Assignment 4
Translational Semantics in AToMPM

Joeri Exelmans
joeri.exelmans@uantwerpen.be

1 Practical Information

In this assignment, we implement a translational semantics for our Traffic DSL, again by means of rule-based model transformations, still using **AToMPM**. We map our Traffic DSL onto **Petri-Nets**.

Overview of this assignment:

1. Build a set of transformation rules and a transformation schedule to generate a Petri-Net alongside the traffic network, connected by traceability links.
2. Build a transformation schedule that executes a Petri-Net transition, and updates the traffic network accordingly.
3. Record a video showing (1) generation of Petri-Net, and (2) execution of the Petri-Net.
4. Write a report.

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 + everything needed to run your solution in AToMPM: your AS, CS definition, all transformation rule models and your schedule model) on Blackboard before **12 December 2023, 23:59h**[1]. If you work in a group, only *one* person submits the ZIP file, and the other person *only* submits a text file containing the name of the partner. Contact Joeri Exelmans if you experience any issues.

2 Requirements

2.1 AS/CS modification(s)

The requirements for AS and CS remain the same as in the previous assignments. At this point, it is unlikely that you will have to make changes to your AS/CS. Nevertheless, if you make changes to AS/CS, please mention this in your report.

[1] Blackboard's clock

Remember: Every time you make a change to your AS or CS, have to:

- Re-compile AS and/or CS
- Re-create all your models
- RAMify again
- Re-create all your transformation rules
- (The transformation schedule will not have to be re-created.)

2.2 Petri-Net Generation

Implement a rule-based transformation that generates a Petri-Net model from a traffic network model.

- Use the 'PNS' (Petri-Net + Schedule) formalism (included in ZIP file, see end of document).
- The behavior of the generated Petri-Net must match the behavior of the original traffic network, as if it were executed via the operational semantics (from previous assignment).
  - You also translate the traffic network "Schedule" to a Petri-Net-schedule (i.e., a sequence of Petri-Net transitions to fire).
- You must create traceability links from elements of your traffic network model to your generated Petri-Net model. There are two reasons for this:
  1. You can use traceability links to generate your Petri-Net incrementally (e.g., first generate the places, then the transitions). With traceability links, you can match a partially generated Petri-Net.
  2. To keep your traffic network in sync with your Petri-Net, during Petri-Net execution.

  Create traceability links as follows:
  - When creating a model transformation rule, load the pattern metamodel of `/Formalisms/GenericGraph`.
  - Now, you can create traceability links between any elements of any formalism(s).
  - Figure 1 shows an example of a rule that uses traceability links. This rule is included in the examples ZIP file.

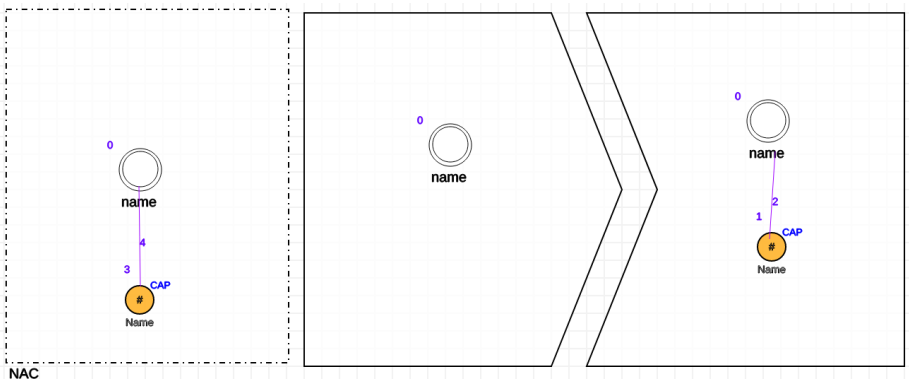

Figure 1: A transformation rule that creates a (capacity constrained) Petri-Net place for an FSA state, connected via a traceability link. The NAC ensures that only one Petri-Net place can be created per FSA state.

> **Beware:** Even though traceability links do not show an arrowheads, ***their direction matters!!*** All links in AToMPM have a direction. To save yourself a headache, always consistently create traceability links in the same direction, for instance, *from* your traffic network and *to* your Petri-Net.

- *Transitions* must be uniquely named in the generated Petri-Net model.
- Layout does not have to be considered in the transformations. That is, you may assume that the user will manually move Petri-Net elements to an appropriate location after they are created.

