Assignment 3
Operational Semantics in AToMPM

Randy Paredis
randy.paredis@uantwerpen.be

1 Practical Information

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

The different parts of this assignment:

1. Build the RAMified domain-specific language for the production system.
2. Build the transformation rules. These rules must visually update the production system model according to the operational semantics defined below.
   - The formalism for this part will be `/Formalisms/__Transformations__/TransformationRule/TranformationRule`
   - You can also quickly create new rules with the *create new rule* button.
3. Schedule the transformation rules
   - The formalism for this part will be `/Formalisms/__Transformations__/Transformation/MoTiF`
   - You can also quickly create new transformations with the *create new transformation* button.
4. Create two production system models that are representative for all the features in your language. Show a few steps of the transformation execution on these models, and create a short video (see below).
5. Write a report that includes a clear explanation of your complete solution and the modelling choices you made, as well as an explanation of your testing process. Also mention possible difficulties you encountered during the assignment, and how you solved them. Don't forget to mention all team members and their student IDs!

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 + model files) on Blackboard before **23 November 2021, 23:59h**[1]. If you work in a group, only *one* person needs to submit the zip file, while all others *only* submit the report. Contact Randy Paredis if you experience any issues.

2 Requirements

This section lists the requirements of the production system operational semantics language. Make sure to test each requirement with test models!

2.1 Abstract/Concrete Syntax Modification

Important: *Feel free to modify your abstract syntax to be able to implement these operational semantics as transformations, or to make your solution more elegant. Your abstract syntax and operational semantics will not be compared against the earlier assignments.*

Make sure you clearly specify **the choices you made**. Explain why you changed the abstract syntax, if there another way but is too awkward/time-consuming, etc. This assignment is not about creating the perfect abstract syntax/operational semantics, but instead about being able to reason about (and discuss in the report) how information should be divided into these different syntax/semantics models.

In particular, sometimes it is difficult to represent operational semantics using rules. Your report should discuss the effects of the choices you made such as any non-determinism caused by rule matching.

2.2 Operational Semantics

This part of the assignment is mostly copied from Assignment 1 for your convenience.

In this part of the assignment, the semantics of the production system will be modelled, including the *Operators*, *Items*, and *Machines*. The goal is for the *Operators* to move between *Machines* and operate them, such that *Items* are assembled, inspected, fixed, received, or destroyed.

The specific requirements are:

- The simulation is broken up into a number of discrete steps. In each step, the *Machines* are operated if *Items* and *Operators* are present, the *Operators* are moved if needed, and then the *Items* are moved (if possible).
- In the initial step, all *Operators* are placed at their start *Machine*. If an *Operator* is scheduled to move to an occupied *Machine*, they must wait until the other *Operator* is finished.

[1] Beware that BlackBoard's clock may differ slightly from yours.

- An *Item* is allowed to move to the next segment if no *Item* is present on that *Segment*, and the *Item* is not in an *Assembler* waiting for the other shape to arrive. The *Join* is excempted for this rule, as it will allow at most two *Items* to be located on the segment.
- An *Arrival* cannot produce a new *Item* if an *Item* is already on that *Segment*.
- When the *Assembler* operates, both *Items* are removed, and replaced with an *AssembledItem*.
- At a *Split*, 50% of the *Items* will move to the first output and 50% to the second one. This can easily be ensured by having the *Split* consecutively toggle between both outputs. For instance, the first item will be outputted to the left, the second one to the right, the third one to the left again, etc...
- At a *Join*, the *Items* are ordered in a *first-in-first-out* order.
- At an *Inspector*, the correctness of an *Item* is determined as an integer in the range of $[0, 100]$. This value determines how the *Item* will be handled/outputted. The chance of the *Item* being accepted (placed on the accepted belt) is 70%. The chance of requiring fixing is 20%. The chance of the *Item* requiring destruction is 10%.
- The simulation continues until some number of *AssembledItems* are accepted. Set this parameter such that the traces are long enough to show interesting behaviour.
- Think carefully about what information should be stored in the abstract syntax of the model, and which information is relevant to the simulator. You are free to choose how much information is stored in your model, as long as all information is useful in some way.

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. I.e., your model files will only be used as a support for your report, not the other way around!

