Simulation of Place Transition Petri Nets in AtomPM

Maris Jukss

Abstract

AtomPM, next generation of model transformation tools, sibling of Atom3 provides a convenient and distributed way for model transformations and domain specific language engineering. One of the formalisms included with AtomPM is Petri Net modeling language for simple place transition nets. AtomPM does not provide the operational semantics for Petri Nets as a built-in feature. In the context of this project, model transformation scheme was developed to execute Petri Nets that were constructed inside AtomPM. In addition, reachability graph generation was introduced along with the feature to export Petri Net to PNML.

Keywords: Petri Nets, Operational Semantics, Reachability Graph, PNML, Execution

1. Introduction

AtomPM is evolving and is now capable of replacing some of the existing specialized modeling tools, like PIPE Bonet et al. (2007) for Petri Net manipulation and execution. However there was no built in functionality to provide operational semantics of Petri Net (PN) Petri (1973) or to perform the basic analysis such as reachability graph generation.

This project's main goal was to execute PNs created from built-in PN formalism. This was achieved by combination of model transformation rules and the action code for model manipulation. As a result several rules were developed that were combined into a single transformation file. To make things easier and more convenient, PN specific tool bar was developed that allows to compile PN and prepare PN for execution and analysis, reachability graph generation, and PNML Billington et al. (2003) export. Reachability

Email address: `maris.jukss@mail.mcgill.ca` ()

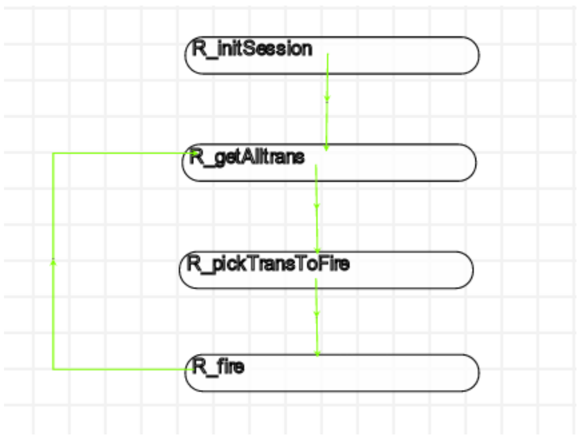

Figure 1: Rule scheduling for PN execution

Figure 2: Petri Net specific bar

graph generation is an important analysis tool presented in PIPE tool and is a valuable feature to have in any serious PN tool. The reachability graph is generated and then plotted using graphviz. We describe the related work in Section 2, and provide implementation details in Section 3. The result are demonstrated and analyzed in Section 4, and finally, we conclude this report in Section 5.

2. Related Work

PIPE tool, the flexible, yet lately unsupported and buggy tool provides many PN functionalities. I have envisioned this project to implement the features available inside that tool, however due to current limitations of AtomPM, it was impossible to conduct this task. For example, one can not affect the colors of fired transitions in this tool. The reachability graph is generated similar to PIPE tool. PN models that are presented were taken from Christensen and Petrucci (2000).

3. Implementation

- To implement PN execution, several rules were implemented and combined in one transformation. The scheduling of the rules in transformation is displayed in figure 1. Following the rules top to bottom, the first rule initializes the data structure that would hold all enabled transitions. The second rule would scan all transitions presented in PN model to determine if any transition is enabled. A transition is enabled if all incoming places provide enough tokens. Once the enabled transitions have been identified, the next rule would randomly choose the transition to fire. In fact, the next rule would perform the change in PN model concrete syntax and visualize the result of the token exchange during the transition firing. Due to the current limitations of AtomPM tool, the rules rely heavily on the action language of rules to achieve the execution of PN. The green lines in figure 1 display the control flow of rule execution. The path is taken after successful execution of a rule.

- In figure 2, the PN specific tool bar that was developed for this project is demonstrated. The first button serves two purposes. It can load the PN execution rule, thus eliminating the need to open the transformation to execute PN. It also prepares the server for the creation of the reachability graph. The second button implements creation of reachability graph. The output svg image of the reachability graph is stored in graph folder of AtomPM installation. The third button is not implemented but was meant to provide the import capability. The forth button saves the current model into PNML file in user's directory.

- Reachability graph generation is an implementation of a straight forward algorithm that is stack-based. Stack contains the states, that we need to explore while constructing the reachability graph. The incidence matrix is used to produce new states when transitions fire, and D-matrix is employed to determine whether any transition is enabled. The reachability graph generation was implemented using Python.

