Where did I go wrong?

Explaining errors in business process models

Niels Lohmann

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Business process modeling is still a challenging task — especially since more and more aspects are added to the models, such as data lifecycles, security constraints, or compliance rules. At the same time, formal methods allow for the detection of errors in the early modeling phase. Detected errors are usually explained with a path from the initial to the error state. These paths can grow unmanageably and make the understanding and fixing of errors very time consuming. This paper addresses this issue and proposes a more intelligible explanation of errors: Instead of listing the actions on the path to the error, only which decisions lead to it are reported and highlighted in the original model.

1 Introduction

Business process modeling is a sophisticated task and received a lot of attention in the past decades. With the advent of domain-specific languages and a growing scientific community, the act of creating and managing business process models has become a discipline on its own. Despite all efforts, design flaws may still occur. This can have different impacts, ranging from syntactically incorrect models, which are harder to understand, up to catastrophic faults and down times in the execution that yield to a loss of money or a legal aftermath. Consequently, a large branch of research focuses in the detection, correction, and avoidance of errors in business process models. Whereas plain control flow analysis is now well understood, other aspects such as data, business rules, or security may introduce more subtle flaws that are harder to detect.

Using the prominent *soundness* [1] property, we can classify existing approaches into three classes: (1) Some approaches exploit certain structural constraints of the business process model, for instance by focussing on workflow graphs that only consist of AND/XOR-gateways, for instance [10]. (2) Other approaches rely on the definition of soundness which can be defined in terms of standard Petri net properties such as boundedness, liveness, or the existence of place invariants [11]. The two mentioned approaches are *domain-specific* in the sense that they exploit the fact that they investigate business process models. In contrast, (3) general purpose verification tools (usually called *model checkers* [3]) can check all kinds of properties as long as they can be expressed in terms of temporal logics. As this is the case for soundness, these tools are also applicable for the verification of business process models.

Due to the ongoing evolution of business process modeling languages, the growing number of aspects that need to be covered by a business process model,

N. Herzberg, M. Kunze (eds.): Proceedings of the 6th Central European Workshop on Services and their Composition (ZEUS 2014), Potsdam, Germany, 27-03-2014, published at http://ceur-ws.org

or the trend toward executable business process models, the verification of business process models has become a moving target. As a consequence, specific approaches may become inapplicable for novel demands, leaving only general purpose approaches as stable tools for the future.

Problem description. In principle, a model checker takes a formal model (e.g., a Petri net) and a formal description of the property to check (usually described by temporal logic formula φ) as input and tries to prove the property by an exhaustive investigation of the model's states. In case the property is violated (e.g., a deadlocking state is detected), a path π to this error state is reported [3]. The path contains all actions of the model that need to be executed to reach the error state from the initial state. Due to this operational nature of paths, the scenario that led to the error can be simulated. It is furthermore possible to explain the scenario in terms of the original model; that is, to map the states of the Petri net back to events of a BPMN model.

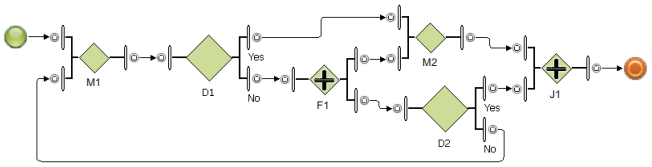
Unfortunately, the size of the paths correlates with the size of the model and paths of industrial models can thus be very long and hardly understandable. Furthermore, the path can contain a lot of irrelevant or diverting information that makes the comprehension of the error very difficult. For instance, the path usually contains actions that only "set up" the process (e.g., initializations and login procedures). These inevitable actions are certainly *necessary* to be able to reach the error state, but are usually not the *cause* of it. Another aspect that makes paths hard to understand is the fact that business process models may span several components where activities are executed in parallel. On the path, these originally unordered activities are reported in a fixed — and possibly arbitrary — order which may yield confusion due to unintuitive error descriptions.

Contribution. This paper addresses the mentioned problems by shortening paths by focussing on the choices made rather than on each individual action. We shall use a large case study as experimental evaluation of our proposed approach.

