Assignment 5
Code Generation

Simon Van Mierlo
simon.vanmierlo@uantwerpen.be

1 Practical Information

The goal of this assignment is to generate code for two different platforms. The first is CPNTools, to do analysis on the Petrinet models resulting from the translational semantics of the previous exercise. The second is Python, by generating code that will actually create your modelled railway network and all its elements. For this, a Python framework is provided.

Write a report that includes a clear explanation of your complete solution, 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. You will have to complete this assignment in groups of 2. Submit your assignment (report in pdf, abstract and concrete syntax definition, example models, all rule-based model transformations, and code generators) on Blackboard before Monday 5 December, 14:00h.

Contact Simon Van Mierlo (`simon.vanmierlo@uantwerpen.be`) if you have a problem.

2 Prerequisites

Download this file: `http://msdl.cs.mcgill.ca/people/hv/teaching/MSBDesign/exported_to_md.zip` and unzip it in your AToMPM folder. Overwrite the default 'exported_to_md' that is already present. Also copy/paste your version of metaDepth.jar in the 'exported_to_md' folder.

3 Requirements

This section lists the requirements for the code generation assignment. They are split into two sections: one for CPNTools, and one for Python. Make sure to provide test models for each part, demonstrating the functionality you implemented!

3.1 CPNTools

For this part, three steps are involved:

1. Create realistic examples of railway networks on which you want to perform analysis. Translate your models to Petrinets (using the model transformation of the previous assignment), and export the generated Petrinets models to metaDepth (using the provided exporter).

2. Write an exporter using the Epsilon Generation Language (EGL) that exports the Petrinets model from metaDepth to CPNTools.

3. Perform analysis on the CPNTools model. Use the analysis to prove whether, for a model with two trains, there is a way to schedule these trains fully independently. This means that they will, on their way from the start to the end station, never enter a segment that was entered by the other train.

3.2 Python

This task also consists of three steps:

1. Export the railway metamodel and your example model(s) to metaDepth (using the provided exporter).

2. Write an exporter using the Epsilon Generation Language (EGL) that exports the railway model from metaDepth to Python.

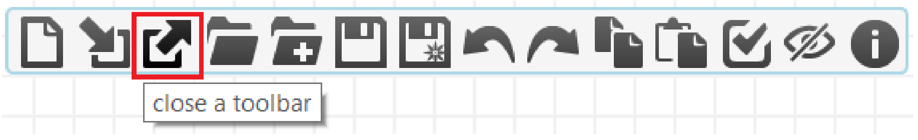
3. Run the program! A visual interface will allow you to play the role of the "control room".

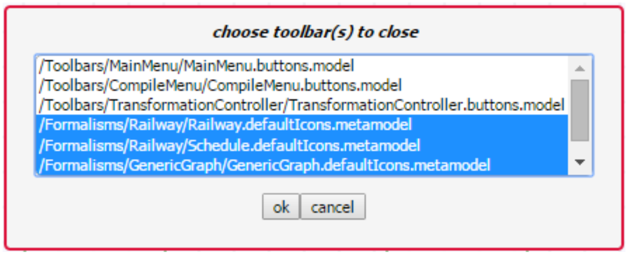
4 Manual

4.1 CPNTools

1. Generate your Petrinet model using your rule-based model transformation for denotational semantics.

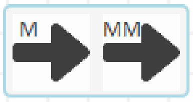
2. Remove all traceability links and railway networks elements by closing the respective toolbars.



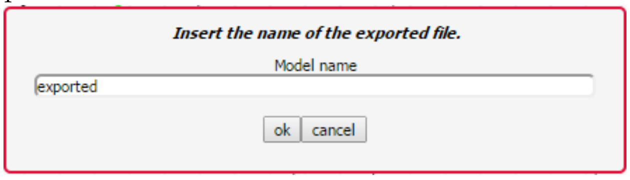


/Toolbars/MainMenu/MainMenu.buttons.model
/Toolbars/CompileMenu/CompileMenu.buttons.model
/Toolbars/TransformationController/TransformationController.buttons.model
/Formalisms/Railway/Railway.defaultIcons.metamodel
/Formalisms/Railway/Schedule.defaultIcons.metamodel
/Formalisms/GenericGraph/GenericGraph.defaultIcons.metamodel

3. Load the MetaDepth toolbar (inside of the */Toolbars/MetaDepth/* folder, it is called *Export.buttons.model*). This toolbar has two buttons: one for exporting models, and one for exporting metamodels. For this part of the exercise, you will normally only need the first one, as the exported PN metamodel is already present in the "exported_to_md" folder.



