Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: April 28, 2025

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:

- Robot programs are a sequence of variable definitions, procedure definitions, variable accesses, and procedure calls.
- Variables are always declared at the beginning of a a program (top of the file) and are defined as a list of names (starting with a small caps letter) and separated by spaces. The list is defined in a block enclosed by a pipe `|` symbol. Variables declared inside a procedure are local to that procedure and should be defined at the beginning of the procedure. For example `|t1 t2 t3|` declares three variables named `t1`, `t2`, and `t3`, respectively
- Procedure declaration starts with the keyword `proc` followed by the procedure name, its parameters, and a code block.
- Procedure names are defined as a combination of strings and parameter values. Procedure names are given by an alphanumeric string starting with a smallcaps letter and ending with a colon (`:`). After the name the parameter is given, separated by spaces from the previous and following parts of the name. An example of a procedure definition is `proc myMethod: param1 and: param2 [ ]`. Note that for procedures without parameters there is no need to add the colon at the end of the name, for example, `proc new [ ]`.
- A procedure call is given by the procedure name and its parameters, finishing with a period (`.`). For example, `myMethod: 3 and: 4.` calls the procedure previously declared with parameters 3 and 4.
- A code block is a sequence of instructions (*e.g.,* variable declarations, assignments, procedure calls, conditionals, loops) separated by periods within square brackets `[ ]`.
- The robot is able to recognize the following set of instructions:
  - Variable assignments that are defined as a variable name the assignment operator (`:=`), the given value, and a period. For example, `t1 := 3 .` The result of this instruction is to assign the value of the number to the variable.
  - constants of the language are alphanumeric strings that start with a `#`. For example `#balloons`

- `goto: x with: 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`. In the first two cases the robot should turn 90 degrees in the direction of the parameter (according to the direction the robot is facing), and in the last case the robot should turn 180 degrees (end up facing the oposite direction).
- `face: O .` – where `O` can be `#north`, `#south`, `#west`, or `#east`. The robot should turn so that it ends up facing direction `O`.
- `put: n ofType: 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 ofType: X .` – where `X` is `#balloons` or `#chips` and `n` is a number or a variable. The robot should pick `n X`'s.
- `move: n toThe: 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.
- `move: n inDir: 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.
- `jump: n toThe: 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.
- `jump: n inDir: 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.

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

- A condition can be:

- `facing: O .` – where `O` is one of: `#north`, `#south`, `#west`, or `#east`
- `canPut: n ofType: X .` – where `X` can be `#balloons` or `#chips`, and `n` is a number or a variable
- `canPick: n ofType: X .` – where `X` can be `#balloons` or `#chips`, and `n` is a number or a variable
- `canMove: n inDir: D .` – where `D` is one of: `#north`, `#south`, `#west`, or `#east`
- `canJump: n inDir: D .` – where `D` is one of: `#north`, `#south`, `#west`, or `#east`
- `canMove: n toThe: O .` – where `O` is one of: `#front`, `#right`, `#left`, `#back`
- `canJump: n toThe: O .` – where `O` is one of: `#front`, `#right`, `#left`, `#back`
- `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). 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 **LexerParserRobotP3**, where we have implemented a compiler for a subset of the language.

This project contains three files:

1. `LexerParserRobot202510.gold`: the main file (the one you must run to test your program via console).
2. `Lexer202510.gold`: the lexer
3. `ParserRobot202510.gold`: the parser

The lexer reads the input and generates a token stream. For the example, the parser accepts blocks of commands with only the following instructions :

- `move: x.` where x is a number or a variable
- `move: x inDir: #right.` where x is a number or a variable
- `turn: #north.`

You have to modify `Lexer202510.gold` so it generates tokens for the whole language. You only have to modify the procedure `initialize`. You also have to modify `ParserRobot202510.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, `moVE` will be interpreted as an identifier, not as the keyword `move`.

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

As before, below we show examples of valid inputs which you can use as inspiration to test full robot (valid and invalid) programs.

```
|nom x y one|

proc putChips: n andBalloons: m [
    |c, b|
    c := n .
    b := m .
    put : c ofType: #chips .  put: b ofType:  #balloons ]

proc goNorth  [
    while: canMove: 1 inDir: #north do: [ move: 1 InDir: #north .
        ]
]


proc goWest [
   if: canMove: 1 InDir: #west then: [move: 1 InDir: #west]  else
      : [nop .]]

[
    goTo: 3 with: 3 .
    putChips: 2 andBalloons: 1 .
]
```