Computational Methods Robot

Contributors:

- Israel vidal paredes - A01750543
- Javier Eric Hernández - A01635390
- Mariana BusTos Hernández - a01641324

LINK TO REPO: https://github.com/BoJavs-svg/RobotComputationalMethods

This project simulates the CPU of a car robot, with its own programming language and compiler. The robot moves in a 2-D square matrix of 10 blocks. The project was implemented using Lex and Yacc, and Python.

Machine state and CPU simulator

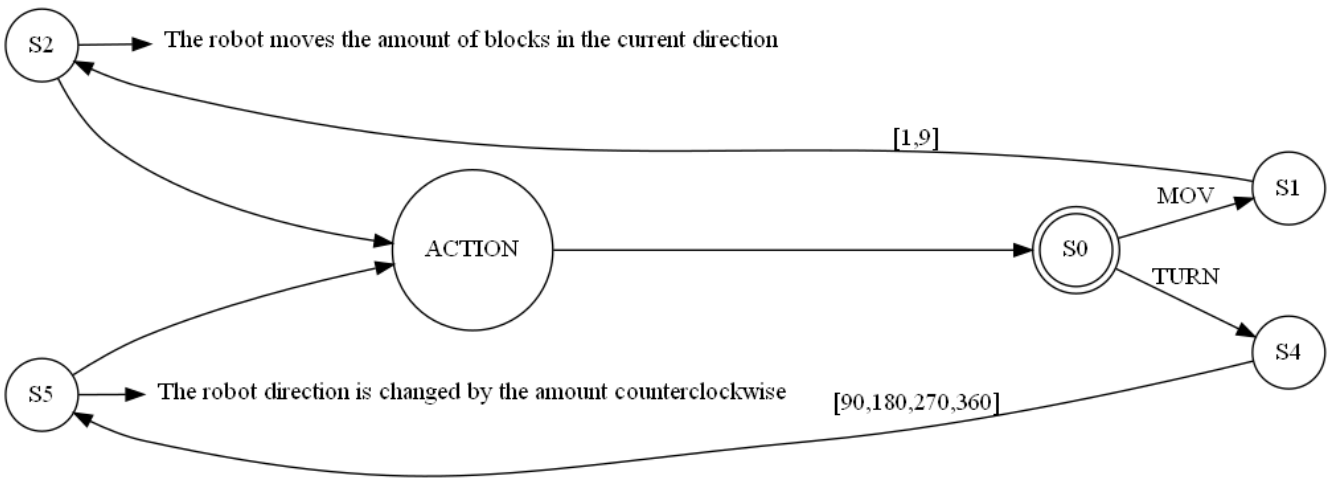
Problem description:

A robot language and compiler first needs a CPU that is capable of reading and executing instructions; in order to simulate the functionalities that such robot would have, a the *CPU.py* file found in the */src* folder has the capacity to:

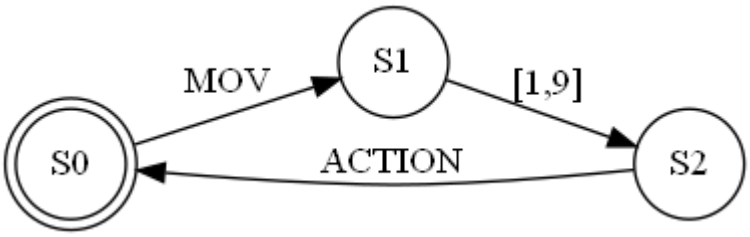
- [x] Open and queue instructions from .asm file
- [x] Understand and execute instructions
- [x] Draw the machine's state on a matrix.

