Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: November 8, 2022

The project work for the course consisted of managing and operating a language to program a robot in a two-dimensional world.

Robot Description

The robot is able to move in the world (delimited by an $n \times n$ matrix); the robot moves from cell to cell. Cells are indexed by rows and columns. The top left cell is indexed as (1,1). North is top; West is left. The robot interacts (picks and puts down) with two different types of objects (chips and balloons). Additionally, note that the robot cannot move on, or interact with obstacles in the world (gray cells).

This final project will use GOLD to perform syntactic analysis of the ROBOT programs.

Recall the definition of the language for robot programs.

- A program defintion begins with the keyword `PROG` possibly followed by a declaration of variables, followed by zero or more procedure defintions, followed by a block of instructions. It ends the keyword `GORP`.
- A declaration of variables is the keyword `VAR` followed by a list of names separated by commas. A name is a string of alphanumeric characters that begins with a letter. The list is followed by `;`.
- A procedure defintion is a the word `PROC` followed by a name followed by a list of parameters within parenthesis separated by commas, followed by a block of instructions and ending with the word `CORP`.
- A block of instructions is a sequence of instructions separated by semicolons within curly brackets.
- An instruction can be a command,a control structure or a procedure call.
  - A command can be any one of the following:
    - An assignment: `name := n` where `name` is a variable name and `n` is a number. The result of this instruction is to assign the value of the number to the variable.
    - `walk(n)` – where `n` is a number or a variable or a parameter. The robot should move n steps forward.
    - `jump(n)` – where `n` is a number or a variable or a parameter. The robot should jump n steps forward.

  * `jumpTo(n,m)` – where `n` and `m` are numbers, variables, or parameters. The robot should jump to position `(n,m)`.
  * `veer(D)` – where `D` can be `left`, `right`, or `around`. The robot should turn 90 degrees in the direction D.
  * `look(O)` – where `O` can be north, south, east or west. The robot should turn so that it ends up facing direction `O`.
  * `drop(n)` – where `n` is a number or a variable or a parameter. The Robot should put `n` chips from its position.
  * `grab(n)` – where `n` is a number or a variable or a parameter. The Robot should get `n` ballons from its position.
  * `get(n)` – where `n` is a number or a variable or a parameter. The Robot should get `n` chips from its position.
  * `free(n)` – where `n` is a number or a variable or a parameter. The Robot should put `n` balloons from its position.
  * `pop(n)` – where `n` is a number or a variable or a parameter. The Robot should pop `n` balloons from its position.
  * `walk(d,n)` – where `n` is a number or a variable or a parameter; `d` is a direction, either front, right, left, back. The robot should move `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
  * `walk(o,n)` – where `n` is a number or a variable or a parameter; `o` is north, south, west, or east. The robot should face `O` and then move `n` steps.

- A control structure can be:

  Conditional: `if (condition)Block1 else Block2 fi` – Executes `Block1` if `condition` is true and `Block2` if `condition` is false.

  Conditional `if (condition)Block1 fi`– Executes `Block1` if `condition` is true does not do anything if it is false.

  Loop: `while (condition)do Block od` – Executes `Block` while `condition` is true.

  Repeat: `repeatTimes n Block per` – Executes `Block` `n` times, where `n` is a variable or a parameter or a number.

- These are the conditions:

  * `isfacing(O)` – where `O` is one of: north, south, east, or west
  * `isValid(ins,n)` – where *ins* can be `walk`, `jump`, `grab`, `pop`, `pick`, `free`, `drop` and `n` is a number or a variable. It is true if *ins*`(n)` can be executed without error.

  * `canWalk(d,n)` – where `D` is one of: north, south, east, or west and n is a number, a variable or a parameter.
  * `canWalk(o,n)` – where `D` is one of: right, left, front, or back and n is a number, a variable or a parameter.
  * `not (cond)` – where `cond` is a condition

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

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).

For this project, you will only have to determine whether a given robot program is syntactically correct.

Attached you will find two GOLD projects.

LexerParserExampleMiniLisp : Here we implement the compiler of a simple language using a transducer for the lexer and a push-down automaton for the parser. In the Eclipse project, the docs folder includes a pptx file that explains how this is done.

LexerParserRobot202220 contains a lexer for a subset of the language. The lexer reads the input and generates a token stream. We also provide a simple parser that only accepts a couple of commands.

- `walk(n)` where n is a number or an identifier
- `look(north)`
- `walk(right,n)` where n is a number or an identifier

From the second project, you have to modify `Lexer202220.gold` so it generates tokens for the whole language. You only have to modify procedure `initialize` (line 210). You also have to modify `ParserRobot202220.gold` so you recognize the whole language.

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.

```
PROG
var n, x, y;
PROC putCB(c, b)
{
   drop(c);
   free(b);
   walk(n)
}
CORP
PROC goNorth()
{

   while (canWalk(north,1)) do { walk(north,1)} od

}
CORP
PROC goWest()
{

   if (canWalk(west,1))  { walk(west,1)} fi

}
CORP
{
go(3,3);
n=6;
putCB(2,1)




}

GORP
```