Specifically, the report must contain:

- A brief outline of how the rules, transformations, and example models meet the requirements of the assignment.
  - This may include metamodels, diagrams, (pseudo-)code, etc. as needed to provide the essential details of the assignment.

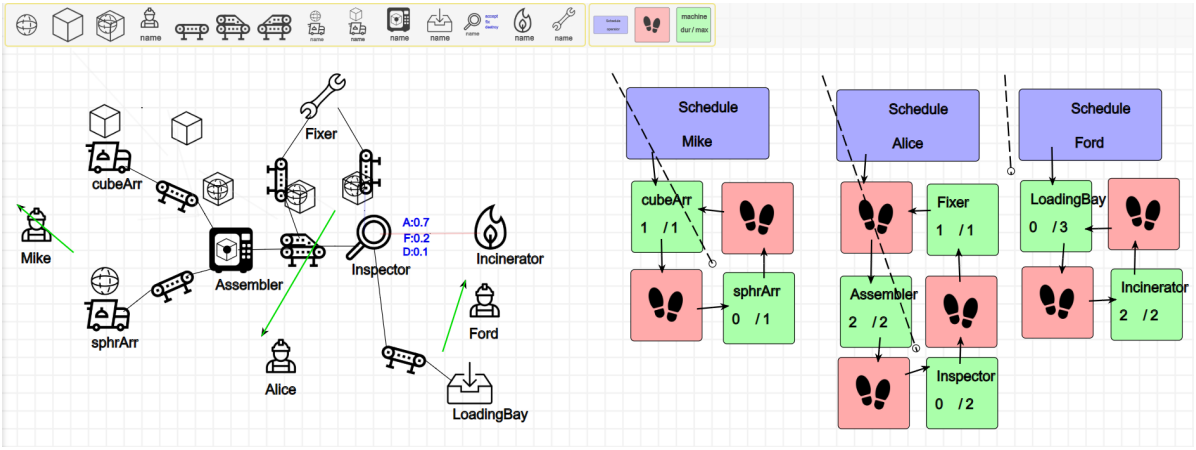

Figure 1: An example production system in AToMPM during the simulation.

- 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:
  - A figure of the production system within AToMPM.
  - Diagrams of at least a few steps of the production system as it is transformed. 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 operational 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 transformation running and showing interesting behaviour.
  - This video should not be submitted with your assignment (due to a large file size), but a link to where your video can be watched should be placed in your report. Note: this is not a download link!
  - For instance, you can upload it to YouTube, Vimeo, Google Drive, Dropbox,... Note that it should be unlisted, so it cannot be found, except when using the link.

  - A short description could be provided below the video, but no captions/voiceover/editing is required.
  - You can use OBS (https://obsproject.com/) or any other screen recording software.
  - Example: https://msdl.uantwerpen.be/cloud/public/5cbdeb

- As in the second assignment, your solution should be more visually pleasing than the figure and video shown here. Please spend time introducing proper sizing, colouring, and placement of elements to make a clear visualization.

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/
- For moving elements, there is an example rule in `Formalisms/RaceCar`.
- You are allowed to create a third run-time language for storing run-time information. This is, however, not required.
- If you change the abstract or concrete syntax for a language, you may need to delete and recreate all elements that you changed. Thus it is **strongly recommended that you work out the rules and languages on paper first**, before creating rules in AToMPM.
- If a rule is failing:
  - Double-check the labels in the LHS, RHS, and actions
  - Labels in the action must be strings. For example, `setAttr('position', [0, 0], '3')`. Note that this last argument is a string, not an integer!
  - If problems persist, try deleting the element and recreating it.

Acknowledgements

Based on an earlier assignment by Bentley Oakes.

Icon authors from www.flaticon.com:

- Sphere - https://www.flaticon.com/authors/good-ware
- Cube, Receiver - https://www.flaticon.com/authors/smashicons

- Belts, Machine, Inspector, Incinerator - https://www.flaticon.com/authors/freepik
- Assembler - https://www.flaticon.com/authors/catalin-fertu
- Fixer - https://www.flaticon.com/authors/srip
- Walk - https://www.flaticon.com/authors/vitaly-gorbachev