On the verification of EPCs using T-invariants

On the verification of EPCs using T-invariants

H.M.W. Verbeek and W.M.P. van der Aalst

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{h.m.w.verbeek,w.m.p.v.d.aalst}@tm.tue.nl

Abstract. To verify a (business) process model, for example expressed in terms of an *Event-driven Process Chain* (EPC), most of the approaches described in literature require the construction of its state space. Unfortunately, for complex business processes the state space can be extremely large (if at all finite) and, as a result, constructing the state space may require excessive time. Moreover, semi-formal modeling languages such as the EPC language require a rather lenient interpretation of their semantics. To circumvent both the state-explosion problem and the semantics-related problems of EPCs, we propose an alternative approach based on *transition invariants* (T-invariants). T-invariants are well-known in the Petri-net community. They do not require the construction of the state space and can be computed efficiently. Moreover, we will show that our interpretation of T-invariants in this context can be used to deal effectively with the semantics-related problems of EPCs. To demonstrate our approach we will use two case studies: one is based on the reference model of SAP R/3 while the other one is based on a trade execution process within a large Dutch bank. We will also argue that the approach can be applied to other (informal or formal) modeling techniques.

1 Introduction

Today, a lot of verification techniques for (business) process models depend on the existence and/or construction of the state space of that model. However, such a state space can be hard to compute, if at all possible. As a result, researchers try to find ways to keep the computation time of state spaces within reasonable bounds [6, 13, 21]. Nevertheless, the problem of extremely large state spaces persists.

In this paper, we take an altogether different approach. Instead of using the state space, we try to use an approximate property that does not require the state space to be computed. The approximate property we explore is the well-known *transition invariant* (T-invariant) [7, 18, 19]. This will enable us to construct a superset of the set of possible good execution paths of a process model. Based on this set, we can conclude whether certain parts of the process model are not covered by these paths. If such parts exists, then the process model definitely contains errors.

T-invariants are well-known and well-defined in the Petri-net community (although the use of their counterparts, P-invariants, seems more widely spread)

[7, 18, 19]. For this reason, we first map a process model onto a Petri net, then we construct the T-invariants, and last we translate the results back to the original process model. An important observation is that our mapping from the process model onto a Petri net needs only to cover the possible execution paths. As a result, some typical semantics-related problems (like the one addressed in [3]) are avoided by our approach.

In this paper, we use *Event-driven Process Chains* (EPCs) [20] as a modeling language. EPCs are widely used (for example as reference models in the context of SAP R/3 [15]) and supported by modeling tools such as ARIS. Verification of EPCs is known to be hard [2, 3, 16] as they contain so-called OR-join constructs. As our approach avoids these OR-join related problems, we expect that our approximate approach will be able to deal with EPCs easily. Nevertheless, we would like to stress that our approach is approximate: Reports will be correct, but not necessarily complete.

The remainder of this paper is organized as follows. Section 2 introduces the formal Petri-net related definitions used in this paper. Section 3 introduces EPCs and presents our mapping to Petri nets. Section 4 presents two case studies: one which is taken from the ARIS for MySAP reference models, and one is a trade execution process from a Dutch bank. Section 5 relates our approach to other approaches described in literature. Section 6 concludes the paper.

2 Preliminaries

This section introduces the formal Petri-net related definitions used in this paper. First of all, we introduce WF-nets [1], a subclass of Petri nets which we use to capture the essential part of the process. Next, we introduce the known concepts of soundness [1] and relaxed soundness [9] on WF-nets. Both these concepts are used to verify processes. A process is called sound if it can always complete properly no matter what, and it is called relaxed sound if all parts of the process can be involved in proper completion. However, both these concepts rely on the ability to generate the entire state space of the process. If this state space is too complex to be generated within reasonable time, soundness and relaxed soundness might remain inconclusive. For this reason, we also introduce a new approach based on the well-known T-invariants. As we will show later on, this approach comes very close to relaxed soundness, but it does not rely on the construction of the state space.

2.1 WF-nets

Basically, a WF-net is a Petri net which has one source place, usually denoted i, and one sink place, o, such that all nodes are covered by the directed paths from i to o.

Definition 1. *net*
A (Petri) net N is a tuple (P, T, F_i, F_o), where:

– P *is a set of places,*
– T *is a set of transitions such that* $P \cap T = \emptyset$*,*
– $F_i \in T \to I\!P(P)$ *maps every transition onto a set of input places, and*
– $F_o \in T \to I\!P(P)$ *maps every transition onto a set of output places.*

Usually, $F_i(t)$ is denoted $\bullet t$, and $F_o(t)$ is denoted $t\bullet$. Furthermore, we extend these notations to places:

– $\bullet p = \{t \in T | p \in t\bullet\}$
– $p\bullet = \{t \in T | p \in \bullet t\}$

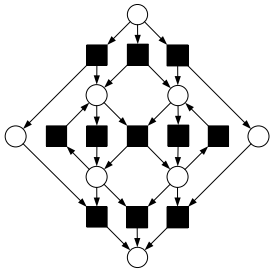


Fig. 1. An example net

Figure 1 shows an example net. As usual, transitions are visualized using rectangles and places are visualized using circles.