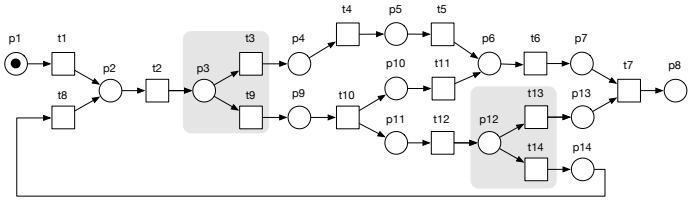
2 Model checking Petri nets

Business process modeling languages are usually semiformal and hence are not directly applicable to a mathematically rigorous proof of correctness criteria. However, the operational semantics can be captured in formalisms such as Petri nets or process calculi. With the advent of executable languages such as WS-BPEL 2.0 or BPMN 2.0, such a formalization became much easier, because a precise execution semantics yielded more careful language specifications. In fact, for most of today's languages from industry or academia, translations into *Petri nets* [9] exists [7].

Example. Figure 1(a) depicts a small business process model from [4] which contains two subtle control flow errors: a lack of synchronization and a local deadlock. Its translation into a Petri net is shown in Fig. 1(b). As we see, the Petri net's structure is very similar to the original model.



(a) business process model from [4]



(b) Petri net model

Fig. 1. A business process model (a) and its translation into a Petri net (b)

Model checking [3] is an approach to prove that a system satisfied a given correctness criterion; for instance soundness, the absence of a deadlocking state, the presence of a sound process configuration, correct data life cycles, or compliance to business rules. In contrast to *theorem provers*, which sometimes need manual inputs, or *testing*, which can only prove the existence of errors, but never their absence, model checking is an automated and complete way to investigate systems.

For the remainder of the paper, we use model checking tool LoLA [12] that takes a Petri net N and a temporal logical formula φ as input. If the formula is satisfied by the Petri net (e.g., if the Petri net is sound), this is reported as "yes" to the modeler. In case the formula is violated (e.g., a deadlocking marking m is found), this is reported as "no" to the modeler. In addition, a path $\pi = t_1 \cdots t_n$ is given to the modeler which explains how m is reachable from the initial marking m_0; that is, $m_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} m$. Depending on the nature of the formula φ, the marking reached by the reported path either is a proof that the formula is not satisfied by the behavior of the Petri net N and is called a *counterexample* or marking itself is the proof that the formula is satisfied (e.g., if φ expresses the reachability of that marking m) and is called a *witness*. In this paper, we do not distinguish the semantics of the marking m and always refer to m as *goal marking*.

Example (cont.). The business process from Fig. 1(a) has a lack of synchronization. This can be detected by checking the Petri net from Fig. 1(b). The following path π describes how a marking m can be reached which puts two tokens on place p_6.

$$\pi = t_1\ t_2\ t_9\ t_{10}\ t_{11}\ t_{12}\ t_{14}\ t_8\ t_2\ t_3\ t_4\ t_5 \qquad m = \{p_6 \mapsto 2\}$$

The path contains 12 transitions. In the remainder of this paper, we use this path to exemplify the proposed reductions.

It is worthwhile to mention that model checking suffers a devastating worst case complexity due to the well-known state explosion problem which yields reachability graphs with exponential blow-ups compared to the size of the models. However, even industrial business process models can be model checked in few microseconds, because heuristics that fight the state space explosion proved to be very effective in this domain [4].

3 Representing paths by made choices

3.1 The problem: long paths = big problems

In the remainder of the paper, we focus on the following problem:

> Given a path π to a goal marking m of a Petri net model N, how can the reason for the error modeled by m be briefly and comprehensively explained to the modeler of N?

Apparently, π describes how the goal marking m can be reached from the initial marking m_0 of N. Consequently, reporting the transitions of π together with the intermediated markings to the modeler should help to understand the reasons m was reached. Unfortunately, this approach is futile in case π contains dozens of transitions. The reasons for such long paths are:

Detours: Model checkers usually investigate the markings of a Petri net in a depth first search[1]. As a result, the reported paths do not need to be optimal and may contain some transitions that model "detours" in the reachability graph that do not contribute in the actual reaching of the goal marking.

