arbitrary. Once it is decided where the control tokens are to be distributed, the natural question arises, whether the distribution is proper in terms of soundness. Can the workflow output place be reached in any of the scenarios and if so, will there be any trash tokens left in the net?

The idea of making the design in a hierarchical way is quite natural. It has been proposed for instance in [10] or [11]. The approach here is different and quite straightforward. It makes use of a kind of a graph grammar which allows us to expand single places or transitions in a structured way.

A Petri net solution to the problem of sound recoveries is addressed in the paper. Petri nets turned out to be quite adequate for solving the control flow presentation and analysis (see for instance [2], [1]). Recoveries cannot be modeled by normal transitions. An extension of reset transitions is proposed in this paper to model the recovery control.

Recovery regions are specified at a design time in a structured manner. The concept was first introduced in [6], but in an unstructured manner, raising some decision problems. A recovery can be resolved by a special transition that takes away all tokens from all the places of the region and puts them on appropriate places, from which we would like to recover the execution. The appropriate definition of regions and recovery transitions is restricted by following some structural conditions that preserve a simple and natural structure of workflow model. Since regions are workflow nets by definition, we can use the general result proven in this paper in the context of regions.

1. Preliminaries

We adopt here the Petri net model proposed in the book [3]. Petri nets are triples $\langle P, T, F \rangle$ where $P$, $T$ are finite and disjoint sets of places and transitions respectively, while $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation. Places are graphically denoted by circles, transitions by boxes and elements of $F$ by arrows. Workflow nets (WF-nets) are Petri nets with two places distinguished: one input place $s$ and one output place $e$, representing the initial and the final state of workflow execution. A marking of a Petri net is a function $M : P \rightarrow N$. For $x \in P \cup T$ let $\cdot x = \{ y \in P \cup T : (y, x) \in F \}$ and $x^* = \{ y \in P \cup T : (x, y) \in F \}$. For $X \subseteq P \cup T : X = \cup x \in X \cdot x, X^* = \cup x \in X x^*$.

We say that a transition $t \in T$ is enabled at a marking $M$ iff $M[p] > 0$ for all $p \in \cdot t$. Transition $t$ enabled at a marking $M$ can fire transforming $M$ into $M'$ such that $M'[p] = M[p] - 1$ for all $p \in \cdot t \setminus \cdot t^*$, $M'[p] = M[p] + 1$ for all $p \in \cdot t^* \setminus \cdot t$ and $M'[p] = M[p]$ for all $p \in P \setminus (\cdot t \cup \cdot t^*)$. If there is a sequence of markings $M = M_0, M_1, \ldots, M_k = M'$ and a sequence of transitions $t_1, \ldots, t_k$ such that for each $j = 1, \ldots, k$, $t_j$ is enabled at $M_{j-1}$ and firing $t_j$ at $M_{j-1}$ transforms $M_{j-1}$ into $M_j$, then we say that $M'$ is reachable from $M$. The set of all markings reachable from $M$ is denoted by $[M]$.

A transition $t$ is dead at a marking $M$ iff none of the markings reachable from $M$ enables $t$. We say that a marking $M$ in is live iff none of the markings reachable from $M$ makes any transition dead. Marking $M$ is bounded iff the set $[M]$ is finite, and is 1-bounded if for all $M' \in [M]$ and all $p \in P :$
$M[p] \leq 1$. A net $N$ is well formed if there exists a live and bounded marking in $N$. For $p \in P$ and $k \in \mathbb{N}$ let us denote by $M_p^k$ the marking which has exactly $k$ tokens on place $p$ and zero tokens on other places.

**Definition 1.1.** A workflow net $N = \langle P, T, F, s, e \rangle$ with $s$ being the input place and $e$ being the output place is sound iff for the initial marking $M_s^1$.

1. no transition is dead
2. $\forall M \in [M_s^1] \exists M' \in [M] : M' \geq M_e^1$
3. $\forall M \in [M_s^1] : M \geq M_e^1 \Rightarrow M = M_s^1$

In other words $N$ is sound if for the initial marking $M_s^1$ there are no dead transitions and from every marking reachable from $M_s^1$ it is possible to reach a marking $M'$, which has exactly one token on the output place, and the only reachable marking having one token on $e$ is the final marking $M_e^1$.

