Visual Analytics for Soundness Verification of Process Models

H. S. Garcia Caballero[1], M. A. Westenberg[1], H. M. W. Verbeek[1], and Wil M. P. van der Aalst[1]

Eindhoven University of Technology, Eindhoven, The Netherlands,
h.s.garcia.caballero@tue.nl

Abstract. Soundness validation of process models is a complex task for process modelers due to all the factors that must be taken into account. Although there are tools to verify this property, they do not provide users enough information about where soundness starts breaking and under which conditions. Providing insights such as states in which problems occur, involved activities, or paths leading to those states, is crucial for process modelers to better understand why the model is not sound. In this paper we address the problem of validating the soundness property of a process model by using a novel visual approach. PSVis (Petri net Soundness Visualization) aims to guide expert users through the process models in order to get insights into the problems that cause the process to not be sound.

Key words: Petri nets, soundness, verification, process models, process mining, visualization, visual analytics

1 Introduction

The use of process models for workflows has been studied for some decades now. It started back in 1979 with the work of Skip Ellis on office automation [9]. Still, it took two decades until notions such as workflow nets and soundness were defined [1] thus linking workflows to Petri nets. As a result of this link, a lot of existing Petri net theory (like [7, 13, 15, 19]) became instantly applicable to the workflow process model domain. Nevertheless, some other approaches to the verification of these models also still emerged, in which was used [20] another graph-like notation for processes in combination with dedicated graph reduction rules.

The application of Petri nets to the workflow domain [2, 3, 12] triggered a new line of research focussing on the soundness verification tool *Woflan* [29], different variations on soundness [5, 27], and extensions to the Petri net formalism (like EPCs [11, 25, 26], BPEL [14], and YAWL [30, 32]). Verbeek et al. [29] also introduced the concepts of the problematic runs (called *sequences* in that paper), which are used in this paper.

None of these approaches offered a comprehensive visualization of the problems using the process model of choice (YAWL, EPC, Petri net, etc.). For instance, the Woflan tool did not visualize the Petri net it was checking soundness on, but just showed a series of messages that included the labels of the nodes in the net. Nevertheless, the Woflan tool was later included in the process mining framework ProM [22, 26], which did allow this net to be visualized. As a result of this inclusion, selected markings (like deadlock

markings) could be visualized by projecting them onto the net, and other Petri-net related properties (like invariants) could also be visualized. However, such visualization means are still limited and users struggle to diagnose real problems.

This paper takes this initial and rudimentary visualization of soundness problems in ProM some steps further by focusing on the visualization. Also, whereas Woflan requires a unique final marking (which should be reached to achieve success in the workflow), the approach in this paper allows for any collection of such final markings. By visualizing any problem that prevents the workflow from reaching any of these final markings, the user is guided towards correcting the root cause of these problems.

The concept of runs as shown by the visualization is known in the Petri net field, and originates from Desel [6]. An example tool that supports these runs is VipTool [4, 8]. VipTool can provide the user with information whether a given *scenario* (say, a partial trace) fits the Petri net at hand. In this paper, we assume that such a partial trace fits the net. If not, we can use alignments to find the closest path in the model [23]. This is not supported by the VipTool, but is supported by ProM. We are more interested in visualizing the execution of the fitting partial trace in the net. Apart from this, VipTool can also synthesize a Petri net from a collection of scenarios. This connects VipTool to the field of process mining [22], which is the natural habitat of ProM. However, we do not use that feature of the tool.

2 Problem Definition

In this section we give definitions of the core concepts that our visualization tool needs to handle. We define the problem that we address, and present a set of tasks. We designed our visualization tool accordingly.

2.1 Basic Notions

Process mining Process mining is a field which involves discovery, monitoring and improvement of real processes by means of knowledge extracted from event logs. It is a middle point between machine learning and data mining, and process modeling and analysis [22].

Petri net The concept of Petri net (PN) was originally defined by Petri [16] and it constitutes the oldest and best investigated process modeling language [22]. A Petri net can be defined as a bipartite graph composed of *places* and *transitions*. Although the structure is static, the behavior is not.

Marking or state A marking consists of tokens assigned to the places of a Petri net [15]. During the execution of a Petri net, the amount and position of tokens may change. Therefore, markings define the execution of a Petri net. Markings are also known as states.