Definition 2. *WF-net*
A WF-net [1] is a net (P, T, F_i, F_o) *such that:*

One source place *There is exactly one place* $i \in P$ *such that* $\bullet i = \emptyset$*.*
One sink place *There is exactly one place* $o \in P$ *such that* $o\bullet = \emptyset$*.*
Directed path *Every node* $n \in P \cup T$ *is on some directed path from* i *to* o*.*

The example net shown in Figure 1 is a WF-net: the topmost place is its only source place, the bottommost place its only sink place, and every node is on some directed path from the topmost place to the bottommost place.

2.2 Soundness and relaxed soundness

In the context of workflow, place i is the entry point for new cases, while place o is the exit point. Furthermore, ideally, every case that enters the WF-net (by adding a token to place i) should exit it exactly once (by removing a token from place o) while leaving no references to that case behind in the WF-net (no tokens should be left behind). Furthermore, every part of the process should be viable, that is, every transition in the corresponding WF-net should be executable. Together, these requirements constitute the soundness property [1].

Definition 3. *Soundness*
Let net $N = (P, T, F_i, F_o)$ be a WF-net with source place i and sink place o. Furthermore, let $[p]$ denote the state with exactly one token in place p (and no tokens in all other places). Net N is said to be sound [1] iff*:*

- *From every state reachable from $[i]$, the state $[o]$ is reachable (completion is always possible).*
- *If in some state s reachable from $[i]$ the place o is marked, then $s = [o]$ (completion is always proper).*
- *No transition is dead.*

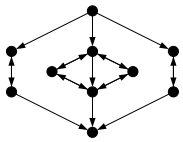


Fig. 2. The state space of the example net

Figure 2 shows the state space of the example WF-net shown in Figure 1. The topmost state corresponds to the state $[i]$, whereas the bottommost state corresponds to the state $[o]$. From this state space, we can conclude that the example WF-net is sound: (1) from every state reachable from $[i]$, there exists a path to $[o]$, (2) $[o]$ is the only reachable state marking place o, and (3) all transitions are present in Figure 2.[1]

Some verification techniques require the addition of an extra transition $*$ such that $\bullet * = o$ and $* \bullet = i$ to a WF-net N. We use $*N$ to denote this *short-circuited* WF-net. Figure 3 shows the short-circuited example net. Note that short-circuited is not a WF-net. The short-circuited WF-net can be used to express soundness in terms of well-known Petri-net properties: A WF-net is sound if and only if its short-circuited net is live and bounded [1]. Recall that liveness and boundedness are two well-known properties in the Petri net world supported by a variety of analysis tools and techniques [18, 19].

In some circumstances, the soundness property is too restrictive. Usually, a designer of a process knows that certain situations will not occur. As a result, certain execution paths in the corresponding WF-net should be considered impossible. Thus, certain reachable states should be considered unreachable. Note that in the verification process we are often forced to abstract from data, applications, and human behavior. Note that it is typically impossible to model the behavior of humans and applications. However, by abstracting from these aspects typically more execution paths become possible in the model. In her

[1] Note that the transition and place labels have been omitted throughout the paper since the mappings are obvious and explicit labels would only distract from the core ideas.

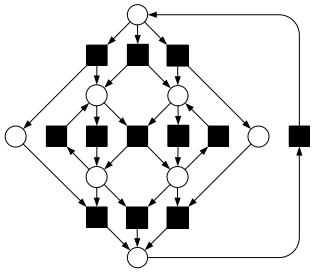


Fig. 3. The short-circuited example net

thesis [9], Juliane Dehnert introduced the notion of relaxed soundness to cope with this phenomenon. A WF-net is called relaxed sound if every transition can contribute to proper completion.

Definition 4. *Relaxed soundness*
Let net $N = (P, T, F_i, F_o)$ be a WF-net with source place i and sink place o. A transition $t \in T$ is said to be relaxed sound [9] iff *there exists an execution sequence $\sigma = t_1 t_2 \dots t_N$ such that:*

- *transition t is included, that is, $t = t_i$ for some i, and*
- *the net effect of σ is moving the token from place i to place o.*

Net N is said to be relaxed sound if all transition $t \in T$ are relaxed sound.

As mentioned before, every case that enters a WF-net should exit it exactly once while leaving no references to that case behind in the WF-net (no tokens should be left behind). Thus, the ultimate goal of a WF-net is to move from place i to place o. The notion of relaxed soundness brings this goal down to the level of transitions: every transition should aid in moving a token from place i to place o. A transition that cannot aid in moving a token from place i to place o cannot help the WF-net in achieving its goal. Hence, such a transition has to be erroneous.

Figure 4 visualizes an execution path in the example net: First transition 1 is executed, then transition 2, and so on. It is straightforward to check that in the example net all transitions are covered by such execution paths.

2.3 T-invariants

An interesting observation[2] now is that an execution path that moves a token from place i to place o corresponds to a cyclic execution path in the short-circuited net: By executing the short-circuiting transition once, the token is back

[2] The same observation has also been used in, for example, [22, 24] to reduce computation time for deciding life-cycle inheritance.

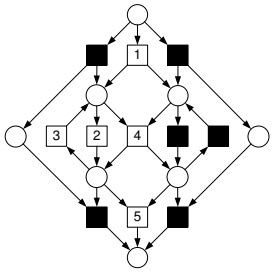


Fig. 4. An execution path for the example WF-net

in place i. It is well-known that a cyclic execution path corresponds to a semi-positive transition invariant. A semi-positive transition invariant (or T-invariant for short) is a bag (multi set) of transitions such that the accumulated sets of input places equals the accumulated sets of output places (where accumulation yields bags, not sets). As a result, the net effect of executing every transition from the bag exactly once is zero.