2.3 Petri-Net Execution

You will now implement a transformation that executes a Petri-Net transition, and updates the state of your traffic network to reflect the changes in the Petri-Net. For instance, if a RoadSegment is represented by a Petri-Net place, and a Car is represented by a token, then if a token is removed from a place, then a Car should also disappear from the corresponding RoadSegment.

It is not your job to implement an operational semantics for Petri-Nets from scratch. The 'PNS' formalism already includes a transformation schedule that executes one Petri-Net transition. You can 'call' this transformation schedule from your own schedule, using a **CRule**.

3 Screen recording

Record a video of Petri-Net generation and its subsequent execution, on a single traffic network model, that contains all features of your language.

4 Report

You will write a report, containing:

- A (brief) explanation of your workflow, and motivations for decisions made.
- An overview of your solution:
  - A screenshot of your abstract syntax meta-model, even if it is identical to assignment 2. I will need this to understand the transformation rules.
  - A screenshot of both transformation schedules (PN generation and execution), and an explanation of how it works (what are the steps, etc).
  - A screenshot of every model transformation rule, complemented with all the (Python) code written (pre- and post-conditions), and a short explanation of what the rule does.
- A link to the screen recording.

5 Grading

- (45 %) Petri-Net generation:
  - Transformation rules
  - Transformation schedule
- (30 %) Petri-Net execution
  - Transformation rules
  - Transformation schedule
- (25 %) Report
  - Explains workflow and difficulties encountered?
  - Can I easily understand your solution?
  - Video

6 Useful Links and Tips

- Download the starting point ZIP file: http://msdl.uantwerpen.be/people/hv/teaching/MSBDesign/examples/TransSemStartingPoint.zip, which includes:
  - 'PNS' formalism: Petri-Nets with Schedule. Also includes capacity-constrained places.
  - Example translational semantics for 'MyFSA' formalism. Note: the example is incomplete and only generates Petri-Net places, not the transitions! (The full example would be giving away too much of the solution...)
- Main resources:
  - AToMPM model transformation tutorial: https://atompm.readthedocs.io/en/latest/modelling_transformation.html
- Use AToMPM version 0.10.0rc3. (Version 0.9.0 is fine but doesn't work with Python >= 3.10)
  - Docker container: https://github.com/AToMPM/atompm/pkgs/container/atompm
  - Linux/Mac users can download the source code: https://github.com/AToMPM/atompm/archive/refs/tags/v0.10.0rc3.tar.gz
    - Install NodeJS and NPM
    - Then run `npm i` in the source directory to install all JavaScript dependencies.
    - Install the following Python packages:
      - python-socketio
      - igraph
      - requests
    - Then, run `node httpwsd.js`
    - In another terminal, start the Python model transformation backend: `python3 mt/main.py`
    - Navigate to http://localhost:8124/atompm
  - Windows users can try the "portable" version, which includes all dependencies: https://github.com/AToMPM/atompm/releases/download/v0.10.0rc3/atompm-portable.zip
- AToMPM peculiarities:
  - If a rule does not use a NAC, then **delete the entire NAC block!**
  - The default values of *link* attributes in NAC/LHS (`result = True`) or RHS (`result = getAttr()`) are not set correctly. You should manually set them correctly.

- Last year, we had some problems with the use of SRule. Use ARule instead. QRule works fine. FRule also seems to work.
- After executing a transformation, if you want to execute a different transformation, open your model again in a new window first.
- Make sure all elements in your NAC, LHS and RHS are actually contained by the NAC/LHS/RHS shape. To check if everything is properly contained, move the NAC/LHS/RHS shape a little bit, and see if the contained elements also move.
- One type of bug that can be hard to track down is when you set the default value of an integer attribute to `"0"` (which is a JSON string literal), but it should be `0` (i.e., JSON number literal). Same for booleans.
  - Supported datatypes (for attributes): https://atompm.readthedocs.io/en/latest/new_language.html#attributes
  - e.g., to create a list of integers, use `list<int>`
- Model transformation conditions and actions are written in **Python**!
  - You can debug your rules by using print()-statements. They will be printed in the terminal that runs the `mt/main.py` script.
  - Please try to do a bit of debugging before you tell me "it's not working".
- Expect some crashes, so save often!
  - If AToMPM crashes, make sure to kill all the `node` processes before starting it again!

Acknowledgements

Based on an earlier assignments by Randy Paredis and Bentley Oakes.