Reachable marking A marking M_n is said to be *reachable* if a sequence of firings that transforms the initial marking M_0 to M_n exists [13]. Therefore, a marking is reachable from another marking as long as there exists a directed path in the state space between the two markings.

State space The set of all markings of a Petri net reachable from the initial marking is called *state space*. The change of state is caused by firing a transition, which represents a change function called *next-state function* [15].

Unbounded place A system is said to be bounded if for every place s there is a natural number b such that the number of tokens in s is at most b for every reachable marking [7]. A place is said to be unbounded if there is no a natural number b such that the number of tokens on this place never exceeds b [18].

Soundness We use a generalization of soundness defined in [2]. In order to ensure the soundness of a Petri net, it has to satisfy the following requirements: 1) the Petri net has an initial marking marking and a finite set of final markings; 2) there is a path in the state space from the initial to each final marking; 3) it is always possible to reach some final marking; and 4) there are no transitions that can never fire.

Causal Run A run is depicted by a causally ordered set of events constituting the causal semantics of a Petri Net [6]. It represents the behavior of the system.

2.2 Problem Analysis

Process modelers tend to ensure the soundness property of the process models. However, it is fairly easy to break this property with only a small number of changes on the model. In addition, models which are derived by discovery algorithms do not always ensure this property. Well-known miners like the Alpha-miner and the Heuristic-miner often produce models that are not sound (e.g., have deadlocks).

Because of the dynamic nature of the behavior of the Petri nets, understanding where the problems occur and the context in which they happen plays a key role for process modelers.

In order to address this problem, we define the following tasks, which form the design basis of our tool:

T1 Obtain an overview of all or a subset of final states of a process model.
T2 Compare disjoint sets of transitions and/or places belonging to a specific area.
T3 Find problematic states.
T4 Explore paths that lead to a problematic state.
T5 Determine when the problem occurs for a specific problematic state.
T6 Analyze the runs for a selection of states.
T7 Examine concurrency, loops and causal order in the runs.

2.3 Soundness Validation

We use an algorithm originally proposed by Verbeek [28] to compute the problematic states of a given model. The output of the algorithm is used as input in our tool. The algorithm computes three sets of states, referred to as *Orange*, *Green* and *Red* areas. An abstraction of the resulting output of this algorithm is shown in Fig. 1.

In our tool, we focus on the border states, that is, the states which depict a transition from the *Orange* area to either *Green* or *Red* area. In Fig. 1, the border states are linked by black-dotted arrows. Notice that there may be cases in which the same transition leads to different areas depending on the source state (e.g., *t1* leads to *Green* and *Red*

areas). These states correspond to parts of the process in which everything becomes well (all reachable states are *Green*) or everything becomes bad (all reachable states are *Red*).

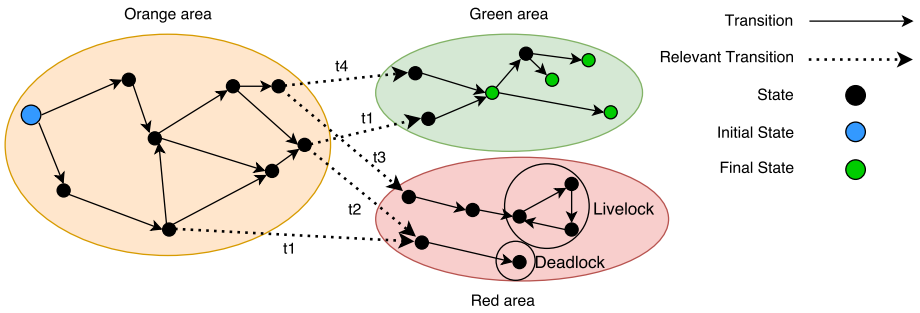

Fig. 1. Abstract representation of the concept of areas introduced in [28]. The state space is divided into three areas: *Orange*, *Green* and *Red*. Dots and arrows depict states of the state space and transitions, respectively. Two possible situations can occur in the *Red* area: deadlocks and livelocks, which are represented by dots without outgoing arrows and cycles.