Definition 5. *T-invariant*
Let net $N = (P, T, F_i, F_o)$ be a net and let $w \in T \to \mathbb{N}$. Function w is a T-invariant of net N if and only if for all $p \in P : \sum_{t \in {}^\bullet p} w(t) = \sum_{t \in p^\bullet} w(t)$.

By definition, every relaxed sound transition is covered by some path from the initial marking $[i]$ to the final marking $[o]$. As a result, every relaxed sound transition is covered by some T-invariant in the *short-circuited* net. However, this does not work the other way around. T-invariants abstract from the state of the net. Therefore, it might be possible that the bag of transitions covered by some T-invariant cannot be executed (because some tokens are lacking). As a result, there may be a transition that is covered by some T-invariant in the short-circuited net, but that is not covered by any execution path from state $[i]$ to state $[o]$. Figure 5 visualizes a T-invariant for the short-circuited example

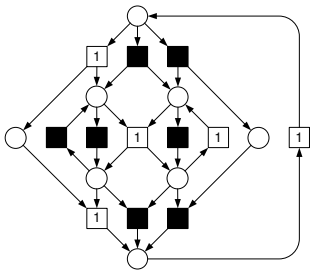


Fig. 5. A T-invariant for the short-circuited example WF-net

WF-net which does not correspond to an execution path, where the numbers indicate transition weights and black transitions have weight zero. Note that the execution path would simply block on the transition in the middle.

Thus, if we are unable to generate the state space in reasonable time, then we can use T-invariants as an approximation. Note that for every execution path from state $[i]$ to state $[o]$ the short-circuiting transition needs only to be executed once to obtain a cyclic execution path. Furthermore, note that there may be cyclic execution paths present in the WF-net itself. For these two reasons, we restrict ourselves to T-invariants where the short-circuiting transition has either weight 0 (corresponds to a cycle in the WF-net itself) or 1 (corresponds to an execution path from $[i]$ to $[o]$).

For constructing a set of minimal (semi-positive) T-invariants, we will use the generic algorithms as described by Colom and Silva [7]. In the worst case, these algorithms are exponential space in the number of transitions, whereas the algorithm to construct a coverability graph is non-primitive recursive space. Thus, constructing a set of T-invariants has a better complexity than constructing a coverability graph. Nevertheless, it might be possible to improve the complexity even further, as we do not need a complete set of minimal T-invariants: We only require a subset of minimal T-invariants that *cover* all transitions that are covered by some minimal T-invariant. However, we did not investigate this yet, as we assume that our approach using T-invariants would already outperform an approach based on a coverability graph.

2.4 Concluding remarks

In this section we have introduced the concepts of WF-nets, soundness, relaxed soundness, and T-invariants. Furthermore, we have shown that T-invariants can be used as an approximation for relaxed soundness. In the remainder of this paper, we investigate how good this approximation works using an EPC case study.

3 EPC

The concept of Event-driven Process Chains (EPCs) is to provide an intuitive modeling language to model business processes. They were introduced by Keller, Nüttgens and Scheer in 1992 [14], and are widely used (for example, in the context of SAP R/3 and ARIS). It is important to realize that the language is not intended to be a *formal* specification of a business process.

An EPC consists of three main elements. Combined, these elements define the flow of a business process as a chain of events. The elements used are:

Functions, which are the basic building blocks. A function corresponds to an activity (task, process step) which needs to be executed. A function is drawn as a box with rounded corners.

Events, which describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the position of one function and act as a precondition of another function. Events are drawn as hexagons.

Connectors, which can be used to connect functions and events. This way, the flow of control is specified. There are three types of connectors: $\wedge$ (and), $\times$ (xor) and $\vee$ (or). Connectors are drawn as circles, showing the type in the center of the circle.

Functions, events and connectors can be connected with edges in such a way that:

- events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (that is, an incoming or an outgoing edge),
- functions have precisely one incoming edge and precisely one outgoing edge,
- connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and
- in every path, functions and events alternate (no two functions are connected and no two events are connected, not even when there are connectors in between.)

From the definition of an EPC it is clear that a process always starts when a certain event occurs. Such an event should be one of the events without incoming edges (start events). After the process is finished, the events that have not been dealt with yet should be events without outgoing edges (final events). If this is the case, we call the EPC *correct*.

Figure 6 shows an example EPC, containing five functions, four start events, three final events, five regular events, and eight connectors (of various kinds).

3.1 Mapping

Usually, mapping an EPC to a WF-net becomes hard if OR-join connectors are present. As mentioned earlier, an EPC is not a formal specification. This is mainly due to the (non-local) semantics of OR-join constructors [2, 3, 16, 26]: When is an OR-join enabled?. However, our approach more or less bypasses this problem, as our focus of attention is an entire execution path from state $[i]$ to state $[o]$. For our approach, the question is not *when* an OR-join constructor is executed, but more *whether* it is executed. The order in which transitions are executed is not relevant for us. As a result, for our approach, it suffices to map an OR-join constructor with n inputs onto $2^n - 1$ transitions (a transition for every possible non-empty set of inputs). Figure 7 visualizes the mapping. Note each node (function, event, or connector) in the EPC is mapped onto one or more start transitions, one place, and one or more end transitions, and that each edge is mapped onto one place. Also note that, for sake of simplicity, we have assumed that connectors can have multiple inputs and multiple outputs.

