University of Amsterdam

Assignment 2: Domain Specific Language

Embedded Software and Systems

November 28, 2021

Students:
Paras Kumar
13102273

Michael Geuens
13628682

Group:
3

Course:
Embedded Software and Systems

1 Introduction

This assignment can be divided in three parts. The first part is to define a DSL that allow experiments with variations of the initial grid and evolution rules. The second part is to make a code generator that generates Java code that correctly implements the semantics of the DSL we have specified. The final part is to add three validation rules to the language we have just created.

2 1. Grammar

For this part we had to define a DSL that allow experiments with variations of the initial grid and evolution rules. We make sure that the user can define which rows and columns are initially alive. The rows and columns which are not defined are initially dead. Furthermore, we allow the user to define evolutionrules. We have chosen that the user can specify the name, operator and the number of live neighbors. As name the user can only chose between the values of *DieAliveUnit*: *die*, *live* and *become alive*. This is also the case for the operator, the user can only chose between the values of *OperatorUnit*: ==, < and >. We have included our grammar, see figure 1 below

```
grammar game.of.life.LifeDsl with org.eclipse.xtext.common.Terminals

generate lifeDsl "http://www.of.game/life/LifeDsl"

Model:
    'InitialGrid' (grids += Grid)*
    'EvolutionRules' (rules += EvolutionRules)*
;

Grid:
    'Row:' row = INT
    'Column:' column = INT
;

EvolutionRules:
    'Rule:' name = DieAliveUnit
    'ComparisonOperator:' operator = OperatorUnit
    'NumberOfLiveNeighbors:' numberOfLiveNeighbors = INT
;

enum OperatorUnit:
    EQ = '==' |
    L = '<' |
    G = '>'
;

enum DieAliveUnit:
 DIE = 'die' |
 LIVE = 'live' |
 BECOME_ALIVE = 'become alive'
;
```

Figure 1: Grammar

```
InitialGrid
Row: 1 Column: 1
Row: 2 Column: 2
Row: 3 Column: 3

EvolutionRules
Rule: live ComparisonOperator: == NumberOfLiveNeighbors: 2
Rule: live ComparisonOperator: == NumberOfLiveNeighbors: 3
Rule: die ComparisonOperator: < NumberOfLiveNeighbors: 2
Rule: die ComparisonOperator: > NumberOfLiveNeighbors: 3
Rule: become alive ComparisonOperator: == NumberOfLiveNeighbors: 3
```

Figure 2: Grammar result

3 2. Code Generation

For this part we had to to make a code generator that generates Java code that correctly implements the semantics of the DSL we have specified. We have created a new generator file called *JavaGenerator.xtend*. In this file we added the code of *RulesOfLife.java* file. We add the predefined alive cells with a function called *initialAlive*. In this function we add to new points of the cells that should be alive. Furthermore, we have created a function called *evolutionRules*. This function will only add the points of becomeAlive and live rules. After the new *RulesOfLife.java* file is created it is located at: *\GoLruntime.zip_expanded\GoLruntime\short.life\src-gen\GameOfLife\model1.gdsl*. This file should manually be moved to the correct location.

```
 * generated by Xtext 2.25.0
package game.of.life.generator

import game.of.life.lifeDsl.Model

/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
 */
class LifeDslGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        val root = resource.allContents.head as Model;
         if (root !== null) {
             var path = "GameOfLife/" + resource.getURI().lastSegment + "/"
             fsa.generateFile(path+"RulesOfLife.java",JavaGenerator.toJava(root))
         }
    }
}
```

Figure 3: Generator

```
package game.of.life.generator

import game.of.life.lifeDsl.Model

class JavaGenerator {
    def static toJava(Model root)'''
package GameOfLife;

import java.awt.Point;
import java.util.ArrayList;

public class RulesOfLife {
    public static void computeSurvivors(boolean[][] gameBoard, ArrayList<Point> survivingCells) {
        «initialAlive(root)»
        // Iterate through the array, follow game of life rules
         for (int i=1; i<gameBoard.length-1; i++) {
             for (int j=1; j<gameBoard[0].length-1; j++) {
                 int surrounding = 0;
                 if (gameBoard[i-1][j-1]) { surrounding++; }
                 if (gameBoard[i-1][j])   { surrounding++; }
                 if (gameBoard[i-1][j+1]) { surrounding++; }
                 if (gameBoard[i][j-1])   { surrounding++; }
                 if (gameBoard[i][j+1])   { surrounding++; }
                 if (gameBoard[i+1][j-1]) { surrounding++; }
                 if (gameBoard[i+1][j])   { surrounding++; }
                 if (gameBoard[i+1][j+1]) { surrounding++; }
                 /* only code for surving cells, so no rule if result is dead cell */
                  /* rule B3 */
                  «evolutionRules(root)»
             }
         }
    }

}
     '''

 def static initialAlive(Model root)'''
  «FOR grid: root.grids»
    survivingCells.add(new Point(«grid.row»-1, «grid.column»-1));
  «ENDFOR»

'''

 def static evolutionRules(Model root)'''
  «FOR rule: root.rules»
    «IF rule.name === DieAliveUnit::LIVE»
    if ((gameBoard[i][j]) && (surrounding «rule.operator» «rule.numberOfLiveNeighbors»)){
        survivingCells.add(new Point(i-1,j-1));
    }
    «ELSEIF rule.name === DieAliveUnit::BECOME_ALIVE»
        if ((!gameBoard[i][j]) && (surrounding «rule.operator» «rule.numberOfLiveNeighbors»)){
            survivingCells.add(new Point(i-1,j-1));
        }
    «ENDIF»
   «ENDFOR»
 '''
}
```

