

Workflow Soundness and Data Abstraction: Some negative results and some open issues

Nikola Trčka

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{n.trcka}@tue.nl

Abstract. Business analysts often design their workflows in a conceptual manner and produce abstract models. These models are, on the one hand, thoroughly used for analysis; on the other hand, they are extended with full details and executed by a workflow management system. In this paper we introduce two commonly used (and Petri net based) types of workflow abstraction, and we question their reliability when it comes to verification of functional properties. In particular we focus on the soundness property, i.e. on the requirement that every workflow must always be able to complete.

1 Introduction

Business analysts often design their workflows in a conceptual manner, thus producing abstract models. If the purpose of modeling is documentation, communication or analysis, the workflow typically stays at the abstract level. If the purpose, however, is to execute the underlying business process by a workflow management system, the workflow is instantiated with full details, a task typically done by business programmers and not the analysts themselves.

In the most common modeling approach only the control-flow of business tasks is fully included. Other aspects, like data-flow and resources, are either completely excluded or included in some limited/conceptual form only. The usual way of incorporating data information into a workflow, e.g., is to abstract from the actual data values and from the details of data operations, but to simply assume that a data element can be read (both as a task input or as a basis for a routing-decision) or written. The resource information, on the other hand, is typically used only for performance analysis; for other types of analyses it is less relevant as resources are external and dynamic in nature.

In this paper we question the reliability of checking workflows for functional correctness in the abstract setting where resources are excluded and data is either also excluded or included in the above explained conceptual form. We only focus on the most popular correctness criterion, i.e. on the notion of (weak) soundness [1, 2]. Soundness is defined as the requirement that the workflow must have an option to complete from any state it can be in.

We first show that complete data abstraction is undesired as it can lead to both false positive and false negative results. We then extend the notion of soundness to also take the (conceptual) data information into account, and show that wrong results can still be obtained. The most surprising result, however, is that, although we incorporated more knowledge into the soundness check, there are many situations in which the complete data abstraction would give a correct result while our method would produce an incorrect one. The paper poses a challenge to find an optimal data abstraction method for workflows that would (at least) preserve the property of soundness.

2 Workflow nets with data

We define two abstract models for workflows: *workflow nets* that only support control-flow, and *workflow nets with data* that also include data read/write operations and guards. We assume the reader to be familiar with standard Petri net notation.

Workflow nets [1] impose syntactic restrictions on Petri nets to comply to the workflow concept. The notion was triggered by the assumption that a typical workflow has a well-defined starting point and a well-defined ending point, and that it has no dangling tasks.

Definition 1. *A Petri net $N = \langle P, T, F \rangle$ is a Workflow net (WF-net) if it has a single place **start** with $\bullet\text{start} = \emptyset$ and a single place **end** with $\text{end}\bullet = \emptyset$, and every node (place or transition) is on a path from **start** to **end** (i.e. if $(\text{start}, n) \in F^*$ and $(n, \text{end}) \in F^*$ for all $n \in P \cup T$, where F^* is the reflexive-transitive closure of F).*

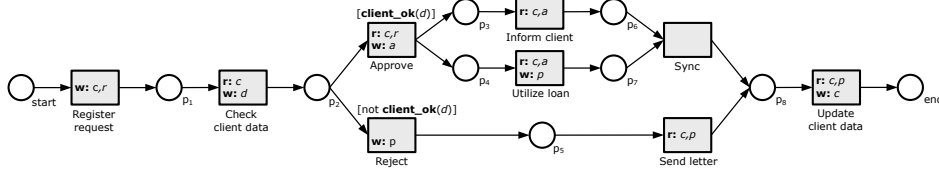
A workflow net with data [3] is a workflow net in which tasks can read from or write to some data element. A task can also have a (data dependent) guard that can block its execution. We formalize the concept of a (conceptual) guard first.

Definition 2. *Let \mathcal{D} be a set of data elements. A predicate (on $d_1, \dots, d_n \in \mathcal{D}$) is an expression $\text{pred}(d_1, \dots, d_n)$ that evaluates to true or false. A guard is either a predicate or the negation of a predicate. The set of all guards over \mathcal{D} is denoted $G_{\mathcal{D}}$.*

We now define workflow nets with data.

Definition 3. *A tuple $\langle P, T, F, \mathcal{D}, \mathbf{r}, \mathbf{w}, \mathbf{grd} \rangle$ is a Workflow net with data (a WFD-net) iff $\langle P, T, F \rangle$ is a WF-net, \mathcal{D} is a set of data elements, $\mathbf{r} : T \rightarrow 2^{\mathcal{D}}$ is the reading data labeling function, $\mathbf{w} : T \rightarrow 2^{\mathcal{D}}$ is the writing data labeling function, and $\mathbf{grd} : T \rightarrow G_{\mathcal{D}}$ is the guarding function, assigning guards to some transitions.*

Fig. 1 shows a WFD-net representing a realistic, albeit simplified, loan approval process. The first task in the process is to register the request of a client.



c : client information, r : loan request, d : decision, a : approval document, p : final report