The only remaining issue for the mapping is the fact that an EPC may contain multiple start and/or final events. This we could solve by introducing a new

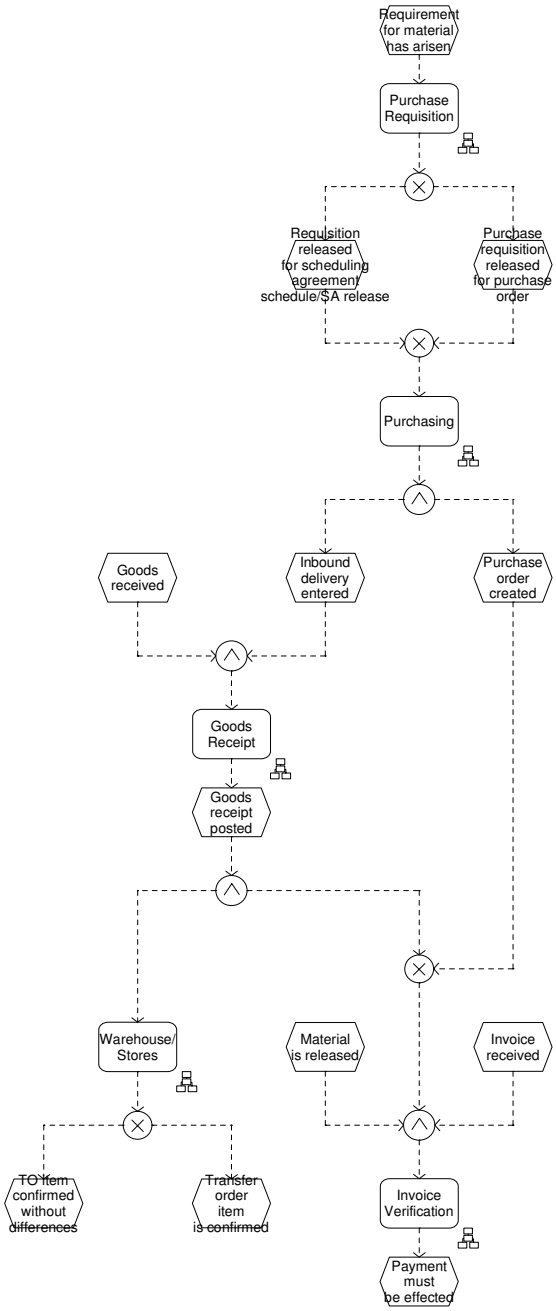


Fig. 6. An example EPC

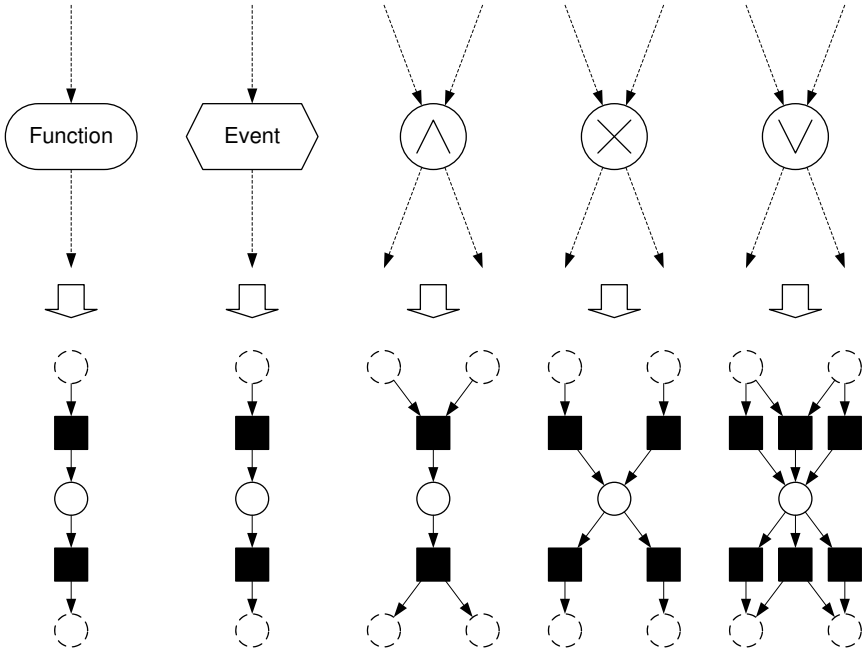


Fig. 7. Mapping EPC constructs

start event and a new OR-join constructor. For the new OR-join constructor, the new start event acts as its only input, and the old start events as its outputs (analogous for final events). However, for example, EPCs are known that contain more than 20 start events. In such a case, an OR-join construct would be inserted that would have to be mapped onto more than a million transitions. For this reason, we have chosen another solution, which is visualized by Figure 8. Note that this solution scales much better than the OR-join solution: For every additional start or final event, two additional transition and three additional places are required.

Please note that OR-join constructs present in an EPC also could have a large number of inputs or outputs. However, we haven't come across such example, hence we assume this to be not a problem. (Also note that this mapping allows for not selecting a single path.) Nevertheless, if such examples would surface, then we could change the mapping of the OR-join constructs using a similar approach.