- Export to PNML feature was implemented using Javascript on the server side of AtomPM. We analyze json file representing the PN model and construct the PNML file. We then transfer all necessary attributes of nodes to receive meaningful PNML model. The results of conversion were verified using PNML viewer tool, PNMLview .

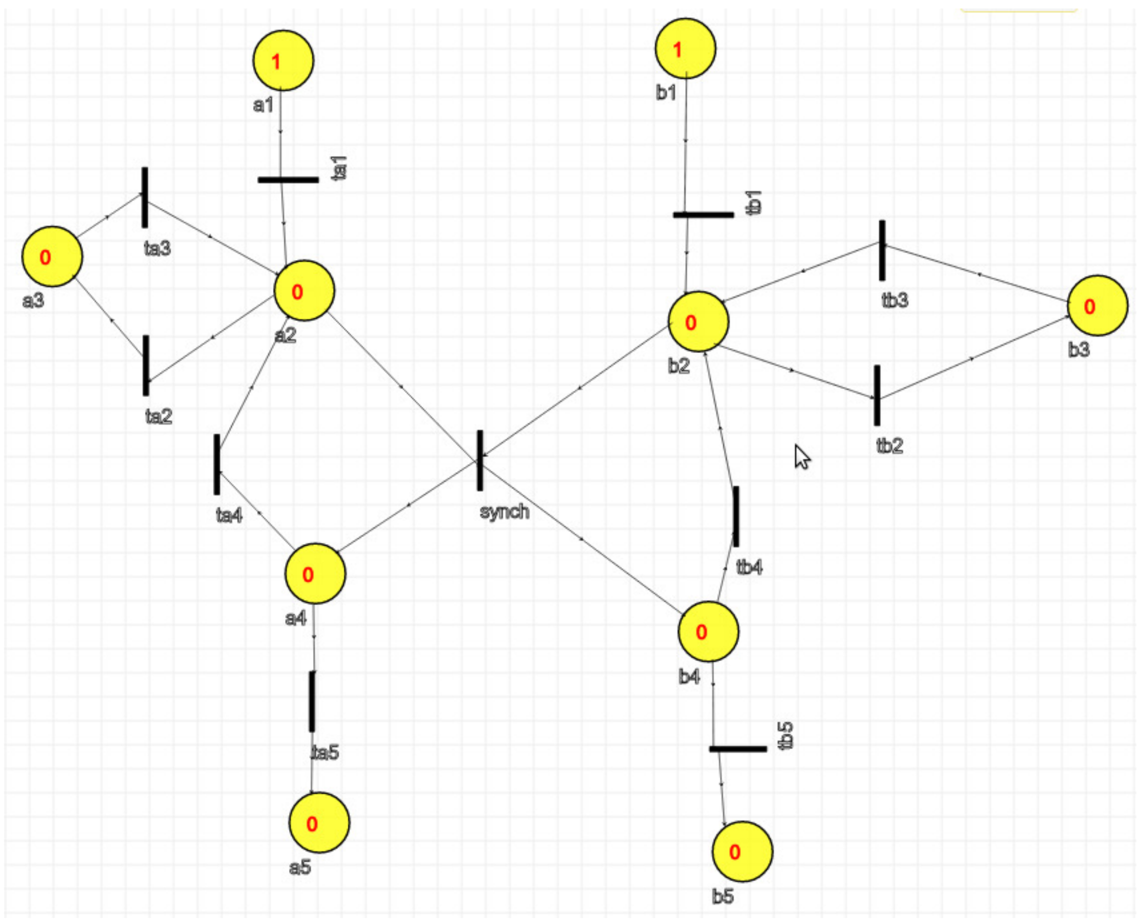

Figure 3: Petri Net used for testing reachability graph generation.

- PN execution, export to PNML, and reachability graph generation are tightly integrated. For example, after executing PN (which can also be paused and resumed) one can generate reachability graph based on current marking of the PN or export it into the PNML file. This is a very practical feature.