Fig. 1. A WFD-net representing a loan approval process

This task creates two data elements: c to store clients information and r for the actual request. In the next task the client's history is checked and a decision d is made. The guard $\text{client_ok}(d)$ evaluates to true if d denotes a positive decision; otherwise it evaluates to false. In the positive case the task **Approve** executes, producing a document (a) describing the approved loan (amount, conditions, etc.). This document is then communicated to the client (task **Inform client**). The actual payment is done in parallel via the **Utilize loan** task that also produces the final report (stored in p). The two parallel branches are synchronized by the task **Sync**. In the case of a negative decision, a report is made (the same data element p is used) and sent to the client. Finally, regardless of the decision, the client's record is updated, based on the information given in the report. We implicitly assume that inside a task reading always precedes writing, so when the last task is executed the old version of c is overwritten and thus lost.

3 Soundness property

Soundness property [1, 2] was defined as the minimum requirement that every (business) workflow net must meet. The property guarantees the absence of livelocks, deadlocks, and other anomalies that can be detected without domain knowledge. A WF-net is considered sound if it *can* always (at any time) complete in full (with no references to it remaining). In Petri net terms, from every reachable marking the final marking (one token in the end place and all other places empty) can be reached. The requirement also can be expressed as the CTL formula $\text{AGEF}\tau$, where τ denotes the final marking. Note that soundness is thus weaker than the notion of mandatory completion ($\text{AF}\tau$), even when the latter assumes strong fairness.

Soundness is extended to the WFD-net model by means of a suitable transformation to classical WF-nets that strictly respects the intuitive semantics of WFD-nets. We explain this transformation now.

To capture the restrictions on the behavior due to guards, we introduce a "guard layer": For every *predicate* pred appearing in some guard we introduce places $\text{pred}_{\text{true}}$ and $\text{pred}_{\text{false}}$. A token on $\text{pred}_{\text{true}}$ indicates that the predicate evaluates to true, while a token on $\text{pred}_{\text{false}}$ means that pred is false.

A *change* in the value of a data element that appears in some predicate may potentially change the evaluation of this predicate. We reflect that by assuming

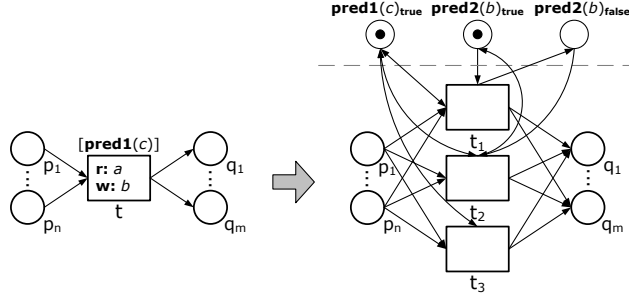


Fig. 2. Decomposition of a task in a WFD-net

that every transition t writing to d might change the value of $\text{pred}(d)$ (or not). Therefore, we split t into three transitions: two to represent possible changes of the predicate value (from true to false and from false to true), and one for leaving the predicate value unchanged. Note that, in case the transition changes data items related to k predicates, it will be in general split into 3^k transitions. An example of a task decomposition is illustrated in Figure 2.

4 False positives and false negatives

In this section we show that neither the WF nor the WFD abstraction is correct, in the sense that they both can give false negatives and false positives.

It is not hard to see that the WFD-net from Fig. 3a is sound, and that any executable workflow of which this WFD is an abstraction is sound as well. However, in this perfectly realistic example, without the data information the sequence t_1, t_2, t_5 that leads to a deadlock would become possible (the information that $\text{pred}(d)$ has the same value in both parallel branches is lost).

Consider now the WFD net in Fig. 3b. This WFD is clearly not sound, and neither is any of its concrete implementations. If t_1 sets the value of pred to true, respectively to false, the upper, respectively the lower, parallel branch would enter a livelock. Without including the data information this workflow would be reported as sound because there would be no indication that actually the same predicate is used in both branches.

Incorporating the data information from a WFD net into the soundness check is, however, also incomplete and can still give false positives and false negatives.

Figure 4a shows a WFD-net that is not sound because it allows for the situation when t_1 sets pred to true and pred_1 to false, which leads to a deadlock. If, however, in some concrete execution t_1 sets the two predicates both to false or both to true, then there is actually no deadlock. The problem appears because the WFD abstraction does not capture the dependencies between the guards.