The *Red* area contains all those states from which is not possible to reach any final marking. Clearly, all reachable states from a *Red* state are *Red* states too. Thus, states within this area are considered as wrong states. On the other hand, the *Green* area represents those states from which a final marking can be reached. Similar to the *Red* states, from a *Green* state only *Green* states can be reached. Lastly, the *Orange* area comprises states from which some *Red*, *Green* and *Orange* states can be reached.

Within the *Red* area, two major problems can be present: deadlocks and livelocks. In Fig. 1, two problems are indicated with circles. Deadlocks occur when a marking from which no transition can be fired is reached, and it is not a final marking. This can be seen in Fig. 1 as the state has no outgoing edges, that is, no transition can be fired at that state. Livelocks happen when a cycle is found, which means that it is possible to iterate between a subset of states for ever.

3 PSVis

In this section we introduce PSVis (Petri net Soundness Visualization), its components and how they interact in order to execute the tasks depicted in Sec. 2.2. An overview of all components is given in Fig. 3.

Our tool assumes that the state space is computable in a reasonable amount of time. If the state space contains unbounded places, it is infinite in size and cannot be computed. But even if all places are bounded, the state space may still be too big to be constructed within reasonable time [29]. To avoid having to spend unreasonable time in constructing the state space, our tool uses a threshold that operates on the number of tokens in a state. If the threshold is set to b, then only states where every place contains

less than b tokens are expanded. This is related to the notion of b-boundedness introduced earlier. Thus, if all places are b-bounded in a net, then setting the threshold to b or higher does not change the state space. If the threshold is reached at some state, we assume that state is a problematic one.

3.1 Glyphs on the Petri net

In order to support T1, process modelers need to visualize the number of tokens of a specific place, and the states that place belongs to. As a result, we introduce *glyphs*, which decorate places in a Petri net model. Glyphs are visual representations of a piece of the data where visual attributes are dictated by data attributes [31].

The number of tokens in a place is represented as dot shapes contained in the place, numbers, or a combination of both. The main problem of this representation arises when we want to visualize more than one final state at the same time. A final state according to the definition is a multiset of places. This implies that a specific place can belong to more than one final state. With the current way of presenting this information, expert users are not able to know the number of tokens contained inside each place for each state and whether a place belongs to more than one state. Therefore, we propose a new way to visually encode this information. This new encode is shown in Fig. 4(b).

In our approach, the state with the highest number of unique places determines the color of the places. Glyphs are then colored with the remainder states. The initial state constitutes an exception and the places that belong to it are always colored accordingly. Fig. 2 shows an example of how our approach works. When the user selects *state 1*, the two places that belong to it are colored red. Next, the user selects *state 2* which has three unique places. Therefore, all the places are blue colored and two glyphs are created to present *state 1*. Lastly, when the user selects the initial state, two places are green colored and glyphs are created to show the remainder states.

If there is more than one token in a place, a label is attached to the glyph (or to the place) indicating the number of tokens. This label is colored dynamically depending on the brightness of the background color. Thus, labels can be colored black or white to make them readable.

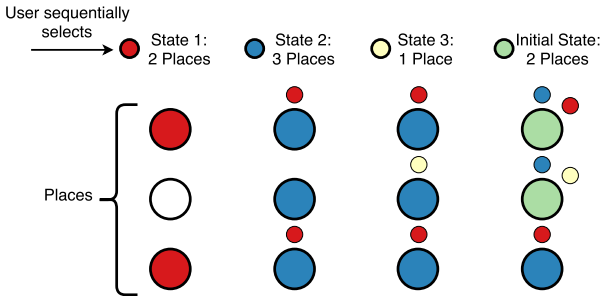

Fig. 2. Assignment of colors to places when selecting states. From left to right, the evolution of the coloring of the places when a user selects states is shown. Glyphs decorating the places of the Petri net are created on demand.