4. Results

Empirical analysis of PN execution is difficult to define and is not included in this report. However, it is worth noting that execution was observed to be

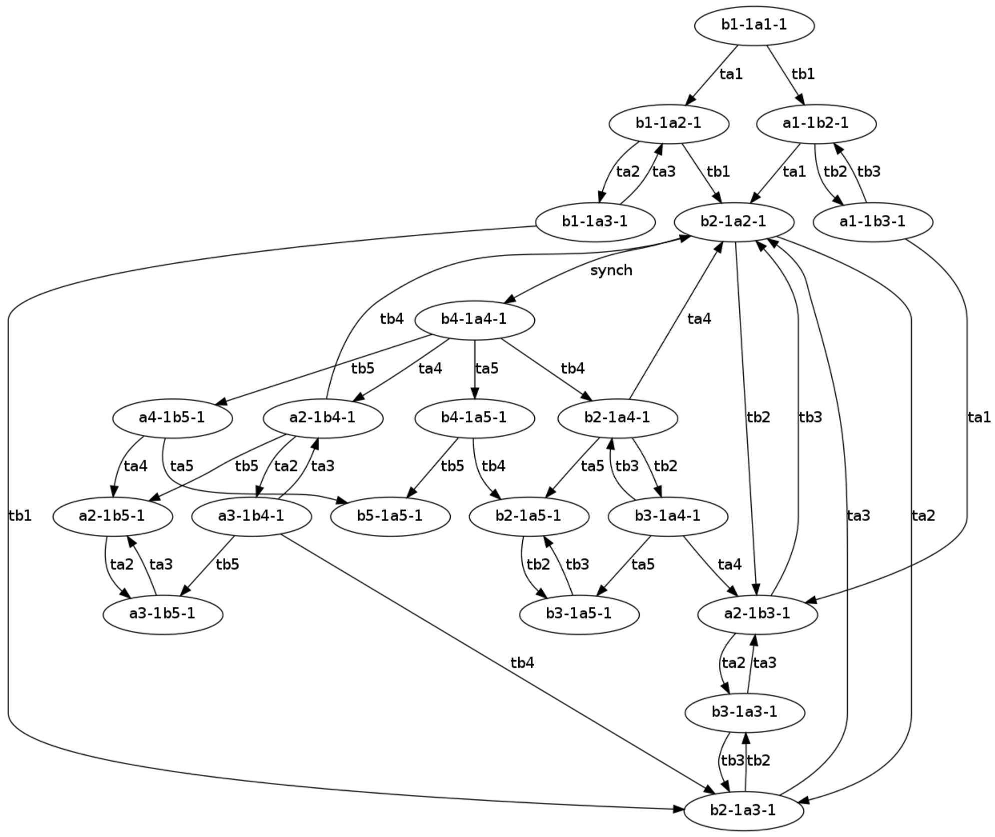

Figure 4: Reachability graph for figure 3.

Table 1: The results of constructing reachability graph for the benchmark model in figure 3. Time without plotting the actual graph.

M0	Time,seconds	Number of nodes	Number of edges
a1=1,b1=1	0.023551	21	43
a1=2,b1=2	0.563132	207	725
a1=3,b1=3	18.873297	1175	5311
a1=4,b1=4	359.258912	4790	25325

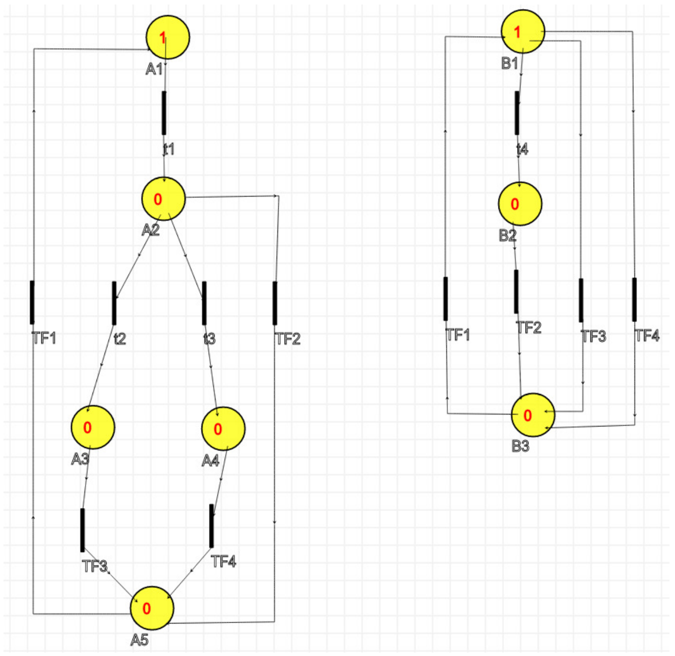

Figure 5: Model to demonstrate export to PNML.

