Embedded Software and Systems 2021
Assignment 2

Deadline: 23:59, November 28, 2021

Late submissions will not be accepted. Submission is done digitally via Canvas and should include a tar or zip file of the root of the Xtext project, cleaned so it does not contain any generated files. Include the GameOfLife.java file if you modified it. Also include some example programs written in your DSL, correct ones, as well as programs that trigger the validation checks that you have implemented.

Setup Instructions

Start the Eclipse installation that was provided with the DSL Tutorial, or an identical installation. This instance of Eclipse will be the meta-level workspace.

- Create a new Xtext project for the Game of Life DSL, following the instructions in the DSL tutorial
- Without modifying the grammar of the DSL, validate the installation by generating the DSL.
- Run the generated DSL as an Eclipse Application, starting the run-time workspace in a second instance of Eclipse

In the run-time workspace

- Choose 'Open project from file system'
- Click 'Archive' and select the GoLruntime.zip file
- Choose to open the project GoLruntime_expanded/short_life
- Right Click short_life project and Run as Java project to verify that the application works
- Create DSL instance in models directory and validate that the default 'Hello' works

You are now ready to create your own Game of Life DSL.

1. Grammar 4 Points

Define a DSL that allows experiments with variations of:

- the **initial grid**, i.e. which cells are initially dead or alive
- **evolution rules**, i.e. when cells 1) die, 2) live, and 3) become alive

Keep the evolution rules simple and avoid complex expressions. It is sufficient to use number of live neighbors <, =, > some defined number in the DSL instance.

It must be possible to specify all variation points mentioned in bullet list above. The grammar must furthermore be readable and avoid unnecessary complexity. The easier the DSL allows you to specify the rules and initialize complex grids, the higher the score.

2. Code Generation 3 Points

Make a code generator that generates Java code that correctly implements the semantics of the DSL you have specified. The generator should generate new versions of the file RulesOfLife.java. The generated file should overwrite the originally provided file, either automatically or by manually moving the file.

The generated code should correctly implement the specified behavior for all DSL instances. The code should furthermore be readable and make good use of Xtend features.

Tip: Start by moving the current implementation of RulesOfLife.java into the code generator and generate the file just like it was originally provided. Then, modify the generator to consider the initial grid and evolution rules you have specified in your language.

3. Model Validation 3 Points

Add three validation rules to the language you have just created. The validation rules should be meaningful and help the user avoid specifying instances of the language that do not make sense. An example could be to make sure the initial grid does not address cells that our outside the grid in the implementation of the game. Another example could be to make sure the number of specified neighbors required to die, live, or become alive is valid. Think carefully about whether a validation failure should be treated as a warning or an error.

Implement at least three meaningful validation rules that helps the user define correct DSL instances. The implementation of the rules should be readable and make good use of Xtend features. The more useful the validation rules are, the higher the score.