It was proven in [3] that if we connect the output and the input place of a sound net by a transition that moves one token from the output place back to the input place, then well formedness of such augmented net is a good criterion for soundness of the original net. We call such augmented net the closure of the workflow net, and denote it by $N^*$. Formally $N^* = \langle P', T', F' \rangle$ such that

- $P = P'$
- $T' = T \cup \{t^*\}$ (we assume that $t^*$ does not belong to $T$)
- $F = F \cup \{(t^*, s), (e, t^*)\}$

Following the ideas of [3], we call a marking $M$ (not necessarily the initial one) sound, if it satisfies the conditions (2) and (3) from the above definition substituting $M$ for $M_s^1$. Sound markings are also live and bounded in the augmented net $N^*$, hence do not lead to dead transitions. In sound WF-nets liveness and boundedness can be assumed an alternative way for defining soundness of a marking (see the theorem about the equivalence of soundness and well-formedness from the cited book).

The problem of deciding if a given marking in a workflow net is sound is in generally a hard problem, but when we know how the net was constructed, it can be easier to determine which markings are sound. The main result of this paper is an algorithm for determining whether a given marking is sound for nets constructed in the top-down manner described in chapter 3.

### 2. Regions

Regions are supposed to be parts of a WF-net that are themselves WF-nets. If we want to create a region, we choose a place during the refinement process and the subnet which emerges from such place is called a region, according to our demand. The decision, which parts are called regions is supposed to be taken during the design time. We require hence that each region has one input place, one output place and some internal places and transitions which transfer tokens from the input place to the output place. A recovery in the region means, that we remove all tokens from the places that are inside the region, and put tokens on some specified places of this region. Often it can be the input of the region, which means that the
region execution has been re-initialized, but this is not the only choice. It can be also the output place, meaning that we wish to skip the entire region execution. But usually we would like to do something in between: save as much work done as possible. We make an attempt to design alternative entry points to a region, than just its default entries, where the region initiates or ends its activity.

Regions are a part of the net definition, so in order to introduce a recovery transition, one must define a region associated with such transition. To one region many recovery transitions can be assigned.

In Fig.1 a part of the workflow designed in Justwin Technologies is presented. It represents a region corresponding to a part of top-level description of the garment production workflow. Let us focus here on selecting fabric, trims, colours and making a concept sketch. Although these actions can be done in parallel, they must be synchronized to make the sample first, and after the sample is produced, to let the accountants create (calculate) the initial costs at the same time as the designers make sample testing. If the cost turns out to be too high, or sample fails to pass the fit test, then we must come back to the design phase, change the concept, the trims or the fabric. Not necessarily all of them — it is quite possible that the concept can remain the same, and we can play only with the fabric or the trims to cut down on costs.

Figure 1. A region and two recovery transitions
The recovery transitions designed in Fig.1 give us a possibility to focus on typical entry points to the design phase. Two kinds of recoveries are presented here. One — tr2 — resetting the whole region to the initial state, and the other one — tr1 — saving some job. In the second case we decide to recover to the phase, when a fabric has already been selected and tested, and what we need is to redesign the trims as well as the concept sketch. This is accomplished by the recovery transition tr1. Note that this will work no matter how advanced we were in making the sample. The recovery could be triggered by — for instance — the change of fashion and the demand from the market to cancel the previous designs. The fabric can be the same as chosen before, but the trims and sketches must be done again.

Let us agree that some parts of the net are distinguished as regions, and that among transitions there are regular transitions \( T \), and recovery transitions \( R \) associated with regions. As with usual Petri nets we assume that a regular (non-recovery) transition \( t \) is enabled by marking \( M \) iff for all its input places \( p \) we have \( M(p) \geq 1 \). Firing an enabled non-recovery transition \( t \) changes the marking \( M \) in such a way, that for all input places of \( t \) the number of tokens decreases by \( 1 \), while for all output places of \( t \) it increases by \( 1 \).

Let’s assume that \( \pi \) is a set of places defined at a design phase to be a region. At this moment we need no restrictions on \( \pi \), but later in the paper such sets of places will be restricted to the ones that emerge from a single place in the refinement process.