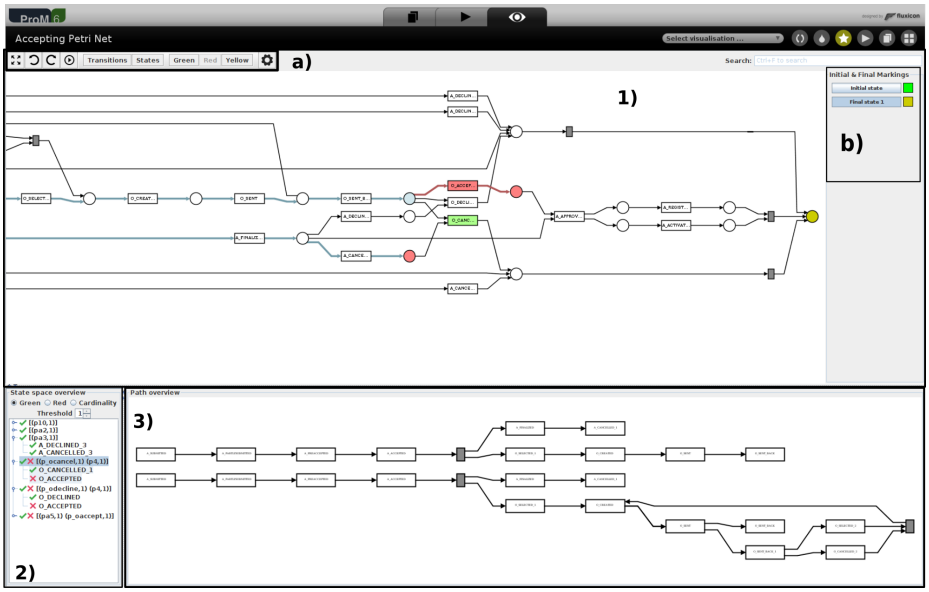

Fig. 3. Overview of our tool. 1) displays a process model as a Petri net in which four places are colored: two in red, indicating the last state reached in the *Red* area, one in blue, indicating that token was consumed in order to reach the *Red* state, and one in golden, indicating the final state of the process model, which was selected in 1.b). 2) shows all the available problematic states, some of them are highlighted indicating that the user selected those ones. Lastly, 3) shows the corresponding runs.

3.2 Petri Net View

This is the main view of our tool. It presents the process model as a Petri net (see Fig. 3(1)) where circles and rectangles depict places and transitions. We implement a version of Sugiyama's approach [21] to layout the Petri net since it gives a good understanding of the flow of the process. Some parameters of the layout algorithm can be modified through the toolbar at the top of the view. In addition, users can perform zooming, panning and dragging of elements directly on the view. Last but not least, nodes can be hovered in order to show the label of such element by using a tooltip.

To the right of the Petri net view, there is a panel which shows the initial and final states of the model. Each state is presented in the tool by two components: a button and a colored rectangle, which are interactive. This component can be seen in Fig. 4(a). The example shows one single initial and five final states. Users can (de)select states with this component and change the assigned colors. When a state is selected, the Petri net view reacts by showing the current selection of states. Given the fact that a single place can be present in multiple states, we use a new approach to represent that a place belongs to several states (see Fig. 2). This feature directly relates to task T1.

The top part of this view (Fig. 3(1.a)) shows two sets of buttons that are dedicated to perform a quick exploration of the problems that have been detected. These buttons enable users to explore places or transitions that belong to just one area (see 2.1). This

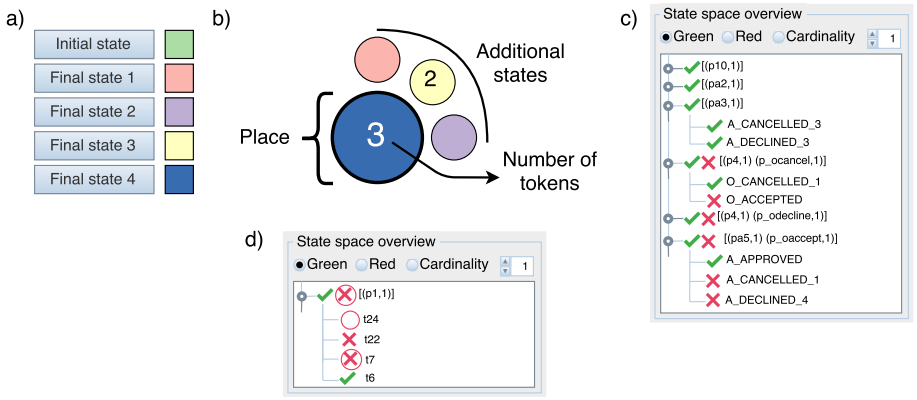

Fig. 4. Components of our tool: a) example of the component showing the initial and final states and their colors; b) four different final states that involve the same place; c) from all available states it is possible to reach a *Green* state, while from three of them only *Red* states are possible to reach; d) either *Green* or *Red* states can be reached. Within *Red* states, livelock, deadlock, and both scenarios are possible.

is useful because it gives a quick overview of the parts of the Petri net that belong to the *Green*/*Red*/*Orange* area. In order to do this, users can select what they want to visualize (places/transitions) and which area they want to explore (*Green*/*Red*/*Orange*). Once the user selects one of these options, the Petri net view highlights those elements which belong to the selected area. This feature relates to task T2.