Figure 4: Java code generator

```java
package GameOfLife;

import java.awt.Point;

public class RulesOfLife {
    public static void computeSurvivors(boolean[][] gameBoard, ArrayList<Point> survivingCells) {
        survivingCells.add(new Point(1-1, 1-1));
        survivingCells.add(new Point(2-1, 2-1));
        survivingCells.add(new Point(3-1, 3-1));

        // Iterate through the array, follow game of life rules
         for (int i=1; i<gameBoard.length-1; i++) {
             for (int j=1; j<gameBoard[0].length-1; j++) {
                 int surrounding = 0;
                 if (gameBoard[i-1][j-1]) { surrounding++; }
                 if (gameBoard[i-1][j])   { surrounding++; }
                 if (gameBoard[i-1][j+1]) { surrounding++; }
                 if (gameBoard[i][j-1])   { surrounding++; }
                 if (gameBoard[i][j+1])   { surrounding++; }
                 if (gameBoard[i+1][j-1]) { surrounding++; }
                 if (gameBoard[i+1][j])   { surrounding++; }
                 if (gameBoard[i+1][j+1]) { surrounding++; }
                 /* only code for surving cells, so no rule if result is dead cell */
                  /* rule B3 */
                  if ((gameBoard[i][j]) && (surrounding == 2)){
                    survivingCells.add(new Point(i-1,j-1));
                  }
                  if ((gameBoard[i][j]) && (surrounding == 3)){
                    survivingCells.add(new Point(i-1,j-1));
                  }
                  if ((!gameBoard[i][j]) && (surrounding == 3)){
                    survivingCells.add(new Point(i-1,j-1));
                  }
             }
         }
    }

}
```

Figure 5: Java code generator result

4 3. Model Validation

For this part we had to add three validation rules to the language we have just created. We first check whether the predefined alive cells are not double. If the row and column are double an error will show. Furthermore, we check if the number of specified neighbors required to die, live, or become alive is valid. If the rules of die or live is not valid an error is show and if the rule of become alive is not valid we have chosen to show an info. Moreover, we check if the evolution rules are not double. If the rules are double an error will show. Additionally we also checked for possible logical errors. We checked if the the row number and column number in the grid is below 0. We also checked whether more than 8 neighbours are alive since this would violate the basic property of the grid.

```
class LifeDslValidator extends AbstractLifeDslValidator {

    @Check
    def checkDoubleInitialGrids(Model root) {
        var glist = root.grids // lists start at position 0
        for (var i = 0; i < glist.size; i++) {
            for (var j = i + 1; j < glist.size; j++) {
                if (
                    glist.get(i).row.equals(glist.get(j).row)
                    && glist.get(i).column.equals(glist.get(j).column)
                ) {
                    error("Double grid", null)
                }
            }
        }
    }

    @Check
    def checkDieAliveUnit(EvolutionRules rules) {
            if (rules !== null) {
                switch (rules.name) {
                    case DieAliveUnit::DIE:
                        if (
                            rules.numberOfLiveNeighbors == 3
                            && (
                                rules.operator == Operator::L
                                || rules.operator == Operator::EQ
                            )
                        ) {
                            error("Neighbors less than or equal to 3 not
                                allowed to die", null)
                        }
                    case DieAliveUnit::LIVE:
                        if (rules.numberOfLiveNeighbors != 2 && rules.numberOfLiveNeighbors != 3) {
                            error("Neighbors less than 2 and more than 3 not
                                allowed to live", null)
                        }
                    case DieAliveUnit::BECOME_ALIVE:
                        if (rules.numberOfLiveNeighbors != 3) {
                            info("Maybe rewrite to live or die", null)
                        }
                }
            }
    }

    @Check
    def checkEvolutionRules(Model root) {
            var rlist = root.rules // lists start at position 0
            for (var i = 0; i < rlist.size; i++) {
                for (var j = i + 1; j < rlist.size; j++) {
                    if (
                        rlist.get(i).name.equals(rlist.get(j).name)
                        && rlist.get(i).operator.equals(rlist.get(j).operator)
                        && rlist.get(i).numberOfLiveNeighbors.equals(rlist.get(j).numberOfLiveNeighbors)
                    ) {
                        error("Double rule", null)
                    }
                }
            }
    }

    @Check
    def checkGrid(Grid grid) {
        if(grid.row < 0 || grid.column < 0){
            error("Cell location can not be below 0",null)
        }
    }

    @Check
    def checkNeighbours(EvolutionRules rules) {
        if(rules.numberOfLiveNeighbors > 8) {
            error("It is not possible to have more than 8 live neighbors",null)
        }
    }
}
```

Figure 6: Validations