Exercise Sheet 0: Prepare yourself for Working with MontiCore

Organizational issues

Use the Moodle infrastructure to build groups of **4-5 Students**. Define **one representative** of your group to hand in your solution as Moodle does **not yet support group solutions**. Always make sure to include your names in your solution.

Exercise 0.1: Download and Run (1 Point)

Visit the getting started page http://www.monticore.de/gettingstarted/ for MontiCore and download the Automaton example project. Choose either the Eclipse or IntelliJ variant and follow the instruction of the getting started tutorial on the website. You should end up with a created target folder *‹../target/generated-sources/monticore/sourcecode›* that holds the structure:

- /automaton/
  - _ast containing the abstract syntax representation of the automaton DSL.
  - _cocos containing infrastructure for context conditions of the automaton DSL.
  - _od containing infrastructure for printing object diagrams of the automaton DSL.
  - _parser containing the generated parsers which are based on ANTLR.
  - _symboltable containing infrastructure for the symbol table of the automaton DSL.
  - _visitor containing infrastructure for visitors of the automaton DSL.
- /reports/Automaton

As a solution of this task, zip and hand in your project **including** the generated target.

Exercise 0.2: Adapt the Example (7 Points)

Hint:
If you have questions/problems solving this task, use the reference manual available here: http://www.se-rwth.de/publications/MontiCore-5-Language-Workbench-Edition-2017.pdf to find further explanations.

Now that you have a running MontiCore language project, make yourself familiar with the projects' structure and apply the following changes:

a) Navigate to the grammar (located here: `src/main/grammars`) and change the keyword `automaton` to `aut`. Adjust the models used as examples (located here: src/main/resources) and as test inputs (located here: `src/test/resources`) accordingly. (1 Point)
b) Navigate to the grammar (located here: `src/main/grammars`) and change the `State` production such that states are allowed to have substates only but not `Transitions` within their body. Adapt the handwritten code (located here: `src/main/java`) accordingly. (1 Point)
c) Create a new Automaton model `TrafficLight` that models a typical traffic light that turns green, yellow, red and red-yellow triggered by timeouts. Additionally, include a state where the traffic light is off, which is reached either by electricity failure or turning of the traffic light deliberately. Create a Junit test called `ParseTraffficLightTest` that parses your model and ensures that your model is correct. (3 point)
d) Use the TOP mechanism to extend the AST of the automaton nonterminal. Add a method `getStateNames` that returns a List of the names of all top level, i.e., directly contained, States. Write a Junit Test called `PrintStatesTest` that parses the `PingPong` example, executes the method and checks the list for correctness. (2 Points)

As a solution of this task, zip and hand in your project **without** the generated target, i.e., after running `mvn clean`. Make sure the generated files are reproducible by just running `mvn install`. **Not working projects, e.g., projects that have test failures or compile errors, will not gain you any points**.