4. Click on the button for exporting models, and leave the name alone (the exported model should be called "exported.mdepth"). Click OK. This will generate a file with name "exported.mdepth" in the "exported_to_md" folder.



exported

5. Write your EGL code in the file "generate_cpntools.egl" (initially contains only necessary global XML tags). This file should generate a valid CPNTools model. For more information on CPNTools, see the "Useful Links" section.

6. **Tips for generating your CPNTools file**:

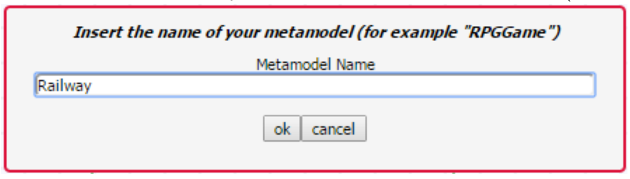
   - We do not use color in our Petrinets, so the type of tokens that can appear in your places is **UNIT**, and a token is represented by the value **()**.
   - To get an idea of the format used by CPNTools, you can create a simple Petrinet, save it somewhere, and open the file in your favourite text editor.
   - Identifiers should be unique, start with **ID**, followed by one or more numbers.

7. To actually generate the .cpn file, execute the command “java -jar metaDepth.jar < commandlist_cpntools” inside of the “exported_to_md” folder. This will generate a “exported.cpn” file.

8. Load the “exported.cpn” file in CPNTools, and perform your analysis.

4.2 Python

1. Using the MetaDepth toolbar, export your railway metamodel. You do this by loading your metamodel (as a model), loading the toolbar and clicking on the “Export MetaModel” button. In the input dialog, write the name of your compiled metamodel (for example, “Railway”).



Insert the name of your metamodel (for example "RPGGame")

Metamodel Name

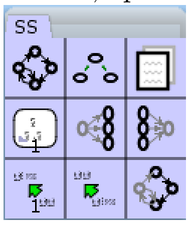
Railway

2. Export the model you want to generate code for using the “Export Model” button. The exported model should be called “exported.mdepth”.

3. Write your EGL code in the file “generate_python.egl” (initially empty). This file should generate a valid Python file, which creates your railway. For an example of a valid game file, look at “exported.py” file in the Railway folder.

5 State Space Analysis in CPNTools

CPNTools allows you to generate the reachability graph for a given Petrinet. First of all, open the *State space* tool, which will show the following palette:



Click the top-left button, and then click on your Petrinet. This will calculate the reachability graph of the Petrinet, and allow you to perform analysis.

CPNTools allows to query the reachability graph, and you will have to create an appropriate query to decide whether the two trains in your model can be scheduled independently. You can write the query as a string (text) in CPNTools, and evaluate it as an ML expression. Documentation on the functions you can use can be found in the following [PDF]. The easiest way to go about this is to try and find a node in the reachability graph for which the condition is *not* satisfied: if such a node can be found, your query will return it, and the trains cannot be scheduled independently. If no such node is found, the trains can be scheduled independently. Use the **SearchNodes** function, and use the **Mark** facility to check the markings of specific places in the nodes of the reachability graph in the condition passed to your call to **SearchNodes**. Please note that **Mark** has its limitations: it is not possible to check the markings of all places without knowing their name. You *have* to hard-code the name of the place as in **Mark.exported'myPlace 1 n** ('myPlace' cannot be a variable). This will probably force you to manually adapt your query whenever another exported model is being checked. Please explain the workflow for checking the condition on arbitrary railway models in your report.

6 Useful Links

- Epsilon Generation Language Documentation: `http://www.eclipse.org/epsilon/doc/egl/`.
- Epsilon book: [PDF]. For this exercise, you need chapter 7 on the Epsilon Generation Language (EGL).
- CPNTools home page: `http://cpntools.org/`
- CPNTools state space manual: [PDF]