Figure 9 shows the result of mapping the example EPC as shown in Figure 6.

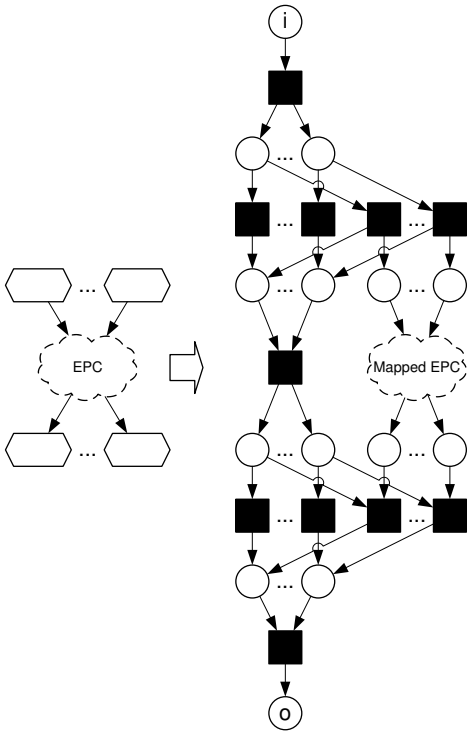


Fig. 8. Mapping start- and final events

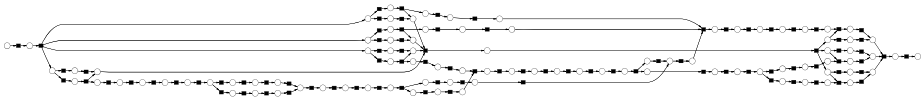
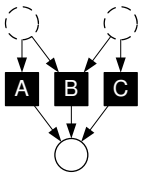


Fig. 9. Mapped example EPC

3.2 Diagnostics based on T-invariants

Given the mapping as presented in the previous section, and using a minimal set of T-invariants in an earlier section, we now detail the possibilities our approach offers. Recall that the goal of our approach is to signal to the user anomalies on the EPC-level. Thus, we are only interested in functions, events, and/or connectors. As a result, we can concentrate on parts of the net generated using Figure 7 and we can ignore parts added using Figure 8. Furthermore, for our explanation, we only need to take the OR connector into account, as all the other constructs are specializations of this connector. Finally, the input behavior and output behavior of an OR connector are similar. Therefore, we restrict ourselves to the input behavior of the OR connector. Figure 10 shows this input

behavior, using labels for the different transitions. Figure 10 also shows the possible warnings our approach generates, depending on which transitions are covered and which are not.



Covered		Warning
Yes	No	
A, B, C		None.
A, B	C	None.
A, C	B	This OR connector could have been an XOR connector.
B, C	A	None.
A	B, C	The rightmost input of this OR connector is not viable.
B	A, C	This OR connector could have been an AND connector.
C	A, B	The leftmost input of this OR connector is not viable.
	A, B, C	This OR connector is not viable.

Fig. 10. Possibilities of our approach

Please note that our approach is not restricted to OR connectors having two input and/or two outputs. For sake of simplicity, we have restricted the figures to OR connectors having only two inputs and two outputs, but our approach also works if there are more than two inputs and/or outputs. Basically,

- an OR connector can be an AND connector if only the transition connected to all inputs (outputs) is covered;
- an OR connector can be an XOR connector if only transitions having one input and one output are covered;
- an input (output) is not viable if none of the connected transitions is covered;
- a function, event, or connector is not viable if none of the transitions is covered.

4 Case studies

4.1 Procurement reference model

To test the approach, we use a SAP Reference Model from the ARIS for MySAP toolset, namely the Procurement reference model. The Procurement reference model contains seven EPCs: `Internal Procurement` (which is the example EPC shown in Figure 6, `Pipeline Processing`, `Procurement of Materials and External Services`, `Procurement on a Consignment Basis`, `Procurement via`

`Subcontracting`, `Return Deliveries`, and `Source Administration`. All seven EPCs were mapped onto one WF-net by simply constructing one large EPC containing all seven EPCs, a new start event, a new final event, and two new XOR-join constructs (one XOR-split for the start events and one XOR-join for the final events). The resulting WF-net contains 553 transitions and 556 places, and is visualized by Figure 11.

To investigate the performance of our approach, we defined six actions, and registered the time it took to perform these actions. The first action was just to load the net, and acts as a base of reference for the other results. The second action was to load the net and to apply well-known liveness and boundedness preserving reduction rules [18] on it, yielding a condensed net. The third action was to load the net and to construct a report based on the T-invariants. The fourth action was to load the net, reduce it, and construct a report based on the T-invariants. The fifth action was to load the net and construct the report based on the relaxed soundness property (that is, based on the state space). The sixth and last action was to load the net, reduce it, and construct a report based on the relaxed soundness property.

For the test program we relied on *Woflan* [23, 25]. Woflan is a Petri-net-based process verification tool, and is capable of constructing coverability graphs, minimal T-invariants, and more.

