Assignment 4

Generic Language Theory

By Marc Craenen and Huib Donkers

Import project into Eclipse

In order to make sure the deliverable.zip file isn't too large for peach, we deleted some auto-generated files from the zip file. In order to make everything work, you can follow the following steps:

- Copy the contents of `deliverable.zip` into a new empty folder
- Open Eclipse
- Switch workspace to the newly created folder
- Import → General → Existing projects into Workspace
- Select the newly created folder and select all projects
- Go to `a4_bounding/src/nl/tue/glt/` and run `GenerateBoundingBox.mwe2`
- Do the same for `GeneratePlatoon.mwe2` in `a4_platoon/src/nl/tue/glt/`

Need to know

Since we wanted this assignment to work properly, we edited our solutions of the previous assignment. Instead of being xtext projects that are coupled to an ecore model, the projects are now just xtext projects that generate ecore models, instead of being coupled to one. We still added the old zip file used for handing in the previous assignment for your convenience, but you will not need it.

The sub-assignments are created in another sequence than they are mentioned in the assignments, so they are covered in this report in this sequence as well, since it is not possible to run it in another sequence.

BoundingBox to Java code

In order to turn a DSL of the BoundingBox into a Java file that returns the coordinates of the lower left and upper right corner, you have to edit `a4_bounding/transform/input.boundingdsl` into the BoundingBox specification of your choice. In this folder there are two other files: the .egx- and .egl file. the .egx file that takes care of transforming the input file to the correct Java code, based on the template in the .egl file.

This .egx file consumes an .xmi file, that is created (and deleted after execution) by `App.java` in `a4_bounding/src/nl/tue/glt`. This Java file first turns the .boundingdsl file into an .xmi file that matches the specification of the .boundingdsl file. After this file is created, the application automatically runs the .egx file, which will produce the correct Java file in the `output` folder.

At last it deletes the generated .xmi file again and copies `BoundingBox.ecore` to the `a4_platoon` project, since it is needed there for the next sub-assignments.

Creating the NXC code

The `a4_platoon` project is set-up in the same way as the `a4_bounding` project, so you have to run `App.java` in order to turn the input file (`transform/input.platoondsl`) into the NXC program (`output/first.nxc` and `output/followers.nxc`) using the .egx and .egl files in the `transform` folder. The generated NXC code needs some explanation:

- Turning (left or right) is only executed when the car in front is too far away. As long as the car is directly in front, the following car should not turn yet. Only when the car in front drives away after turning, the following car should turn and follow. So only when the sensor detects that there is no car directly in front, it checks for the last received command, and turns left or right accordingly.
- We assume the name of each vehicle starts with 'Lego' and is followed by a number. As explained in the previous exercise, we assume all names to be unique.

Platoon route to BoundingBox

The same `a4_platoon` project and `App.java` is used for turning a Platoon specification into a BoundingBox specification. This is done in four steps:

- Turning the input file (`transform/input.platoondsl`) into an .xmi file.
- Turning the .xmi file of the Platoon into an .xmi file of a BoundingBox. This is done using `transform/main.etl`. There are no special design choices in this file.
- Turning the generated .xmi file into a specification in the BoundingBox dsl language, which is stored in the `output` folder.
- Deleting the .xmi files that were automatically generated and used by `App.java`.

The only design choice we made is that we forced a new line in the xtext specification in order to make the output file readable, since otherwise everything was placed on one line.