For the recovery transitions we define a different firing rule.

**Definition 2.1.** A recovery transition \( t_r \in R \) having a region \( \pi \) as its input is enabled, iff at least one place in this region has a token. If \( t_r \) is enabled, firing \( t_r \) means removing the tokens from all the places in \( \pi \) and putting tokens on the output places of \( t_r \), as with the normal firing rule.

So recovery transitions are a kind of a combination of the reset transitions and normal ones. The reset transitions, studied for instance in [8], have reset input arcs that clear the input place connected by such arc. The recovery transitions resemble the reset transitions, but they are different. Here we require that a recovery transition is enabled if at least one token is present within the area (the region is marked). The outputs of such recovery transitions are quite standard. With the classical reset transitions having at least one token is not required for enabling them. It should be mentioned, that classical reset transitions are difficult to analyze, because many problems (like reachability, liveness, . . . ) turn out to be undecidable. Within our framework we profit from allowing resetting (great expressive power), without losing the analysis possibilities. Liveness not only is decidable, but it turns out to be decidable in a linear time.

Consider again Fig.1. The transition tr1 is enabled, if there is at least one token in the region. No matter how the tokens are distributed inside the region, after the transition is fired, exactly three output places of this transition will have a token inside this region.

With each such recovery certain predefined compensation procedures can be assigned to facilitate the reaction on a possibly unexpected situation. Let us remark, that a recovery transition can be fired both because of an expected reason (like in the presented example of a garment production net) and because of an unexpected reason, letting the user define a safe recovery scenario.

In the next section we introduce structured nets, which enable a definition of regions in a manner that allows nesting, but not overlapping.
3. Structured nets

In the area of workflows, we often face complex scenarios. Having hundreds of tasks is nothing special. The idea to structure the design and presentation seems to be adequate. Similar problems were encountered in the late sixties of the 20th century, when the paradigm of structural programming took over. Instead of writing a code in a bottom-up, or unstructured manner, the code designers switched eventually to modular programming, achieving great success, and making possible writing programs having hundreds of thousands lines of code and, moreover, being able to understand them. Maybe it is the time to try this approach in the workflow area. There are multiple advantages of such structuring on the horizon. To name a few:

- Even big projects look locally small;
- In the case of any changes in the workflow structure, the impact area can be well limited and defined;
- Workflow structured in a proper way can correspond to the structure of management, hence allowing appropriate design of roles;
- The visualization of workflow parts can be done even on small displays, without losing the whole image;
- The requirement for structuring enforces sometimes better net design (as writing structural programs enforces producing better code).

The subject has been addressed for instance in [4]. Our approach concentrates on building a hierarchical workflow model based on Petri nets — a model used in concurrency theory, which has proven to support the hierarchy concept well.

A top-down stepwise refinement design methodology has been proposed in [7]. The general idea is the following. We start with a single place, and for each node we apply one of the 5 refinement rules, as in Fig.2. There are three refinement rules for places and two for transitions. It has been shown in the cited paper that all nets resulting from such refinement process are sound. The question arises what we can tell about the soundness of other markings than the initial one having just one token on the input place.

A necessary and sufficient condition for soundness of any marking, requiring a linear time algorithm is presented in this paper. We get the answer by investigating the refinement tree, introduced in [7]. Since the workflow model is aimed at the problem of unexpected recoveries, we assume that with each region we associate the procedures that restore the workflow execution within a region, once an exception is being raised. Due to the nested structure of regions we are able to describe different recovery levels. A good region design reflects the management structure and support the decisions to be taken at an appropriate level.

We illustrate the approach using the stepwise refinement method. Let us think about places in Petri net as being not static states, but rather as representing the state of execution of a part of the workflow. We begin with a single place, being the coarsest view of the workflow description: it is just nothing more than creating the root of the refinement tree. Let us start with the basic transformations allowing us to model the sequences of actions, the choice between two or more tasks and the parallel split. The five basic refinement transformations are shown on Fig. 2.
For all the rules displayed in Fig. 2, except the first one we require that they can be applied only if there is at least one input and at least one output arc associated with the node of the net to make the transformation valid. Moreover, we presume that the input and output arcs of the node are copied accordingly (as in the picture).