Interleaving of concurrent transitions: A marking of N may activate two transitions t_1 and t_2 which are not mutually exclusive. That is, firing either transition first does not disable the other one. A typical reason for this is that t_1 and t_2 do not share any resources. Consequently, the order in which t_1 and t_2 occur on the path π is arbitrary. If each transition belongs to different components of the underlying business process model, then these arbitrary interleaving of the transitions may be irritating to the modeler if she tries to understand the path π. In the example path, transition t_{11} and t_{12} are concurrent and the reported order in path π (t_{11} before t_{12}) is arbitrary.

Indisputable parts: Though the path π is an actual proof *that* the goal marking m can be reached in N, not every transition on the path is an actual cause of m. In the example process, *any* path will begin with firing t_1 and hence does not need to be reported to the modeler as reason for an error.

[1] Breadth-first approaches are not applicable to many classes of formulae.

3.2 The solution: don't report the obvious

To tackle the problem of long paths with redundant or unhelpful information, we shall exploit two aspects to shorten paths in the remainder of this section: *progress* and *conflicts*.

Progress is the assumption that the model never "gets stuck" in case a transition is activated. That is, if a marking activates one or more transitions, then this marking is eventually left by firing on of these transitions. Progress is a natural assumption for business process models in which the execution of tasks also cannot be postponed indefinitely. Though the actual occurrence of message or timer events cannot be precisely predicted, the respective states are always assumed to be eventually left by the modeled actions.

A *conflict* is a situation in which there exist more than one possible continuations (e.g., an XOR gateway). In terms of Petri nets, it is a marking in which two transitions t_1 and t_2 are enabled, but after firing either of them, the other transition is disabled. This situation is dual to concurrent transitions (see above) that do not disable each other. A detailed discussion of these aspects can be found in [9].

The combination of these aspects brings us to the following intuitive observation: *Only the conflicts on the path π carry information on how to reach the goal markings.* Any other marking m on the path between the initial and the goal marking either (1) enables no transition: Then this must be the goal marking itself, because it has no successor marking. Alternatively, (2) marking m enables exactly one transition: Then this transition is eventually fired due to the assumption of progress. Consequently, this transition does not need to be reported to the modeler as its firing was already determined by the previous transition on π thad lead to m. Finally, (3) marking m enables several concurrent transitions. These transitions may fire independently, and if all of them are on π, then the exact order is arbitrary.

Example (cont.). The conflicts of our running example are shaded gray in Fig. 1(b): transitions t_3 and t_9, as well as t_{13} and t_{14} are conflicting. As a result, we can reduce the path π as follows:

$$\pi_{\text{reduced}} = t_9\ t_{14}\ t_3 \qquad m = \{p_6 \mapsto 2\}$$

The firing of all other transitions is clear from the context from the intermediate markings and the assumption of progress. Note that the transition names need to be translated back into the terms of the original model. A different representation of π_{reduced} could be: "After (1) decision D1: *No*, (2) decision D2: *No*, and (3) decision D1: *Yes*, a lack of synchronization occurs after after merge M2." This is depicted in Fig. 2.

3.3 Experimental results

To evaluate the path reduction algorithm, we applied it to a large collection of industrial process models created by IBM customers using the *IBM WebSphere*

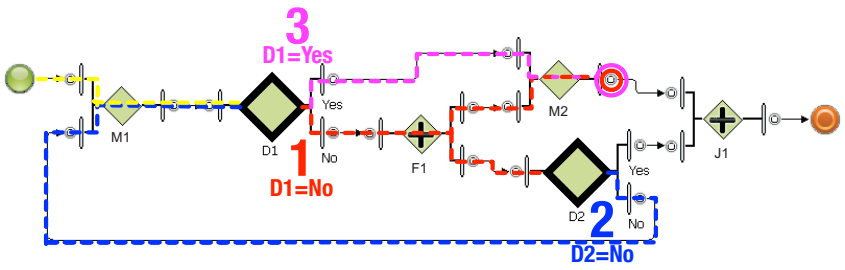


Fig. 2. Mapping back the reduced path run to the original process model

Business Modeler. The models were first presented in a report by Fahland et al. [4], where the 1386 process models were checked for soundness using different approaches. As a general-purpose model checker, *LoLA* [12], took part in this investigation, the process models were also translated into Petri nets.[2] The models are partitioned into five libraries (A, B1, B2, B3, C) and stem from different business areas, ranging from financial services, automotive, telecommunications, construction, supply chain, health care, and customer relationship management.