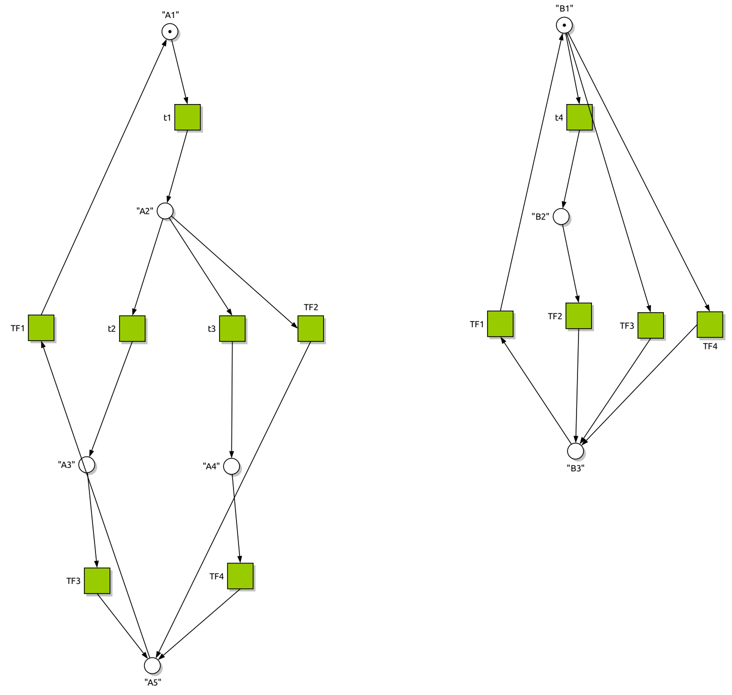

Figure 6: Resulting PNML from pnmlview.

non-deterministic in fashion, as expected. We can analyze the construction of reachability graph. Table 1 presents the performance analysis of reachability graph construction. As can be seen, the construction time dramatically increases due to the state space explosion phenomenon. Interestingly, PIPE tool did not generate the reachability graph for the initial marking of $M0 = (a1 = 2, b1 = 2)$ of equivalent net and onwards in complexity, while, the graphviz software was able to plot the this graph inside AtomPM. However, even on a powerful 16 GB machine, graphviz could not handle plotting the graph for initial marking of $M0 = (a1 = 4, b1 = 4)$. An example of a successfully generated reachability graph is shown in figure 4.

The export feature was tested on various models and performed well, except for the strange occurrence of misplaced transitions or places inside pnmlview tool. I presume that this is the result of tool's malfunction, perhaps due to its canvas being unable to accommodate the dimensions imported from AtomPM. Based on my observations, this incident occurs only for large models. Since I was not able to find another tool to display PNML file, I cannot conclude that the conversion is at fault. Figure 5 represents the PN model used to demonstrate PNML export feature. The final PNML file is displayed in figure 6.

5. Conclusions and Future Work

In this project, I successfully incorporated the PN tool bar with a functionality that would benefit PN users of AtomPM. The complexity of this project was the multi-layered architecture of AtomPM and the lack of full transformation language support to perform PN execution purely through graph rewriting. At first, I implemented the marking of enabled transitions using the attributed generic links. However, that approach appeared to be very slow. The action code approach is a necessity in the current implementation of AtomPM. In the future release of AtomPM, transformation language would provide more flexibility to transformation engineers. For the future work, one can envision the import of PNML files into AtomPM. The illumination of fired transitions would be very informative and helpful, however currently it is impossible to implement. At the moment, when transition fires, text of the name of transition toggles to new value and back to old value rapidly to indicate the firing.

References

Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M., Jun. 2003. The Petri Net Markup Language: Concepts, Technology, and Tools. In: Applications and Theory of Petri Nets 2003: 24th International Conference. Eindhoven, The Netherlands, pp. 1023–1024.
URL `http://www.springerlink.com/content/rp1dqtlmqr5q665b`

Bonet, P., Llado, C., Puijaner, R., Knottenbelt, W., Oct. 2007. Pipe v2.5.: a petri net tool for performance modelling. In: 23rd Latin American Conference on Informatics.

Christensen, S., Petrucci, L., 2000. Modular analysis of petri nets. Comput. J. 43 (3), 224–242.

Freek, W., Apr 2012. Pnmlview tool.
URL `http://www.vanwal.nl/pnmlview/`

Petri, C. A., 1973. Concepts of net theory. In: MFCS. Mathematical Institute of the Slovak Academy of Sciences, pp. 137–146.

Rapahael, M., Apr 2012. Atompm tool.
URL `http://msdl.cs.mcgill.ca/people/raphael/files/usersmanual.pdf`