A refinement tree is the tree that can be obtained from the refinement process by making some changes in the full tree of the refinement process. During the tree construction we would like to forget about the unimportant differences leading to the same result. For instance when we execute a sequence of \( n \) sequential refinements, then no matter in which order the refinements are made, after performing such \( n \) steps we always end up with a sequence of \( 2n + 1 \) nodes (we start with one node and each refinement step we add 2 more nodes). In such case we just put all the \( 2n + 1 \) nodes that were added, as descendants of the common ancestor node. They become children of this node, and preserve the order, reflecting the precedence relation in the sequence.
Similarly, when we perform a sequence of AND-splits of a place, no matter in which order we make them, we obtain the same result. In such case we put all the resulting nodes as the children of the parent node, but this time we forget about their ordering, as it is completely irrelevant with respect to the Petri net semantics.

The sequential and the OR-splits of transitions are performed in an analogous way. If we create \( n \) loops that result from a single place, then the parent node in the refinement tree will have \( n + 1 \) children: the common initial place and the \( n \) looped transitions that were created during this process.

It has been proven in [7] that for each WF-net resulting from the proposed refinement rules, all the refinement trees are isomorphic, so we can talk about the refinement tree, meaning the isomorphism class of such trees. It means that the refinement tree forgets about unimportant differences (like the order in which nodes were refined in the sequence), and captures all the important information about the refinement history that has lead to the creation of the considered WF-net. In the refinement tree all nodes of the Petri nets are in the leaves, and the internal nodes reflect the history of the net composition.

4. Region creation

When we want to define a region we just label a place in the refinement tree, fixing the future subnet resulting from this place to be a region. During the process of further refinement it is required that only the sequential split can be applied to such labeled place immediately after it is labeled as a region. This requirement enforces the existence of exactly one input and one output place, hence making a region a structured WF-net by itself. Regions can be nested, but no nodes can be shared between two non-nested regions. So for two regions either one of them is nested in the other one or they are disjoint. Each of the regions (also the nested ones) can have its own recovery transitions, which can fire independently, but not at the same time. If for instance region \( r_1 \) is a subregion of \( r_2 \) (in the refinement tree \( r_1 \) would be a subtree of \( r_2 \)), then we consider the recovery transitions of \( r_1 \) and \( r_2 \) to be in a conflict — they may not fire simultaneously, but any one of them can fire if both have at least one token inside.

Sometimes a place is simultaneously the output place of one region, and we would like it to be at the same time the input place for another region. We encourage the designers to adopt our policy of not sharing places. If we have two consecutive regions, and the output of one of them is supposed to be the input of the other one, we suggest doing the following. With a place being the common predecessor of these two regions in the refinement tree, we first split the place sequentially, obtaining the sequence \( p_1 \rightarrow t \rightarrow p_2 \), and next label \( p_1 \) and \( p_2 \) as regions. When we refine the places \( p_1 \) and \( p_2 \) to regions, the transition \( t \) becomes a silent transition switching the execution context from the region associated with \( p_1 \) to the region associated with \( p_2 \). Once a token comes to the output place of the first region, it is not yet in the second region unless the transition \( t \) fires finishing the execution of \( p_1 \), and initiating the execution of the region of \( p_2 \).

So now, since regions are structured WF-nets, we concentrate on solving a general problem: given a marking in a structured WF-net, determine if the marking is sound. In such case arbitrary decisions concerning the distribution of the output places of recovery transitions can be verified against possible unsoundness created by the introduction of non-structural arcs. We assume here that the recovery transitions can also be added at runtime by managers, when some dynamically created recovery is introduced.
5. Sound markings in structured nets

We are ready now to present a decision algorithm determining if a given marking in a structured workflow net is sound.

We start the algorithm construction with defining two functions associated with the refinement tree. The first of them is defined in a prefix manner, and depends on the tree structure only, while the second one is defined in the postfix manner and depends also on the marking considered. Throughout the paper $Q$ means the set of rational numbers. For every node $x$ of a tree, by $Ch(x)$ we denote the set of all children of $x$.

