Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: November 12, 2024

This final project will use GOLD to perform syntactic analysis of the robot language from projects 0 and 1.

Here's a reminder of the language for robot programs:

Input for the robot can be an execution command or a definition.

- An execution command is preceded by the word `EXEC` and followed by a block.
- Definitions are preceded by the keyword `NEW`. We can define variables and macros.
  - A variable definition is of the form: `NEW VAR name=n` where `name` is a variable's name and `n` is a value used to initialize the variable.
  - A macro definition is of the form: `NEW MACRO name (params)B`. This is used to define a procedure. Here `name` is the procedure name, `(Params)` is a list of parameters separated by commas, and `B` is a block. A parameter is a name.
- A block is of the form $\{instruction; \ldots instruction;\}$
- An instruction can be a command or a control structure.
- A command can be:

1. Assignment: `name = n` where `name` is a variable's name and `n` is a value. The result of this instruction is to assign the value `n` to the variable.
2. Macro invocation: A macro name followed by its parameter values within parenthesis and separted by commas, for example `macroName(p1,p2,p3)`.
3. `turnToMy(D)`: where `D` can be `left`, `right`, or `back` (The robot should turn 90 degrees in the direction of the parameter (if the parameter is `left` or `right`), and 180 if the parameter is `back`.
4. `turnToThe(O)`: where `O` can be `north`, `south`, `east`, or `west`. The robot should turn so that it ends up facing direction `O`.
5. **walk**`(n)`: where `n` is a value. The robot should move `n` steps forward.
6. `jump(n)`: where `n` is a value. The robot should jump `n` steps forward.
7. **drop**`(n)`: Where `n` is a value. The robot should drop `n` chips.
8. **pick**`(n)`: Where `n` is a value. The robot should pick `n` chips.
9. **grab**`(n)`: Where `n` is a value. The robot should get `n` balloons.

---

10. `letGo(n)`: Where `n` is a value. The robot should put `n` balloons.
11. `pop(n)`: Where `n` is a value. The robot should pop `n` balloons.
12. `moves (Ds)`: where `Ds` is a non-empty list of directions: `forward`, `right`, `left`, or `backwards` separated by commas. The robot should move in the directions indicated by the list and end up facing the same direction as it started.
13. `nop`: a command that does not do anything
14. `safeExe(C)`: C is any command (5-11, above). C is executed if it will not cause an error. Otherwise, it does not do anything.

- Values: A value is a number, a variable, or a constant. The constants that can be used are:
  - size : the dimensions of the board
  - myX: the x position of the robot
  - myY: the y position of the robot
  - myChips: number of chips held by the robot
  - myBalloons: number of balloons held by the robot
  - balloonsHere: number of balloons in the robot's cell
  - chipsHere: number of chips that can be picked
  - roomForChips: number of chips that can be dropped
- Control Structures:

  Conditional: **if** `(condition)` **then** `B1` **else** `B2` **fi**: Executes `B1` if `condition` is true and `B2` if `condition` is false. `B1` and `B2` are blocks.

  Loop: **do** `(condition)B od`: Executes `B` while `condition` is true. `B` is a block.

  RepeatTimes: `rep n` **times** `B per` where `n` is a value and `B` is a block. `B` is executed `n` times.

- These are the conditions:
  - `isBlocked?(D)`: `D` is `left`, `right`, `front` or `back`. The condition is true if the robot is blocked at the corresponding direction.
  - `isFacing?(O)`: `O` is `north`, `south`, `east`, or `west`.The condition is true if the robot is facing the corresponding orientation.
  - `zero?(n)`: Returns true if `n` is zero.

– **not**(C): C is a condition. It is true if C is false.

Spaces, newlines, and tabulators are separators and should be ignored.

Remember the robot cannot walk over obstacles, and when jumping, it cannot land on an obstacle. The robot cannot walk off the board or land off the board when jumping.

Task 1. The task for this project is to use GOLD to perform syntactical analysis for the Robot. Specifically, you have to complete the definition of the lexer (a finite state transducer) and the parser (a pushdown automata). You only have to determine whether a given robot program is syntactically correct: you do not have to interpret the code.

Attached you will find the GOLD project **LexerParserRobot202420**, where we have implemented a compiler for a subset of the language.

This project contains three files:

1. `LexerParserRobot202420.gold`: the main file (the one you must run to test your program via console).
2. `Lexer202420.gold`: the lexer
3. `ParserRobot202420.gold`: the parser

The lexer reads the input and generates a token stream. The parser accepts execution instructions and only the following commands:

- `drop(n)` where n is a number.
- `turnToThe(north)`
- `turnToMy(right)`
- `jump(var)` where `var` is an identifier.

You have to modify `Lexer20242020.gold` so it generates tokens for the whole language. You only have to modify the procedure `initialize`. You also have to modify `ParserRobot202420.gold` so you recognize the whole language.

Important: For this project, we asume that the language is *case sensitive*. This is to say it does distinguish between lower case and upper case letters. For example, `waLK` will be interpreted as an identifier, not as the keyword `walk` .

You do not have to verify whether or not variables and functions have been defined before they are used.

Below we show an example of a valid program.

```
 EXEC  {
if (not(isBlocked?(left))) then  { turnToMy(left); walk(1); }
   else {nop;}  fi
}

EXEC {
 safeExe(walk(1));
 moves(left,left, forward, right, backwards);
}


NEW VAR rotate= 3
NEW MACRO foo (c, p)
{  drop(c);
   letgo(p);
   walk(rotate);
}
EXEC  { foo (1 ,3) ; }

NEW VAR one= 1
NEW MACRO      goend ()
{
   if (not (blocked?(front)))
   then  { move(one); goend();  }
   else  { nop; }
    fi;
}

NEW MACRO fill ()
  {
  repeat roomForChips times
  {  if (not (zero?(myChips))) { drop(1);}  else { nop; } fi ;}
     per ;
  }

  NEW MACRO fill1 ()
  {
  do (not (zero?(roomForChips)))
  {  if (not (zero?(myChips))) { drop(1);}  else { nop; } fi ;
  } od ;
  }

NEW MACRO grabAll ()
{ grab (balloonsHere);
}
```