Similarly, the WFD-net from Figure 4b is reported sound. However, if t_2 we, e.g., take $x := 0$ for t_2 and $x > 0$ for pred , then the workflow has a livelock. The problem is that in the WFD-net the information that t_2 can always make the

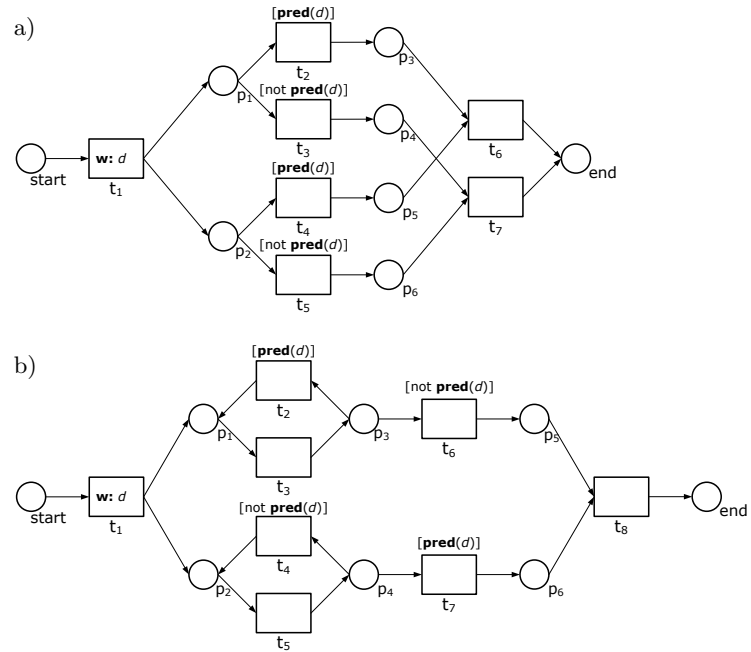


Fig. 3. Complete data abstraction can give both a) false negatives and b) false positives

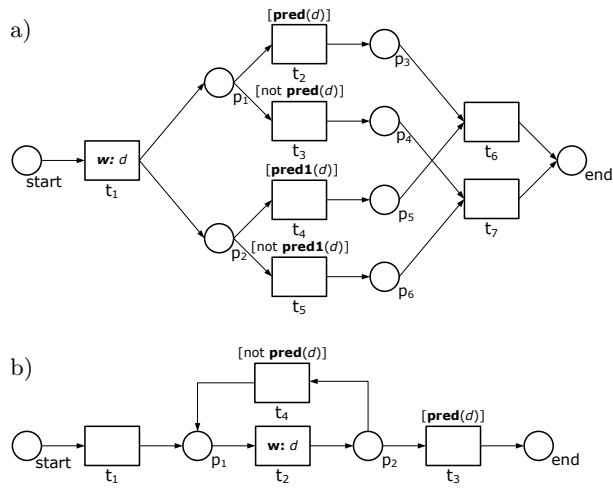


Fig. 4. WFD abstraction can still give a) false negatives and b) false positives

predicate `pred` false is not present. We could, of course, add such possibilities to the WFD decomposition, but then almost every “while-do” loop would become erroneous.

False positives and false negatives suggest that WFD might also not be a suitable abstraction for executable workflows as it does not allow for a reliable soundness verification. Even worse, in some cases a simple WF-net abstraction (i.e., with no data information at all) would do a better job. For example, if we merge the start place of the WFD-net from Figure 4b with the end place of the WFD-net from Figure 3a, we would get a WFD-net that is reported sound. However, if we instantiate this WFD-net by putting $x := 0$ for t_2 and $x > 0$ for `pred`, we would get a non-sound executable workflow. Note that, in the case without data, the non-soundness would be correctly reported, but the reason would be the deadlock in the first part of the workflow, and not the correct one, namely the livelock in the second part. Similarly, consider the WFD-net from Figure 5. This net is not sound (in the WFD sense) as it has a livelock when t_1 sets the value of `pred` to false. However, in the concrete implementation where t_2 stands for $x := 0$ and `pred` stands for $x \geq 0$, this (now executable) workflow becomes sound. Note that soundness would correctly be reported if all data is removed; the “reason” for this is the constant possibility of leaving the loop.

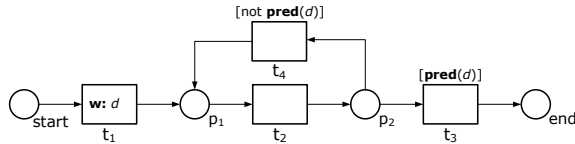


Fig. 5. Simple WF-net abstraction can correctly reports soundness

5 Conclusions

With a series of counterexamples the paper showed that neither WF-nets nor WFD-nets were suitable abstractions for executable workflows as the property of soundness is neither preserved nor reflected. It is believed that this is due to the nature of the soundness property including both the universal and existential modalities.

We have shown that the WFD abstraction works no better than the more permissive WF abstraction. It is, however, the case that every “bad” scenario in the executable workflow that is captured by the WF-net representation would, of course, also be captured in the corresponding WFD-net.

While false negative results could be expected and tolerated, a positive answer on an abstract level should guarantee a positive answer on the concrete workflow. It is an open problem to find an abstraction method that satisfies this, or to characterize the cases where the problem does not appear. In addition, as WFD-nets are sometimes just configurable skeletons for a number of

executable workflows, it would be interesting to look for cases where soundness is preserved/rejected in all possible refinements.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.
3. N. Trčka, W.M.P. van der Aalst, and N. Sidorova. Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In *21st International Conference on Advanced Information Systems (CAiSE'09)*, 2009. LNCS. To appear.