**Definition 5.1.** Let $T$ be the refinement tree of a WF net $N$. The weight allocation to the nodes $V$ of $T$ is defined as a function $w : V \rightarrow Q$ in the following way.

$$
\begin{align*}
    w(v) &= 1 & \text{if } v = \text{root} \\
    w(parent(v)) &= w(v) & \text{if } v \text{ is a child of a node which is not an AND split node} \\
    w(parent(v)) &= \frac{1}{|c|}w(Ch(v)) & \text{if } v \text{ is a child of an AND-split node, with } c \text{ children}
\end{align*}
$$

Such weight allocation reflects the token flow count. When we encounter an AND-split with $c$ output places, $c$ tokens are generated from one, so $c$ of them are equivalent to the one before the transition causing such AND-split firing. Hence each of the child tokens should be counted as $1/c$th part of the token before the AND-split. Such situation, when the number of tokens in the execution of a firing sequence varies, is often caught by the information stored in P-invariants. The relative ratio of token weight can be reflected by the values of a P-invariant vector (if such a P-invariant exists at all).

**Definition 5.2.** Consider an arbitrary marking $M$ of a WF-net with the refinement tree $T$. Let $W_M : V \rightarrow Q$ be defined as

$$
W(p) = \begin{cases} 
    M(p)w(p) & \text{for each place-leaf } p \\
    0 & \text{for each transition-leaf } t \\
    \sum_{y \in Ch(x)} W(y) & \text{for all internal nodes } x
\end{cases}
$$

So we start the construction of $W(x)$ values from the leaves. The place leaves have values $w(p)$, whenever there is a token on the corresponding place, while the transition leaves, as well as the place leaves without tokens have value 0. These values propagate up being summed up to their parents. Observe that both functions are easy to compute, and each of them requires a single tree traversal. The size of the tree is proportional to the size of the net, so computing these functions takes linear time. Now we are ready to express the sufficient and necessary condition for soundness in terms of these two functions.

**Theorem 5.1.** A marking $M \neq 0$ in the workflow net $N$ is sound if and only if for each node $v$ in the refinement tree either $W_M(v) = w(v)$ or $W_M(v) = 0$. 

Proof:
The induction is done following the structure of the refinement tree.

The basis. The simplest WF net consists of no transitions and just one place, which is both the input place and the output place at the same time. The refinement tree consists of one node $p$, being the only leaf of the tree. We have $w(p) = 1$ and the only marking $M$, which is sound, is the one containing a token in $p$, hence satisfying $W_M(p) = w(p) = 1$.

The induction step. Assume that the thesis holds for all trees up to $n$ nodes. We make a split of some of the nodes, and show that the thesis holds also for the new tree with new functions $w$ and $W_M$. Let $T$ be the tree of some net $N$ before the split, $T'$ be the tree with $n + 1$ nodes after the split resulting in a net $N'$, and $w$, $w'$ be the respective weight allocations. The last split must have been done just above the leaves level. It means that we have performed this last split for some place or transition which was a leaf $l$ in $T$. The result did not change anything except unfolding $l$ into an internal node $l'$ and some children $c_1', \ldots, c_k'$ for $k \geq 2$ depending on the kind of the split. Observe that $k$ can be bigger than $2$, as there can be more than 2 sibling nodes of the newly created nodes.

If the parent node of $l$ is a sequential split, then after performing our split we immediately shift the nodes up to the level of $l$, adding new nodes on the same level as $l$.

Consider some marking $M'$ in $N'$. If none of the places among $c_1', \ldots, c_k'$ is marked, then for all nodes $v'$ of $T'$ other than $c_1', \ldots, c_k'$ we have $W_M(v') = W_{M'}(v')$, where by $W_{M'}(v')$ we mean the value of the function $W_M$ of the node which corresponds to $v'$ in the tree $T$ (formally it is a node of a different tree). Since no tokens are present inside the refinement area, the marking $M'$ is sound if and only if the marking $M$ is sound — the refinement rules preserve soundness, as it was proven in [7]. According to the induction hypothesis, $M$ is sound if and only if for all nodes $v$ of $T$ either $w(v) = W_M(v)$ or $w(v) = 0$. But $W_{M'}(v') = W_M(c_1') = \cdots = W_M(c_k') = 0$, and because for all other nodes $v'$ of $T'$ the values of the functions $W_M$ and $W_{M'}$ are the same, we get the result immediately from the induction hypothesis.

