Assignment 4
Production System Translational Semantics in AToMPM

Bentley James Oakes

1 Practical Information

The goal of this assignment is to build a rule-based transformation for executing the translational semantics of the production system modelling language in the visual modelling tool **AToMPM**.

The different parts of this assignment:

1. Build a transformation to generate a Petri net alongside the production system model, connected by traceability links.
2. Build a transformation which executes the Petri net and simultaneously visually updates the production system model.
3. Create two production system models that are representative for all the features in your language. Show a few steps of the execution of these transformations for these models, and create a short video for one.
4. Write a report that includes a clear explanation of your complete solution and the modelling choices you made.

This assignment should be completed in groups of two if possible, otherwise individually is permissible.

Submit your assignment as a zip file (report in pdf, commented syntax and semantics models, and example models) on Blackboard before **Tuesday, November 24th, 23:59h**. Contact Bentley Oakes (bentley.oakes@uantwerpen.be) if you have any issues.

2 Requirements

This section lists the requirements of the production system translation semantics transformations and the report. Make sure to test each requirement with test models!

2.1 Petri Net Generation

Implement a rule-based transformation that generates a well-formed Petri net modelfrom a production system model.

- Ensure that this transformation is complete. That is, a well-formed production system will produce a well-formed Petri net.
- The Petri Net formalism you use should be (or be based on) the `/Formalisms/PN` formalism. Feel free to create your own version by extending the original to add positioning, snapping, as needed. Don't forget to hand in your custom Petri Net formalism as well.
- Do not remove your production system during this transformation. It is instead left untouched, and *Places*, *Transitions*, and arcs created for each element in the Petri net.
  - Note that inhibitor arcs cannot be used! This is due to the analysis tool in the next assignment.
- Note that *Places* and *Transitions* must be uniquely named in the Petri net model. This is to allow for the analysis in the next assignment.
- Traceability links must be created from the production system element and the created *Places* and *Transitions*, such that the transformation in Section 2.2 can operate on both models.
- Simplifications:
  - Assume that there are a maximum of two operators present in the production system.
  - Assume that operators stay a maximum of one step at every machine.
  - Assume all possibilities of the *Inspection* machine have equal weight. That is, do not model the probabilities.
  - Layout considerations do not have to be considered in this transformation. That is, you may assume that the user will manually move Petri net elements to an appropriate location as they are created.
- **Warning:** There is a bug in AToMPM where the action code on the *PtoT* or *TtoP* associations in the Petri Net formalism is not reset to `result = True` or `result = getAttr()` when it is placed in the LHS, RHS, or NAC. You will have to **manually change the action code for all these created associations**. Apologies for the inconvenience.
  - Note that an exception has been added to the transformation server, such that a transformation cannot be loaded with rules with any unreset action code. Otherwise, rules would silently fail.
- Figure 1 shows the schedule for a production system and the generated Petri net.

2.2 Executing the Petri Net

Implement a rule-based transformation that executes the Petri net, and simultaneously visually updates the production system model.

- Base this transformation on the ones in the */Formalisms/PN* folder.
- This basic Petri Net execution transformation detects transitions, places a pivot on the transition to fire, consumes attached tokens, and then produces tokens.
- Create your own version by adding rules which match on the transition to fire (using the pivot), and performs the visual update of the connected production system elements.
  - One way to select the rule to fire could be through an encoding scheme on the *Transition* name.
  - Example: The *R_CubeArrival* rule has a pattern which matches on the ending token of the *T_Cubearr_CubeArrival* transition. In this way, the transition "type" indicates which production system elements should be updated.
- Run-time info (items accepted, number of timesteps, operators having moved, etc.) must be represented in the Petri net by *Places*. These will be used in the following assignment for analysis.
- Simplifications:
  - If needed, you may allow items to move multiple times in a time-step. Operators must still only move once per time-step.
- As a heads-up, in the next assignment the order of transition firings will be specified. This will be done through a string containing the list of transitions to fire in the model, similar to the token parsing in the FSA example. Therefore, ensure that your solution can be extended to handle this.

3 Report

There are a number of requirements for the report. Above all, the **marker must be able to read the report and have a clear understanding of all aspects of the assignment, without having to investigate the model files**.

Specifically, the report must contain:

- A brief outline of how the rules, transformations, and example models meet the requirements of the assignment
- A discussion of any interesting decisions made.