3.3 States View

This component can be seen in Fig. 4 (c and d) and it enables the exploration of the most important states within the state space of the Petri net. Those states correspond to scenarios in which the process can lead from an *Orange* state to either a *Green* or *Red* end state. In this view, we only display the border cases, which are derived from the relevant transitions of the state space (see Fig. 1).

Our approach uses a two-level tree to visualize the different states in which the process experiences a problem. The first indicates the states in which an *Orange-to-Red* or *Orange-to-Green* scenario was found, and the second level indicates the transition that is involved in the detected scenarios. For each state, the user can find a variable number of transitions that could trigger a step in the state space leading from an *Orange* state to a *Green* state, or a *Red* state in which two situations can ultimately happen: deadlock or livelock.

In some cases, the algorithm that our tool uses detects a problem only because of the threshold that it uses. In those cases, this component indicates that by marking the problematic state with an asterisk. In this way, users can easily differentiate those states that are always problematic from those that are problematic because the algorithm did not continue exploring.

We use icons to give a quick overview of the scenarios found by the algorithm. They indicate the scenario for a specific state (aggregated view) and for a specific transition

(specific view). Therefore, the first level of the tree might display multiple icons indicating all the possible scenarios that can be reached through that state. Seven scenarios are possible: 1) reaching a *Green* state; 2) a *Red* state that eventually leads to a deadlock is reached; 3) a *Red* state that eventually leads to a livelock is reached; 4) a *Red* state from which a deadlock and livelock can be reached; 5) either a *Green* state or a *Red* deadlocking state can be reached; 6) either a *Green* or a *Red* livelocking state can be reached; and 7) either a *Green* or *Red* dead/live-locking state can be reached.

The nodes of the tree can be sorted by three different criteria. By clicking on the corresponding radio buttons, the view sorts the states by the criterion chosen by the user. Thus, users can sort states by the type of scenario that they represent (either *Green* or *Red*) and the cardinality of the states.

Next to the sorting functions there is a spinner, which is used to set the threshold used by the algorithm that computes the state space. By default, this parameter is set to 1. When the user interacts with the spinner, the tool recomputes the state space, partitions the recomputed state space into *Green*, *Red* and *Orange*, and recomputes all the relevant information related to them, such as runs or disjoint sets of states and transitions.

Users can interact with the nodes of the tree to explore the different scenarios. This way, the nodes can be selected to be displayed in the Petri net view. When a node from the first level of the tree is selected, the main view shows all the available scenarios for that specific state by coloring the nodes of the Petri net that are involved in that specific state. In Fig. 3 state $\{p\_ocancel,\ p4\}$ is selected. The places that define the selected state are colored blue, while the transitions that can be triggered leading the process to a *Green* or *Red* state are colored green and red.

Once users select a (set of) state(s), it is possible to interact with the main view to explore the behavior of the model. This is done by enabling users to click on the transitions that have been colored to show the paths that lead to the selected state, and the final marking reached by triggering that transition. This feature connects this view and the runs view, which is described below.

3.4 Runs View

This view helps users perform tasks T4 and T6. An example is shown in Fig. 3(3). Runs are displayed as disconnected graphs, which can be projected as paths in the Petri net view. When users select a state from the *states view*, this component shows the runs that lead to the chosen states. Then, two major interactions are provided: nodes hovering, and path selection. On the one hand, the first interaction aids users in linking nodes of the runs to nodes of the Petri net view. On the other hand, the second interaction assists in visualizing the path that goes from the initial state to the selected state directly on the Petri net.

Through these two interactions, users can detect states that share similar segments of path, helping users get insights into the problematic scenarios of the model.

