Lab 4
Operational Semantics for Mu language

Objective

- Write semantic and typing rules.
- Implement them as visitors.

First download the archive from the course website.

4.1 Semantics for expressions

The (natural) semantics for expressions is given by the following rules seen in the course: (I do not recall the notations)

$$\mathscr{A}[n]\sigma = \mathscr{N}(n)$$
$$\mathscr{A}[x]\sigma = \sigma(x)$$
$$\mathscr{A}[e_1 + e_2]\sigma = \mathscr{A}(e_1) +_I \mathscr{A}(e_2)$$

EXERCISE #1 ► **Expressions**
We give you a visitor that computes this semantics. Compare the implementation and the semantics.

4.2 Semantics for Mu-language

We give you the syntax of the Mu language, as a full grammar depicted in Figure 4.1.

EXERCISE #2 ► **Mu semantics**
Write on paper the (big steps) operational semantics as already seen in the course. You can forget the `log` construction, that basically prints the expression given in argument.

EXERCISE #3 ► **Evaluator!**
Write the evaluator for this mini-language. We give you the structure of the code and the evaluator for numerical expressions and boolean expressions. For the moment, only consider well-typed expressions.

EXERCISE #4 ► **Typing**
Write typing rules for expressions (on paper). Then, implement a type checker for the Mu language[1] (as a standalone visitor). We provide you a (basic) class for basic types. Do not forget to add informative exception handlers and to intensively use new test files.

EXERCISE #5 ► **Bonus (on paper)**
We want to extend our mini language with imperative arrays. The syntax is augmented with the three following constructions:

- `Alloc(e)` allocates a new array of size equal to the value of e, with undefined values
- `Read(e1,e2)` reads the e_2^{th} value of array e_1.
- `Write(e1,e2,e3)` modifies the e_2^{th} value of array e_1 with the value of expression e_3.

Complete the semantics of expressions, then give new rules for array modification.

[1]We do not ask for a decorated AST, only type checking.

```
grammar Mu;

prog
 : block EOF
 ;

block
 : stat*   #statList
 ;

stat
 : assignment
 | if_stat
 | while_stat
 | log
 | OTHER {System.err.println("unknown␣char:␣" + $OTHER.text);}
 ;

assignment
 : ID ASSIGN expr SCOL #assignStat
 ;

if_stat
 : IF condition_block (ELSE IF condition_block)* (ELSE stat_block)? #ifStat
 ;

condition_block
 : expr stat_block  #condBlock
 ;

stat_block
 : OBRACE block CBRACE
 | stat
 ;

while_stat
 : WHILE expr stat_block #whileStat
 ;

log
 : LOG expr SCOL #logStat
 ;
```

Figure 4.1: MU syntax. We omitted here the subgrammar for expressions