Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: November 23, 2021

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 proyect will use GOLD to perform syntactic analyis of the ROBOT programs.

Recall the language definition:

- A command can be any one of the following:
  - `MOVE n` – where `n` is a number or a previously defined variable. The robot should move n steps forward.
  - `RIGHT n` – where `n` is a number or a previously defined variable. The robot should turn n degrees clockwise.
  - `LEFT n` – where `n` is a number or a previously defined variable. The robot should turn n degrees counter-clockwise.
  - `ROTATE n` – where `n` is a number or a previously defined variable. The robot should turn n degrees in any direction (at random).
  - `LOOK O` – where `O` can be `N`, `E`, `W`, or `S`. The robot should turn so that it ends up facing direction North if `O` is `N`, East if `O` is `E`, West if `O` is `W`, and South if `O` is `S`.
  - `DROP n` – where `n` is a number or a previously defined variable. The Robot should drop `n` chips.
  - `FREE n` – where `n` is a number or a previously defined variable. The Robot should let go of `n` balloons.
  - `PICK n` – where `n` is a number or a previously defined variable. The Robot should pickup `n` chips.
  - `POP n` – where `n` is a number or a previously defined variable. The Robot should pop `n` balloons.

- `CHECK O n` – where `O` is either `C` for chips, or `B` for balloons, and `n` is a previously defined variable or a number. The robot check if there are `n` chips or balloons (depending on the value of `O`) in the robot's position.
- `BLOCKEDP` – is a boolean predicate to check if the robot is blocked (it can or cannot move forward)
- **NOP**. the robot does not do anything.
- A block of commands: `(BLOCK commands)` where commands is simply a sequence of one or more commands (separated by new lines).
- Iterative instructions: `REPEAT n [commands]`, where `n` is a variable or a number describing the number of times the commands inside the `[]` will repeat, and `commands` is a sequence of basic commands separated by new lines.
- A conditional command: `IF expr [ commands ]`, where `expr` is a boolean expression, and `commands` is a sequence of basic commands separated by new lines
- `DEFINE n val` – defines a new variable `n` assigning it value `val` (an integer). Note that variable names need to be lowercase.
- `TO f :param OUTPUT expression END`. Functions are defined between the `TO` and `END` keywords, giving them a name `f` and a list of space separated parameters each defined by the colons before its name (as in `:param`). The inner works of a function are given as an expression or block of commands in its `OUTPUT`.

Task 1. The task for this project is to use GOLD to perform a syntactical analysis of tokenized routines for the Robot. Specifically, you should use a pushdown automata to parse the routines of the Robot language.

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

We include a GOLD file (`LexerParser1.gold`) that implements 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.

The file `LexerParserRobot202120.gold` contains a lexer for the language. The lexer reads the input and generates a token stream. We also provide a simple parser that only accepts the rotate command.

It is not required for you to verify whether or not names have been previously defined. This would require modifying the lexer. You may have a token for a previously defined variable and for a new one. Your parser should know when to accept new variables, and when to accept previously defined variables.

The following table shows the symbols used to represent each token.

Input	Token Representation
"("	(
")"	)
"["	[
"]"	]
"!"	!
"\n"	⋈
"N"	N
"S"	S
"W"	W
"E"	E
"C"	C
"B"	B
"NOP"	K
"MOVE"	M
"RIGHT"	r
"LEFT"	l
"ROTATE"	R
"LOOK"	L
"DROP"	D
"FREE"	F
"PICK"	P
"POP"	p
"CHECK"	c
"BLOCKEDP"	w
"BLOCK"	b
"REPEAT"	X
"IF"	?
"DEFINE"	d
"TO"	T
"facing?"	F
"OUTPUT"	O
"END"	$
name	v
Name	V
number	#
parameter	x

An example of a valid input for the robot is shown below. You may assume that the name space is maintained throughout the execution of the examples. This is to say: you can enter each instruction separately, and definitions will be stored throughout the execution of the whole program.

```
ROTATE 3

IF BLOCKEDP [MOVE 1
NOP]

(BLOCK
IF BLOCKEDP [MOVE 1
NOP]
LEFT 90
)

DEFINE one 1

TO foo :c :p
OUTPUT
   DROP :c
   FREE :p
    MOVE one
END
foo 1 3


TO goend
OUTPUT IF !BLOCKEDP [
   (BLOCK MOVE 1
          goend)
   NOP
  ]
END
```