There is a third interaction which links directly with the Petri net view. When the user clicks on a transition that is involved in the problematic scenario that the user is exploring, the Petri net view takes the run that leads to the ultimate state reachable from the current state, that is, a deadlock or livelock, and shows the path. Providing the

context in which the process ends up in a *Red* state helps the user to understand how the process led to that problem.

3.5 Implementation Details

Our tool is implemented as a plug-in within the ProM [24] framework using Java v6. Prefuse [10] is used to manage the graph structures and the visual properties of the visualization of the Petri net. The jBPT library [17] for Petri nets is used to compute the runs.

4 Use Cases

In this section we demonstrate how our tool can be used to assess process models. We focus on two models, which were designed by students who participated in the study developed in [29]. These models are in the format of Accepting Petri Nets (APNs), which already have information about the initial and final states of the model.

The first case exposes an example in which no problematic situations were found. Figure 5 summarizes this use case. It shows an overview of the Petri net and an area of interest in more detail. Clearly, there are no problematic states in the states view. However, we observe that some of the places are colored red (EXPLAIN WHAT IT MEANS) and some are not colored at all. The latter places do not appear in the state space, which means they are not reachable. The final state is also not colored, and therefore the process can never reach the final state. By going backwards in the Petri net, we can spot the origin of the problem. The source is in the second place, which corresponds to a choice. If the process goes through one of the possible transitions, it finishes in a deadlock (see the path highlighted in red).

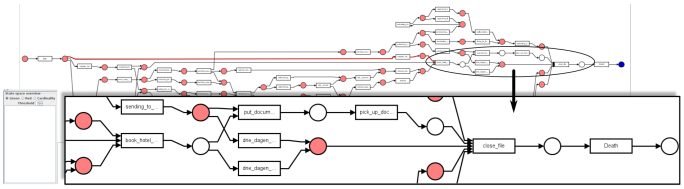

Fig. 5. Screenshots of the suage of the tool for a dataset.

The second use case depicts a process model in which several problematic scenarios have been found. Initially, we proceed by exploring the disjoint sets of places and transitions. We can easily spot some straightforward paths that lead to the final state as well as some paths that eventually finish in the *Red* area. Furthermore, we see some places and transitions that have not been colored. We now focus on those elements in order to explore what occurs there. By hovering on some of the places that have not been colored, we can see their labels. One interesting place is the one labeled as *Status7* since it is present in three of the problematic states shown in the states view. We pick

one of those states to explore the runs that lead to that problematic state. The tool shows two different runs. By looking at these, we can see that they share some of the initial steps in the process, but that they differ at a later point. Clicking a run visualizes the paths that those runs represent in the Petri net. If we compare the two runs in this way, we immediately see that they finish in the same problematic state, even tough they are execute different activities. We can also see where the deadlock occurs by clicking on the transition colored red. Fig. 6(b) shows the two paths (edges colored blue) for the two runs, as well as the deadlock that is reached (place colored red). The place colored blue indicates that the token must be consumed by firing the red transition.

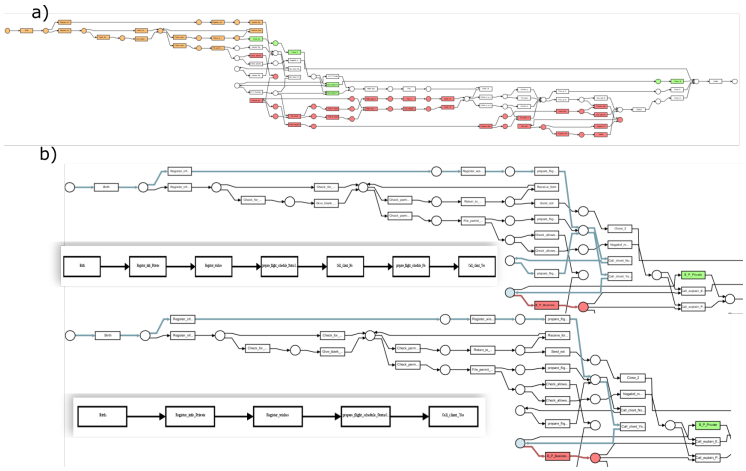

Fig. 6. Screenshots of the usage of the tool for a dataset.