Table 1. Obtained test results

Action	Times (in seconds)					Average (in seconds)
1 (load)	0.18	0.20	0.19	0.18	0.20	0.19
2 (load, reduce)	0.21	0.21	0.22	0.22	0.22	0.22
3 (load, T-inv.)	0.23	0.23	0.25	0.24	0.24	0.24
4 (load, reduce, T-inv.)	0.22	0.22	0.22	0.23	0.25	0.23
5 (load, state space)	More than ten minutes					More than ten minutes
6 (load, reduce, state space)	13.97	13.78	13.64	13.52	13.81	13.74

Table 1 shows the results obtained (we used a Pentium 4 3.00 Ghz computer with 3.49 GB of RAM running Microsoft Windows XP Professional Version 2002 Service Pack 2). This table shows why we included reduction techniques for this model: The fifth action was aborted after ten minutes in all five cases. As a result, we did not obtain a relaxed-soundness based report we could compare to the T-invariants based report. Fortunately, for the reduced WF-net we did obtain a relaxed-soundness based report. This report was compared to the T-invariants based report obtained for the reduced net, and they turned out to be identical: The *same set of 33 transitions* is not covered by either relaxed soundness or T-invariants. From these reports, we could conclude that the EPCs for `Internal Procurement` and `Procurement via Subcontracting` contain flaws.

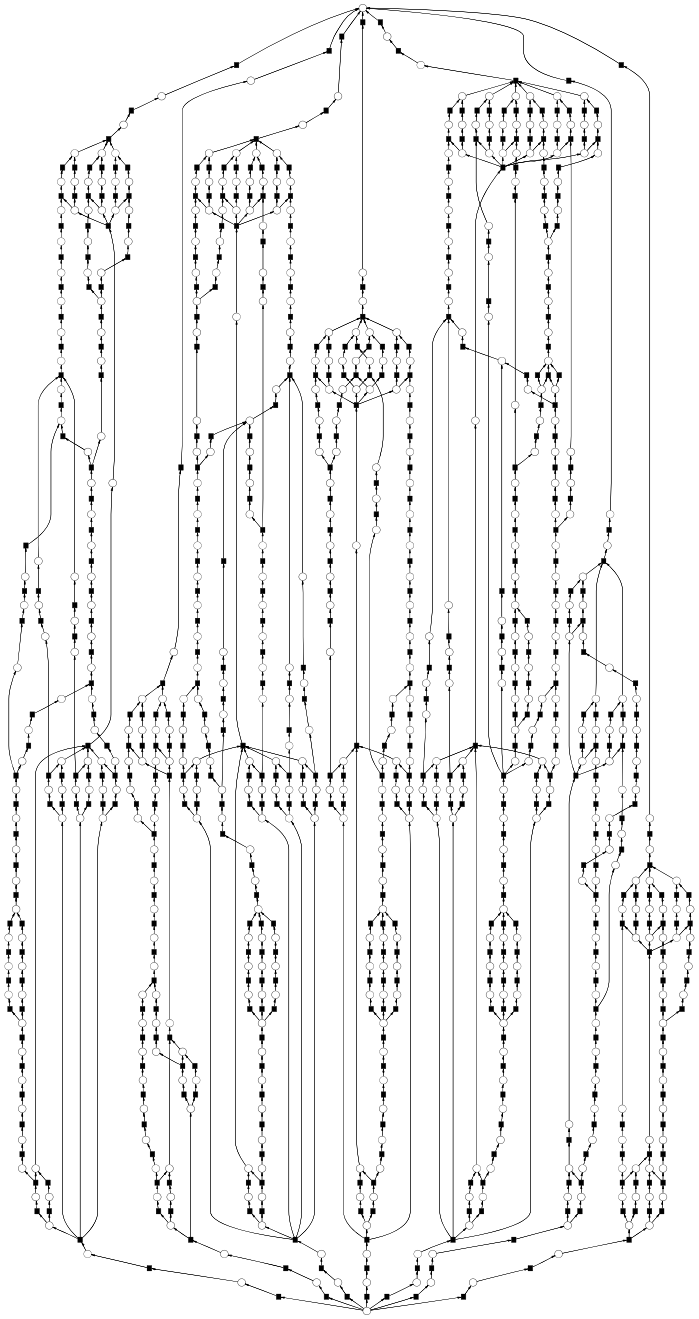


Fig. 11. Mapped Procurement reference model

For the two EPCs, the report contains the following warnings[3]:

Internal Procurement

```
Task "TO item confirmed without differences" should not be executed
Task "Transfer order item is confirmed" should not be executed
Task "Payment must be effected" should not be executed
```

Procurement via Subcontracting

```
Task "Shipping notification transmitted" should not be executed
Task "Material is issued" should not be executed
Task "GR items for components corrected" should not be executed
Task "TO item confirmed without differences" should not be executed
Task "Transfer order item is confirmed" should not be executed
Task "Payment must be effected" should not be executed
```

These warnings indicate that the listed tasks cannot contribute in the proper completion of a case, that is, there is no firing sequence moving the net from $[i]$ to $[o]$ while executing any of the tasks (that is, functions in EPC jargon) mentioned. This implies that a case can only complete properly if these tasks are *not* executed. This corresponds to the bottommost warning listed in Figure 10, that is, none of the transitions linked to these tasks is covered by a suitable T-invariant.