Assume now that for some new leaf place node $c'$ we have $M'(c') = 1$. Let us distinguish now the cases corresponding to the different kinds of the last split made.

Sequential transition split. In this case a transition has been split into a sequence transition-place-transition. Let $p$ be the middle node of the last split. If $M'(p) = 0$, then the function $W_M$ is the same as the function $W_{M'}$ on all nodes except the ones involved in the last split. So since $W_{M'}(p) = 0$, the result of checking if $w'(v) = W_{M'}(v)$ or $W_M(v)$ is determined by these other places. The marking $M'$ is sound in $N'$ if and only if $M$ is sound in $N$, because the new sequence does not alter the behaviour in other way than just adding new transition firing — this cannot change the soundness of the marking. If, on the other hand, $M'(p) = 1$, then there are two cases.

- All the other sibling nodes of $p$ are unmarked, and then the marking $M'$ is sound if and only if it would be sound with $l$ being the leaf, before it was split to $l'$ and the sibling nodes. In this case the induction hypothesis works, and it is easy to verify that the newly added places and transitions in the sequence do not alter the soundness of the marking.
- At least one more sibling place is marked. In this case the marking $M'$ cannot be sound, because two tokens in a sequence cause inevitably some trash tokens, that will remain there when we reach the final marking with the output place marked. This violates the soundness
definition. To verify this observe that the net before the split with the place $l$ marked can be marked so that the condition of Theorem 5.1 is satisfied. But to create value 1 in this place it is necessary and sufficient to have exactly one token in the unfolded part of the net. So we can freeze one token, and use the other one to produce a sequence ending in the final state. The frozen token would be a trash token violating soundness. The last observation completes all the cases of the sequential-transition split.

**Sequential place split.** The arguments are analogous to the previous case.

**OR-split** In case of the OR-split we have several transitions added, hence none of them can have any token, and the function $W_M, W'_M, w, s'$ are the same for each node except the new transitions created. It is easy to verify that introducing the OR-split transition adds up just one more alternative way to pass a token that corresponds to the possibility that was offered by the transition before the split. So soundness and unsoundness are preserved while the considered functions on all other places are equal. So the whole condition is preserved too after the split.

**AND-split** This time we have another situation. The resulting AND-split nodes are weighted with a fraction of the weight of the node $l$. The intuition explaining, why we are doing this that way is that when we perform an AND-split, the parallel threads are later merged. So a token at the entrance to such split must be equalized by $c$ tokens on each of the $c$ threads. None of them can be missing and exactly one of them must be present at each thread. We obtain this by verifying that each thread has exactly $\frac{1}{c}$ weighted tokens in total. The weight $w(l)$ of the internal node $l$ stands unchanged, and we have to distribute the $W$ values among the split places in such a way, that they sum up to $w(l)$. Before the split each of the places was assigned the value $\frac{1}{c}w(l)$. After the split one of the places will make two of them, hence the total number of the children of $l'$ is now $c + 1$ with the value for each $p' \in Ch(l') : w(p') = \frac{1}{c+1}w(l')$. Now, again consider 2 cases:

- There are no tokens on any of the children of $l$. In this case it is easy to verify that if the thesis holds for $\mathcal{T}$ then it holds also for $\mathcal{T}'$, since the value $W_M(l')$ is 0.
- At least one of the children of $l$ contains a token. In this case the marking is sound if and only if all the children of $l'$ contain a token — otherwise the place missing a token would block the synchronization that follows such split, causing a deadlock. If all children $p'$ of $l'$ have tokens, then the values $W'_M(p') = \frac{1}{c+1}w(l')$ sum up to the desired value $W_M(l') = W_M(l) = w(l) = w(l')$, hence by induction hypothesis, the marking $M'$ is sound. Observe that the only way to obtain the desired $w(p')$ value is to assign one token for each of the children of $l'$: other ideas like assigning two tokens for one of them and none for another would cause the inequality of $w$ and $W_M$ values for each of these nodes.

