Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: May 12, 2023

---

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 for the robot is the keyword ROBOT_R possibly followed by a declaration of variables, a definition of procedures, and ends with a a block of instructions.
- A declaration of variables is the keyword VARS followed by a list of names separated by commas. A name is a string of alphanumeric characters that begins with a letter. This declaration should end with a. semicolon ";".
- Procedure declaration starts with the keyword PROCS followed by one or more procedure definitions.
- A procedure definition is the procedure's name followed by "[", then its parameters, followed by the instructions separated by semicolons (";"). The parameters are specified by a list of names separated by commas preceded and followed by the symbol "|"; the definition ends with "]".
- A block of instructions is a sequence of instructions separated by semicolons within square brackets [] .
- An instruction can be a command or a control structure or a procedure call
  - A command can be any one of the following:
    - `assignTo: n , name` where `name` is a variable's name and `n` is a number. The result of this instruction is to assign the value of the number to the variable.
    - **goto**`: x, y` – where `x` and `y` are numbers or variables. The robot should go to position `(x, y)`.

  - `move: n` – where `n` is a number or a variable. The robot should move n steps forward.
  - `turn: D` – where `D` can be left, right, or around. The robot should turn 90 degrees in the direction of the parameter.
  - `face: O` – where `O` can be north, south, east or west. The robot should turn so that it ends up facing direction `O`.
  - **put**`: n , X` – where `X` corresponds to either Balloons or Chips, and `n` is a number or a variable. The Robot should put `n X`'s.
  - **pick**`: n ,X` – where `X` is Balloons or Chips and `n` is a number or a variable. The robot should pick `n X`'s.
  - `moveToThe: n, D` – where `n` is a number or a variable. `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.
  - **moveInDir**`: n, O` – here `n` is a number or a variable. `O` is north, south, west, or east. The robot should face `O` and then move `n` steps.
  - `jumpToThe: n, D` – where `n` is a number or a variable. `D` is a direction, either front, right, left, back. The robot should jump `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
  - `jumpInDir: n , O` – here `n` is a number or a variable. `O` is north, south, west, or east. The robot should face `O` and then jump `n` steps.
  - `nop:` The robot does not do anything.

- a procedure is invoked using the procedure's name followed by ":" followed by its arguments separated by commas.
- A control structure can be:

  **Conditional:** **if**`: condition` **then**`: Block1` **else**`: Block2` – Executes `Block1` if `condition` is true and `Block2` if `condition` is false.

  **Loop:** **while**`: condition` **do**`: Block` – Executes `Block` while `condition` is true.

  **RepeatTimes:** **repeat**`: n Block` – Executes `Block` `n` times, where `n` is a variable or a number.

- A condition can be:
  - **facing**`: O` – where `O` is one of: north, south, east, or west
  - **canPut**`: n , X` – where `X` can be chips or balloons, and `n` is a number or a variable

* **canPick**: `n , X` – where `X` can be chips or balloons, and `n` is a number or a variable
* `canMoveInDir: n , D` – where `D` is one of: north, south, east, or west
* `canJumpInDir: n, D` – where `D` is one of: north, south, east, or west
* `canMoveToThe: n ,O` – where `O` is one of: front, right, left, or back
* `canJumpToThe: n, O` – where `O` is one of: front, right, left, or back
* **not**: `cond` – where `cond` is a condition

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

The language is not case-sensitive. This is to say, it does not distinguish between upper and lower case letters.

**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 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.

**LexerParserRobot202310** : where we have implemented a compiler for a subset of the language.

This project contains three files:

1. `LexerParserRobot202310.gold`: the main file (the one you must run to test your program via console).
2. `Lexer202310.gold`: the lexer
3. `ParserRobot202310.gold`: the parser

The lexer reads the input and generates a token stream. The parser only accepts a couple of commands.

- `move: n` where n is a number
- `face: north`
- `moveToThe: var, right` where var is an identifier

Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: May 12, 2023

---

You have to modify `Lexer20231020.gold` so it generates tokens for the whole language. You only have to modify procedure `initialize` (line 210). You also have to modify `ParserRobot202310.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, **`movetothe`** will be interpreted as an idetifier, not as the keyword **`moveToThe`**.

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.

---

```
1  ROBOT_R
2  VARS nom, x, y, one;
3  PROCS
4  putCB [ |c, b| assignTo: 1, one;
5  put : c,  chips;  put: b ,  balloons ]

7  goNorth  [| |
8  while: canMoveInDir 1 , north do: [ moveInDir: 1 , north ]
9  ]


12 goWest [  |   | if: canMoveInDir: 1, west then: [MoveInDir: 1 ,
       west]  else: [nop:]]

14 [
15 goTo: 3, 3
16 putcb: 2 ,1
17 ]
```

---