For the Internal Procurement process, all three 'tasks' correspond to final events, and the warnings indicate that none of the three final events should be reached. As a result, we conclude that this process contains a serious error that prevents this process from completing in a proper way. A simple run of this EPC reveals that (see Figure 6) the AND connector following function `Purchasing` is matched by the XOR connector following event `Purchase order created`. Figure 12 shows a snippet of the Internal Procurement EPC (see also Figure 6) and highlights the problem involving these two connectors. Clearly, either the AND connector should be replaced by an XOR connector, or the XOR connector by an AND connector.

For the Procurement via Subcontracting process, something similar holds. Again, none of the final events should be reached, and again a connector mismatch can be located.

4.2 Trade execution process

A second case study involved an even larger model: the model corresponding to a trade execution process of a Dutch bank[4], which contains 17 EPCs. Business

[3] In fact, the warnings read like `Task "TO item confirmed without differences_2sfz##0#####x##/Target/tEvent" should not be executed`, but for sake of readability we have removed certain non-relevant details.

[4] We cannot disclose the name of the bank.

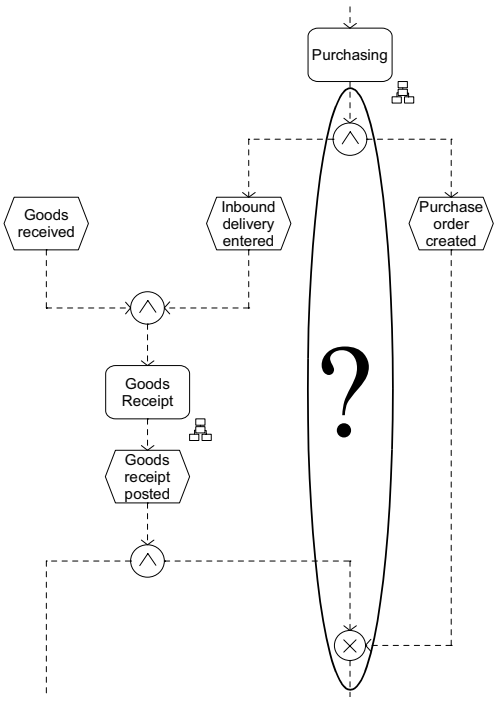


Fig. 12. Error in Internal Procurement

consultants working in the bank made large EPCs modeling the trade execution process. They used the ARIS toolset for modeling their business processes and approached us to verify these processes. To test our approach using T-invariants, we used their model as a starting point. The entire model was mapped onto a WF-net containing 2211 places and 2343 transitions. Figure 13 visualizes this process *after* the liveness and boundedness preserving reduction rules have been applied.

Table 2 shows the results obtained for the six actions in the previous section.

Table 2. Obtained test results

Action	Times (in seconds)					Average (in seconds)
1 (load)	1.44	1.46	1.42	1.42	1.42	1.43
2 (load, reduce)	1.55	1.54	1.53	1.55	1.55	1.54
3 (load, T-inv.)	2.48	2.50	2.52	2.47	2.51	2.50
4 (load, reduce, T-inv.)	1.95	1.94	1.95	1.95	1.94	1.95
5 (load, state space)	More than ten minutes					More than ten minutes
6 (load, reduce, state space)	1669.44	1669.10	1667.82	1669.95	1667.29	1668.72

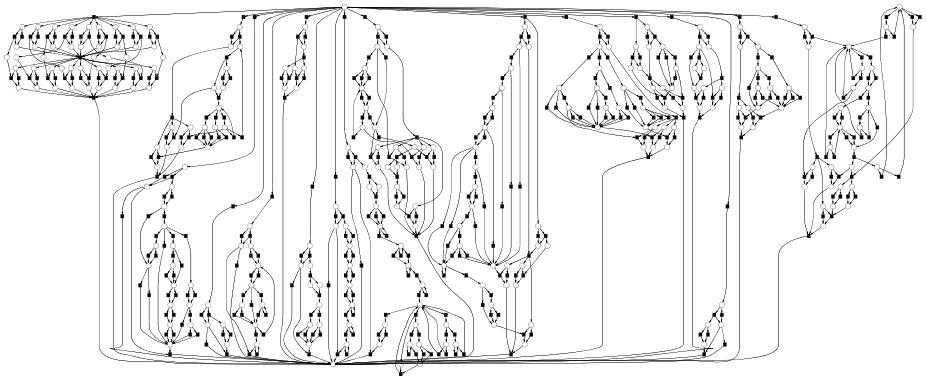


Fig. 13. Mapped trade execution model (after reductions)

Again, our approach was able to pinpoint errors in the model. Analysis using T-invariants revealed that in one of the EPCs, called "0. settlement", the event "trade not ours, therefore neglected" was not covered. As a result, this event can in no way help in completing any case. The findings of this second case study are consistent with the first one. The approach based on T-invariants requires a negligible amount of time compared to the state-based approaches (even after reduction).

5 Related work

Especially in the field of workflow management, some work has been done on mapping EPCs onto Petri nets. Some authors [3, 10, 16] have made attempts to formalize the semantics of EPCs using Petri nets. Others [2, 17] have attempted to map EPCs onto Petri nets regardless of the operational semantics of EPCs. However, often the class of EPCs is restricted to a subclass which can be mapped onto sound Petri net. As a result, the ideas might be appealing from a scientific point of view, but are not useful from a practical point of view.

