UNIVERSITÉ TOULOUSE III PAUL SABATIER

Labwork 5 – Extensions to CellLang

These labworks are automatically assessed based on an archive you have to deliver on time on the Moodle webpage. To make the archive, you have to type the command:

```
> make archive
```

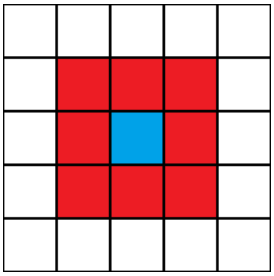
This produces a file named `archive.tgz` that you have to deposit. As the compilation labworks are held each week, deposit deadline is set one day before your next session (to perform the assessment). This deadline is not strict but delay will have a negative impact on your mark and you will not benefit from the comments of your teacher.

This exercise consists in implementing ane extension to *CellLang*. The following process has to be applied:

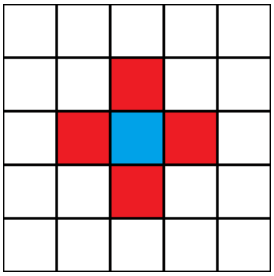
1. Understand the proposed syntax and extension.
2. If needed, add required tokens to `parser.mly`.
3. If needed, add the token scanning to `lexer.mll`.
4. Add the required rules to `parser.mly`.
5. Add the AST and build them in `parser.mly`.
6. Provide the translation to quads in `comp.ml`.

Very important! replace the `Makefile` and `ast.ml` with the ones provided in Moodle.

An important part of CellLang programs is to perform the same calculation with the neighbor cells that are around the current cell. Although there are a lot of different ways to select which neighbor cells to work with, there are two main neighborhood traversal policies:



Moore's neighborhood



Von Neumann's neighborhood

This extensions aims to provide loop constructions to implement these neighborhood policies. They will have the following syntax:

- *for* `i` *in* `moore` *do* `s = s + i` *end*
- *for* `i` *in* `vonneumann` *do* `s = s + i` *end*

The statements between `do` and `end` is repeated as many times as there are cells in the neighborhood policy (the order of traversal is undefined). The variable `i` (or any other identifier) gets the value of the current neighbor cell in the loop. There is no constraints on the identifier `i`: it could have been already declared or not.

Tests

- `test52/moore.auto` for Moore's policy.
- `test52/voneumann.auto` for Von Neumann's policy.

Hint 1: the constants for the directions – `pNORTH`, `pNORTEAST`, ..., have been cleverly chosen to make easier the implementation of these loops (look in `cell.ml`).

Hint 2: with the VM command `cGET`, the lookup direction was determined by a constant direction stored in b argument. To implement this extension, we need to be able to pass a direction stored in a register. This is implemented by the command `cGETV` (value 1005): b argument is the register number containing the direction.

Hint 3: in order to be used in the body of the `for` loop, the variable i needs to be declared but the `for` action is only invoked when the whole rule is parsed. To get a chance to declare i before the body is parse, the `for` header can handled in sub-rule which action is invoked before the rest of the loop and this rule has to declare i and return its register number.

```
for_header: FOR ID
        { (* declare and return register number *) }
;
```