Assignment 3
Operational Semantics in AToMPM

Joeri Exelmans
joeri.exelmans@uantwerpen.be

1 Practical Information

In this assignment, we create an operational semantics for our Traffic DSL by means of rule-based model transformations, in **AToMPM**. The starting point for this assignment is the AToMPM formalism (Abstract Syntax (AS) + Concrete Syntax (CS)) you created in the previous assignment.

The different parts of this assignment:

1. RAMify your traffic DSL, resulting in a new pattern language for your DSL.
   (a) Use the *compile an abstract syntax model (...) into a pattern metamodel* button.
2. Build the **transformation rules**, which update your traffic models.
   - Create a new rule with the *create new rule* button. Then load your pattern language using the *load a pattern metamodel* button.
3. Create a **transformation schedule** (not to be confused with the 'Schedule' object that is part of our DSL) that specifies in which order your transformation rules should fire.
   - Create a schedule with the *create new transformation* button.
4. Create a **conforming model** (according to constraints of previous assignments), that is sufficiently interesting (see below for requirements), that demonstrates the operational semantics. Record a video showing the transformations happening.
5. 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 **28 November 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.

Note: You have three weeks to complete this assignment. However, 19-24 November I will be attending a workshop abroad, and will have very little time to help you out. So start early!

2 Requirements

This section lists the requirements of the traffic DSL.

2.1 AS/CS modification(s)

The requirements for AS and CS remain the same as in the previous assignment. However, it is possible that you want to make some small changes to your AS (and therefore also CS). For instance, you could change the inheritance relations between your classes a bit, to allow for fewer and/or simpler transformation rules. Make sure you write down any changes made (so you can mention them in your report).

Tip: 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.)

Therefore, it is recommended to create all transformation rules and a transformation schedule **on paper** first!

[1] Blackboard's clock

Tip: There are at least two ways to implement your Schedule class:

1. Connections have name-attributes. Your Schedule is a list of Connection-names. With each step (operational semantics), you pop a Connection from the Schedule.

2. Your Schedule is a linked list of Connection objects. You have a Schedule object that points to the head of this list. With each step, head is updated to point to the next Connection. To make this work, a Connection must be an Object (instance of a Class), rather than a Link (instance of an Association).

2.2 Operational Semantics

You must create a transformation schedule (consisting of a bunch of rules) that executes **one step**. The meaning of a 'step' remains identical to assignment 1 (MetaDepth):

> *During a step, one car may move along a Connection from one elements to another. A move is only possible if: (1) the previous element has a Car available, and (2) the next element has enough remaining capacity.* Availability *is defined as follows:*
>
> - ***Generator*** *Always has a car available.*
> - ***Collector*** *Never has a car available.*
> - ***RoadSegment, Split, Join*** *have a car available if there is currently a car on it.*
>
> Capacity *is defined as follows:*
>
> - ***Generator*** *Never has capacity (capacity 0).*
> - ***Collector*** *Always has capacity (infinite capacity).*
> - ***RoadSegment, Split, Join*** *have capacity 1. Only if there is currently no car on it, there is remaining capacity.*
>
> *The schedule defines a sequence of Connections along which cars will be moved. The step function always pops the* last *Connection from the Schedule, and attempts to move a car along it. If the Schedule is empty, a step has no effect.*

Put all rules in the `/Formalisms/<YourTrafficDSL>/OperationalSemantics` directory. This directory already contains an empty schedule `T_OperationalSemantics.model`. You can add your schedule to this file.

Tip: To move a Car along a Connection, you can remove the Car from the source segment and place the Car on the target segment in two different rules (of course, after having checked that a move is possible).

Tip: To give you some clue, my own solution consists of 4 Q-Rules and 3 A-Rules.

3 Example model

To test your transformation rules, you probably want to begin testing them on small models first, and incrementally work your way up to more complex ones. You do not have to submit these small models.

However, as part of your solution, you must create and submit **one example model** that demonstrates the operational semantics. This model must be sufficiently interesting:

- It must contain all element types: Generator, RoadSegment, Split, Join, Collector.
- It must contain at least 10 elements total.
- The schedule must initially contain at least 10 steps (Connections).

Create a screen recording of the execution of the operational semantics and include a link to it (YouTube, Vimeo, Google Drive, Dropbox, ...) in your report. You can use OBS (https://obsproject.com/) or any other screen recording software.

4 Report

You will write a report, containing:

- An 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 your model transformation schedule, 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 a screen recording of the simulation of your example traffic model.

About 1-2 pages of text (not including figures, code, screenshots) will do.

5 Grading

- (40 %) Transformation rules, with correct use of:
  - NAC
  - LHS
  - RHS
  - Python pre- and post-conditions
  - Abstract classes in NAC, LHS, RHS
- (30 %) Transformation schedule
  - Correctness of schedule (and rules)
- (30 %) Report
  - Explains workflow and difficulties encountered?
  - Can I easily understand your solution?
  - Video

6 Useful Links and Tips

- Main resources:
  - AToMPM model transformation tutorial: https://atompm.readthedocs.io/en/latest/modelling_transformation.html
  - FSA semantics example: http://msdl.uantwerpen.be/people/hv/teaching/MSBDesign/examples/MyFSA-with-semantics.zip
- 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 Model Transformation tutorial (covers everything): https://atompm.readthedocs.io/en/latest/modelling_transformation.html
    - 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>`
        * 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.

- AToMPM peculiarities:
    - If a rule does not use a NAC, then **delete the entire NAC block!**
    - Last year, we had some problems with the use of SRule. Use ARule instead. It is possible to solve this assignment with only ARule and QRule.
    - After executing a transformation, if you want to execute a different transformation, open your model again in a new window first.
    - AS constraints (previous assignment) are written in JavaScript. 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.
    - 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.