In her PhD thesis [9], Juliane Dehnert uses EPCs to model processes, and WF-nets to implement them. Although her approach needs the WF-nets to be sound, the EPCs need not be sound (because this would restrict the modeler too much). Instead, she introduces the concept of relaxed soundness and requires the EPCs to be relaxed sound. To decide on soundness and relaxed soundness, the state space needs to be constructed [9]. Our approach with T-invariants comes very close to relaxed soundness, but might not be identical. Nevertheless, as we will argue later on, we could use the T-invariants as a first approximation for relaxed soundness.

The approach of Van Dongen et al. [11] also uses the concepts of soundness and relaxed soundness. Their approach first uses a EPC-based reduction technique, which alleviates the problem of constructing the state space to a large

extent. If the EPC reduces to a trivial EPC, then the EPC is error-free. Otherwise, the EPC is mapped onto a WF-net, its state space is constructed, and, soundness and relaxed soundness are computed. If the WF-net is sound, then the EPC is error-free. If the Petri net is not relaxed sound, then the EPC must contain errors. Otherwise, if the WF-net is relaxed sound but not sound, then the modeler is provided with information to decide whether the model contains errors. The complexity of our approach is better than the approach described in [11]. However, the results may also be less complete: Our approach can conclude that errors exist, but not that errors cannot exist. Note that in [11] the authors also abstract from an operational semantics for EPCs. Unlike our approach based on T-invariants, this approach does not detect OR connectors that can be replaced by AND or XOR connectors, but can detect 'redundant' connectors (connectors with only one input and one output). This approach has been used in [12] to also verify the SAP R/3 Procurement reference models. Although both techniques use completely different approaches, the results obtained are comparable.

6 Conclusions

As demonstrated in this paper, our approach using T-invariants is correct and provides a viable alternative for more time-consuming methods constructing the state space. The approach has been implemented in the context of our analysis tool Woflan [23, 25]. A major advantage of using T-invariants is that they do not need the entire state space to be constructed. As a result, an approach based on T-invariants typically outperforms an approach based on the state space. *The two case studies have shown that the difference can be an order of magnitude: The reports based on T-invariants were generated in seconds, while the reports based on the corresponding state spaces needed at least ten minutes.*

However, the approach using T-invariants is not a complete approach: A process model for which the T-invariants report no errors, can still contain errors. Therefore, our approach cannot guarantee that no errors exist. On the other side, our approach could also be used in a non-EPC setting. For example, it could also be used in a YAWL [4] or BPEL [8, 5] setting. The only requirement is the possibility to map the process model onto a WF-net. Note that like EPCs, YAWL also contains the concepts of OR connectors. Unlike EPCs however, the semantics of YAWL have been formalized [26]. Nevertheless, as our approach abstracts from these operational semantics, our approach would have been able to verify YAWL process models even if a different semantics for the OR-join would have been chosen.

We feel that our approach can be used successfully as a first check in a verification process. As mentioned earlier, generating a report based on T-invariants took only a fraction of a second. Therefore, it could be used even as a on-the-fly check that is running in the background. While the designer is working on a process model, valuable information can be reported almost immediately. For

example, the errors made in the SAP reference model for Procurement would have been spotted by such a check.

References

1. W.M.P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, France, June 1997. Springer, Berlin, Germany.
2. W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.
3. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80. Gesellschaft für Informatik, Bonn, 2002.
4. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
5. BEA, IBM, Microsoft, SAP AG, and Siebel Systems. Business process execution language for web services (version 1.1). ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf (last visited in March 2005), 2003.
6. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.
7. J.-M. Colom and M. Silva. Convex geometry and semiflows in P/T nets: A comparative study of algorithms for computation of minimal P-semiflows. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer, Berlin, Germany, 1990.
8. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.0. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
9. J. Dehnert. *A Methodology for Workflow Modelling: from Business Process Modelling towards Sound Workflow Specification.* PhD thesis, Technische Universität Berlin, Berlin, Germany, August 2003.
10. J. Dehnert and W.M.P. van der Aalst. Bridging the gap between business models and workflow specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
11. B.F. van Dongen, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of EPCs: Using reduction rules and Petri nets. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer, Berlin, Germany, June 13-17 2005.
12. B.F. van Dongen and M.H. Vullers-Jansen. Verification of SAP reference models. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 464–469. Springer, Berlin, Germany, 2005.

13. P. Godefroid. *Partial-order methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1996.
14. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische processmodellierung auf der grundlage ereignisgesteuerter processketten (epk). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.
15. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
16. E. Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer, Berlin, Germany, 2004.
17. P. Langner, C. Schneider, and J. Wehler. Petri net based certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer, Berlin, Germany, 1998.
18. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
19. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.
20. A.W. Scheer. *Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises*. Springer, Berlin, Germany, 1994.
21. K. Varpaaniemi. *On the Stubborn Set Method in Reduced State Space Generation*. PhD thesis, Helsinki University of Technology, Helsinki, Finland, May 1998.
22. H.M.W. Verbeek. *Verification of WF-nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, June 2004. BETA dissertation series D65.
23. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based workflow diagnosis tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer, Berlin, Germany, 2000.
24. H.M.W. Verbeek and T. Basten. Deciding life-cycle inheritance on Petri nets. In W.M.P. van der Aalst and E. Best, editors, *24th International Conference on Application and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 44–63, Eindhoven, The Netherlands, June 2003. Springer, Berlin, Germany.
25. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnozing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
26. 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, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443. Springer, Berlin, Germany, 2005.