As experiment, we replayed verification checks reported in [4,2] and reduced the generated paths. The results summarized in Table 1–3 report are promising: we report reductions between 77 % and 95 %, leaving average reduced path lengths between 2 and 7 transitions. Though the reduced paths consist of Petri net transitions, it can be easily translated back into the nomenclature of the original model as demonstrated in Fig. 2.

4 Further reduction: remove spurious conflicts

In the previous section, we showed how paths to errors in business process models can be reduced by only reporting conflict transitions. This reduction decided, for each marking that activates a transition, whether conflicting transitions are also activated. This check is local in the sense that it is not checked whether those transitions that were not taken in the decisions actually could have avoided the next conflict transition on the path.

Intuitively, a transition t_i on a reduced path π is a spurious conflict iff every transition t in conflict to t_i eventually reaches the marking m_{i+1} which enables the next transition t_{i+1} on path π. In this case, choosing any transition from the ith conflict will eventually enable the next conflict on the path to the goal state. Consequently, reporting the spurious conflict t_i is of little help to the modeler to understand the error itself.

The check for spurious transitions defined above can be straightforwardly be implemented using a model checker.[3] We integrated this check as postprocessing

[2] The original models and their Petri net translations are available for download at `http://service-technology.org/soundness`.

[3] We check whether N with initial marking m_i' satisfies the CTL formula $\varphi = \mathbf{AF}\, m_{i+1}$.

Table 1. Paths from the checks for local deadlocks [4]

library	A	B1	B2	B3	C
avg. path length before / after	17.51 / 1.83	17.52 / 2.11	16.06 / 1.54	20.34 / 1.67	13.40 / 2.30
max. path length before / after	53 / 8	66 / 7	56 / 6	54 / 5	21 / 3
sum of path lengths before / after	1699 / 178	1419 / 171	1349 / 129	1688 / 139	134 / 23
reduction	89.52 %	87.95 %	90.44 %	91.77 %	82.84 %

Table 2. Paths from the checks for lack of synchronization [4]

library	A	B1	B2	B3	C
avg. path length before / after	30.83 / 3.17	10.47 / 0.66	12.16 / 0.68	11.50 / 0.59	51.00 / 7.57
max. path length before / after	89 / 13	52 / 7	100 / 8	103 / 14	120 / 17
sum of path lengths before / after	1079 / 111	1047 / 66	1459 / 82	1507 / 77	357 / 53
reduction	89.71 %	93.70 %	94.38 %	94.89 %	85.15 %

Table 3. Paths from the checks for noninterference [2]

library	A	B1	B2	B3	C
avg. path length before / after	12.06 / 2.79	13.82 / 2.55	18.13 / 2.33	14.27 / 2.55	11.27 / 2.33
max. path length before / after	44 / 7	70 / 7	95 / 7	95 / 7	27 / 3
sum of path lengths before / after	19699 / 4557	5707 / 1054	13835 / 1777	17494 / 3130	169 / 35
reduction	76.87 %	81.53 %	87.16 %	82.11 %	79.29 %

step after reducing the paths as described in the previous section. Note that executing a model checker can be very time and memory consuming. However, even if a check is not finished with a reasonable amount of resources, we just failed to proof whether a conflict is spurious and can continue with the investigation of the next transition. That said, the postpocessing can be aborted at any time — any intermediate result is still correct.

We applied the reduction of spurious conflicts to the case studies described in the previous section. Table 4–6 summarize the results. In all three experiments, the paths could be further reduced by 50–86%. Note that in some cases, the check for spurious conflicts has been aborted after more than 2 GB of memory were consumed. In these cases, the conflict was kept in the path and the check proceeded with the next conflict.

5 Concluding remarks

Related work. The analysis and verification of business process models is a broad field of research. Consequently, there exists a variety of domain-specific approaches (e.g., the decomposition of workflow graphs into SESE regions to check soundness [10]). However, we are not aware of other approaches that postprocess error information from general purpose model checkers to explain these errors to the modelers.