- A discussion of possible improvements to the rules and transformation syntax.
- Two example production systems.
- For each production system, show:
  - Diagrams of at least a few steps of the production system during the generation and execution transformations. Highlight the items being assembled, destroyed, fixed, etc. and explain the behaviours you are showing.
  - These diagrams and your explanations must convince the reader that your transformation implements the translational semantics.
  - Note that (textual) traces are not required for this assignment, as the visual representation of the production system should show the desired behaviour.
- Choose one production system and produce a short screen recording of the Petri net execution transformation running and showing interesting behaviour.
  - This video should be uploaded to YouTube and the link placed in the report. Note that it should be unlisted, so it cannot be found except for the link.
  - A short description should be provided below the video, but no captions/voiceover/editing is required.

4 Useful Links and Tips

- AToMPM main page: `https://atompm.github.io/`
- Download and code: `https://github.com/AToMPM/atompm`
- Documentation: `https://atompm.readthedocs.io/en/latest/`

Acknowledgements

Based on an earlier assignment by Simon Van Mierlo.
Icon authors from `www.flaticon.com`:

- Cylinder - `https://www.flaticon.com/authors/kiranshastry`
- Cube - `https://www.flaticon.com/authors/smashicons`
- Belts, Machine, Inspector, Incinerator, Receiver - `https://www.flaticon.com/authors/freepik`
- Arrival Machine - `https://www.flaticon.com/authors/catalin-fertu`
- Fixer - `https://www.flaticon.com/authors/srip`

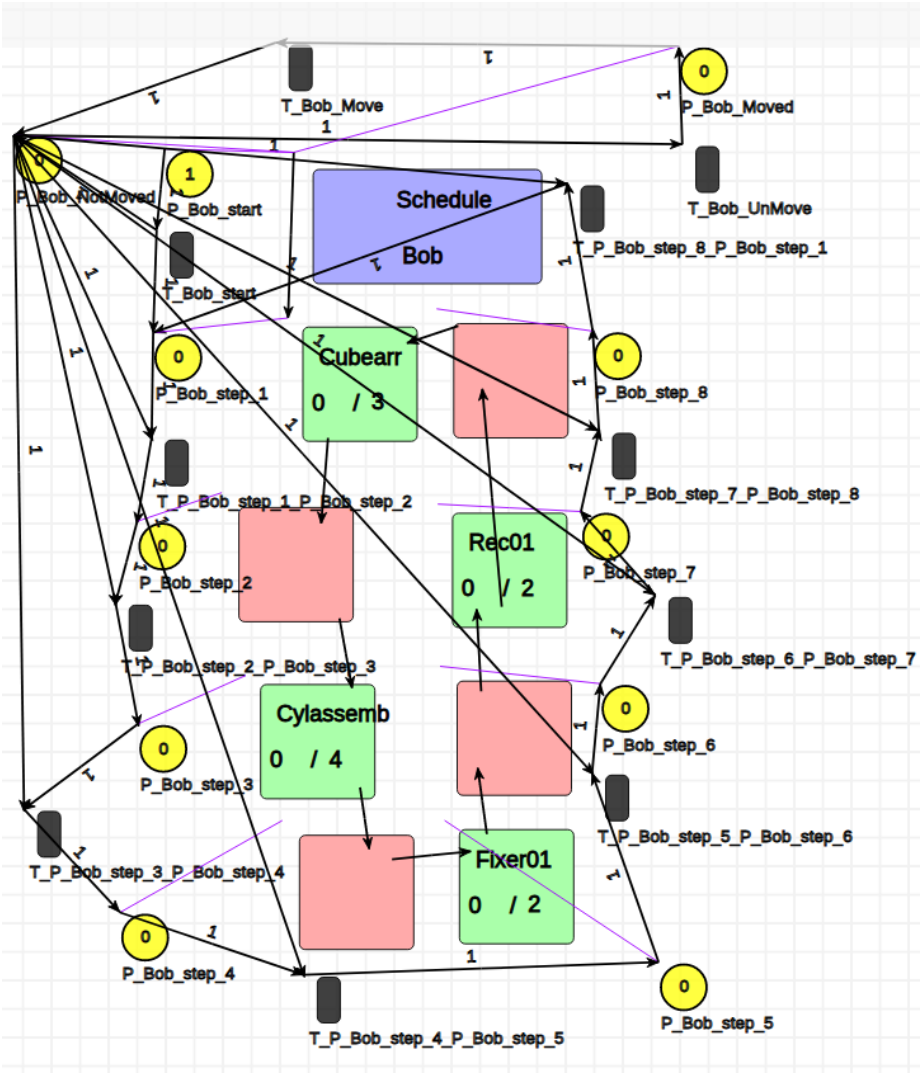

Figure 1: An example of a schedule and the created Petri Net.