**Loop-split** It is probably the simplest case, as the loop split does not create any place, so the argumentation for preserving both soundness and the function equality is analogous to the OR-split case.

This case completes the whole proof. □

In fact, the proposed characterization of sound markings is general, and can be applied to any marking in a structured net. Anyway, when we use this theorem to regions, where the recovery transitions are
defined, the scope can be limited to the node from which the region has been created. In the tool that supports making dynamic changes it can be detected if the proposed recovery is sound or not almost immediately.

To understand the above theorem better, we could also look at it as a matching of two tree traversals. One, being a prefix-order, assigns to each node a fraction of the initial token that is dependent on the net structure only. On the other hand, the other postfix-order traversal collects the actual information from the marking in the leaves and returns true if every internal node in the refinement tree has as many weighted tokens in its subtree, as it should.

The result can serve in general for deciding soundness of any marking in a structured workflow net. We present it in the context of recoveries, because — to our belief — when a net is constructed in a structured way, it should be naturally marked with one token on the input place. Regions are structured too and each of them is a valid workflow net itself. Recoveries are the situations in which we must perform an action which is not structural, but having such general result, we can deal also with arbitrary recoveries on different levels (regions can be nested).

6. An example of application

Let us concentrate on the example from Fig.1. One can see on Fig.3 the refinement tree of the considered net. To facilitate reading the picture, the internal nodes were marked by symbols that resemble the role of splits: here we have only sequential place splits and AND-splits.

If we decide to put the tokens on places $p_2, p_3, p_4$ (as the recovery transition $tr_1$ does), we can easily check that this distribution of tokens is compliant with the weight allocation, and the condition from Theorem 5.1 is satisfied. The nodes $p_1$ and $p_2$ carry the value $1/4$. These values transfer one level up, and sum up to $1/2$ in their latest common ancestor two levels up. The value $1/2$ is next transferred yet one level up. Summed up with another $1/2$ coming from the parent of $p_3$ gives the value $1$, as it is expected by their common parent just one level below the root. All the other nodes have value 0, so no one objects, and the marking turns out to be sound.

If, instead, for instance, not being very careful, we put a token on $p_1$ instead of $p_3$, then the common parent of $p_1$ and $p_4$ would object. It expects the total of a quarter of a token, and the two quarters provided by $p_1$ and $p_4$ sum up to one half. Hence the marking $(p_1, p_3, p_4)$ is not sound, which we can easily verify in the net.

Similarly if we were trying to put any token on any of the places $p_1, \ldots, p_6$, while we would mark any of the places $p_8, \ldots, p_{11}$, then there is no chance to obtain a sound marking. One and only one of the 7 child nodes of the root should carry a non-zero value (it must in fact be 1), if we want the marking to be sound. For the same reason it is clear that if we put a token on the input or the output place, then any other token would demolish soundness.

7. Future work

The 5 refinement rules we are investigating form a backbone for the construction of the workflow control. In practice they require some extensions to model real-life situations. For instance with these rules it is impossible to model a communication between concurrent threads or a synchronization of concurrent
events. In [7] some non-structural rules on top of the canonical refinement rules and a condition guaranteeing soundness were proposed. It is not difficult to extend the result from this paper also to the nets with communication and synchronization mechanisms. Together with some other rules introduced it will be the subject of another paper.

8. Conclusions

We are proposing a framework for sound designs. Instead of drawing schemes, and ask questions if they enjoy good properties, we rather encourage the designer to construct a sound net directly. The main target is to prevent the designer from making an unsound net. The structural refinement makes possible not only the proper design of the workflow skeleton, but also gives a basis for fast verification if a given
marking is sound. Soundness in general WF-nets is hard to decide (EXP-space hard). Bringing it down to the linear decision algorithm gives us hope that even the analysis of such powerful, complex and dangerous constructs like the recovery transitions can be restrained to acceptable complexity.

There are issues left aside, like the data management and data consistency, which should be addressed separately (see eg.[9]).

9. Acknowledgements

I acknowledge Piotr Góra for discovering an error in the formulation of the main theorem in an earlier version of this paper, as well as valuable comments of two anonymous referees.

References