5 Conclusion

We have presented PSVis, a tool to visually assess the soundness of a process model. We have formulated the most important analysis tasks, and have demonstrated the usage of our tool through the exploration of two process models. The first use case showed a simple scenario. With smooth interaction we discovered why the final marking was not reachable. The second use case represented a more complex example in which more actions were performed. Through different actions we observed different aspects of the process model such as deadlocks and common paths that led to them.

One of the main limitations of our tool is that it relies on the state space, which cannot not always be computed in reasonable time. Even though we have some workarounds (setting a threshold to limit the computation of branches), it may take a considerable amount of time to finish. We plan to study alternatives to compute the parts of the state space that are used in the analysis. One option might be to explore the state space incrementally by computing just portions of it.

In future work we aim to perform experiments to compare different approaches for displaying the runs since the current representation lacks of an explicit way to display loops and concurrency.

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 *lncs*, pages 407–426, Toulouse, France, June 1997. springer.
2. 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.
3. N.R. Adam, V. Atluri, and W.K. Huang. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems*, 10(2):131–158, March 1998.
4. R. Bergenthum, J. Desel, G. Juhás, and R. Lorenz. *Can I Execute My Scenario in Your Net? VipTool Tells You!*, pages 381–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
5. J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *lncs*, pages 157–170. springer, 2001.
6. J. Desel. *Validation of Process Models by Construction of Process Nets*, pages 110–128. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.
7. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
8. J. Desel, G. Juhás, R. Lorenz, and C. Neumair. Modelling and validation with viptool. In *International Conference on Business Process Management*, pages 380–389. Springer Berlin Heidelberg, 2003.
9. C.A. Ellis. Information control nets: A mathematical model of office information flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, USA, 1979. ACM Press.
10. J. Heer, S.K. Card, and J.A. Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM, 2005.
11. J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and prediction of errors in epcs of the sap reference model. *Data & Knowledge Engineering*, 64(1):312 – 329, 2008. Fourth International Conference on Business Process Management (BPM 2006)8th International Conference on Enterprise Information Systems (ICEIS' 2006)Four selected and extended papersThree selected and extended papers.
12. G. De Michelis, C. Ellis, and G. Memmi, editors. *Proceedings of the Second Workshop on Computer-Supported Cooperative Work, Petri nets and Related Formalisms*, Zaragoza, Spain, June 1994.
13. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
14. C. Ouyang, H.M.W. Verbeek, Wil M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming*, 67(2):162 – 198, 2007.
15. J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1981.
16. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962. In German.

17. A. Polyvyanyy and M. Weidlich. Towards a compendium of process technologies: The jbpt library for process model analysis. In *Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 106–113. Sun SITE Central Europe, 2013.
18. L. Popova-Zeugmann. Time petri nets. In *Time and Petri Nets*, pages 31–137. Springer, 2013.
19. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. springer, 1985.
20. W. Sadiq and M.E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000.
21. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.
22. W. M. P. van der Aalst. *Process Mining: Data science in Action*. Springer-Verlag, Berlin, Germany, 2016.
23. W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 2(2):182–192, March 2012.
24. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. pages 444–454. Springer, Berlin, Heidelberg, 2005.
25. B.F. van Dongen, M.H. Jansen-Vullers, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of the sap reference models using epc reduction, state-space analysis, and invariants. *Comput. Ind.*, 58(6):578–601, August 2007.
26. B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek. *Verification of EPCs: Using Reduction Rules and Petri Nets*, pages 372–386. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
27. K. van Hee, N. Sidorova, and M. Voorhoeve. *Generalised Soundness of Workflow Nets Is Decidable*, pages 197–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
28. H.M.W. Verbeek. *Verification of WF-nets*. Eindhoven University of Technology Eindhoven, the Netherlands, 2004.
29. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
30. H.M.W. Verbeek and M.T. Wynn. Verification. In A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors, *Modern Business Process Automation: YAWL and its Support Environment*, Database Management & Info Retrieval, chapter 20, pages 517–545. Springer, Berlin, Germany, 2010.
31. M.O. Ward. Multivariate data glyphs: Principles and practice. In *Handbook of data visualization*, pages 179–198. Springer, 2008.
32. M.T. Wynn, H.M.W. Verbeek, Wil M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Business process verification - finally a reality! *Business Proc. Manag. Journal*, 15(1):74–92, 2009.