Related to the presentation of error information is the automated correction of flawed business process models [6,5]. These approaches use similarity metrics

Table 4. Reduced paths from the checks for local deadlocks

library	A	B1	B2	B3	C
avg. path length before / after	1.84 / 0.91	2.11 / 0.67	1.54 / 0.57	1.67 / 0.41	2.30 / 0.90
max. path length before / after	8 / 2	7 / 1	6 / 1	5 / 1	3 / 1
sum of path lengths before / after	178 / 88	171 / 54	129 / 49	139 / 34	23 / 10
reduction	50.56 %	68.42 %	62.79 %	75.54 %	60.87 %
aborted checks	1	0	0	0	0

Table 5. Reduced paths from the checks for lack of synchronization

library	A	B1	B2	B3	C
avg. path length before / after	3.17 / 0.86	0.66 / 0.17	0.68 / 0.14	0.59 / 0.09	7.57 / 1.00
max. path length before / after	13 / 2	7 / 2	8 / 2	14 / 2	17 / 2
sum of path lengths before / after	111 / 30	66 / 17	82 / 17	72 / 12	53 / 7
reduction	72.97 %	54.55 %	79.27 %	84.42 %	86.79 %
aborted checks	1	4	0	0	4

Table 6. Reduced paths from the checks for noninterference

library	A	B1	B2	B3	C
avg. path length before / after	2.79 / 0.99	2.55 / 0.75	2.33 / 0.55	2.55 / 0.63	2.33 / 0.40
max. path length before / after	7 / 2	7 / 2	7 / 2	7 / 2	3 / 1
sum of path lengths before / after	4557 / 1614	1054 / 310	1777 / 423	3130 / 772	35 / 6
reduction	64.58 %	70.59 %	76.20 %	75.34 %	82.86 %
aborted checks	12	4	4	7	0

to find a correct business process model which maximally resembles the flawed model. These approaches have the benefit of avoiding lengthy manual correction steps altogether.

Future work. In this paper, we focused on reducing paths to error states and neglected the retranslation into the original business process model. Visualizations such as Fig. 2, possibly enriched with animations, need to be automated and evaluated by business process modelers. Here, understandability criteria [8] could be of great value. However, this was out of scope of this paper which aimed at evaluating the idea of using conflicts to reduce paths with three experimental setups checking different correctness criteria with thousands of industrial business process models.

We see in this paper a first step toward a diagnosis framework which uses general purpose verification tools to verify business process models. As motivated in the introduction, domain-specific approaches are very closely coupled to the structure or the property under investigation, but may become inapplicable for future developments. In contrast, the modularization (a translation into Petri nets as frontend, a general purpose model checking tool as middleware, and a diagnosis framework as backend) may be more flexible when it comes to novel business process languages and properties.

References

1. W. M. P. v. d. Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. R. Accorsi and A. Lehmann. Automatic information flow analysis of business process models. In *BPM 2012*, LNCS 7481, pages 172–187. Springer, 2012.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
4. D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Instantaneous soundness checking of industrial business process models. In *BPM 2009*, LNCS 5701, pages 278–293. Springer, 2009.
5. M. Gambini, M. La Rosa, S. Migliorini, and A. ter Hofstede. Automated error correction of business process models. In *BPM 2011*, LNCS 6896, pages 148–165. Springer, 2011.
6. N. Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *BPM 2008*, LNCS 5240, pages 132–147. Springer, 2008.
7. N. Lohmann, H. Verbeek, and R. M. Dijkman. Petri net transformations for business processes – a survey. *LNCS ToPNoC*, II(5460):46–63, 2009.
8. J. Mendling, H. A. Reijers, and J. Cardoso. What makes process models understandable? In *BPM 2007*, LNCS 4714, pages 48–63. Springer, 2007.
9. W. Reisig. *Petri Nets*. Springer, EATCS Monographs on Theoretical Computer Science edition, 1985.
10. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and more focused control-flow analysis for business process models through SESE decomposition. In *ICSOC 2007*, LNCS 4749, pages 43–55. Springer, 2007.
11. H. M. W. Verbeek, T. Basten, and W. M. P. v. d. Aalst. Diagnosing workflow processes using Woflan. *Comput. J.*, 44(4):246–279, 2001.
12. K. Wolf. Generating Petri net state spaces. In *ICATPN 2007*, LNCS 4546, pages 29–42. Springer, 2007.