Diagram

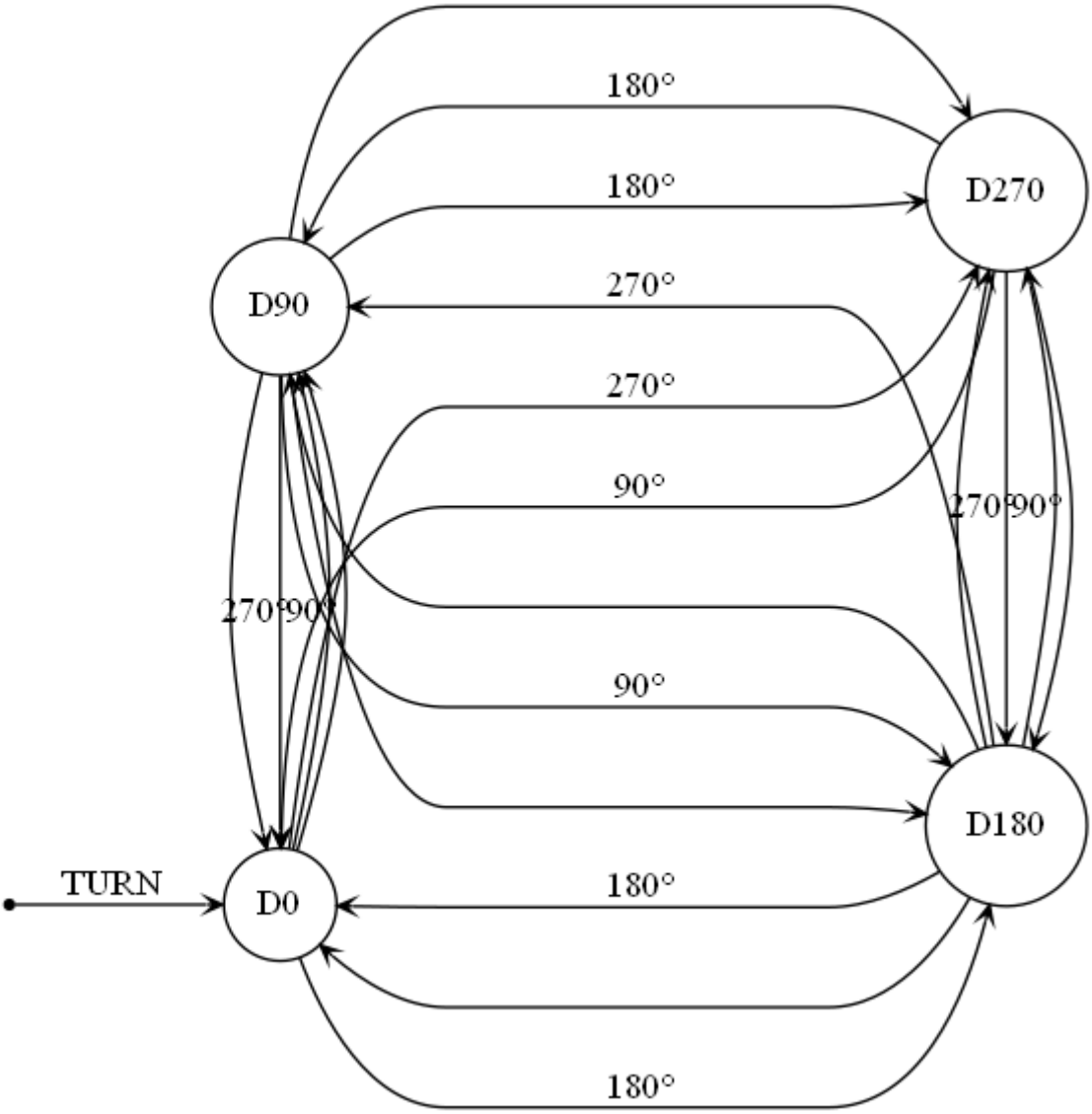
First, the robot awaits a comand to be executed, which can be classified as either a "*mov*" comand or a "*turn*" comand:



For a **mov** comand, the robot takes takes the specified amount and executes the action:



For a **turn** comand, the robot keeps track of its own direction in order to turn the correct amount of degrees to end up facing a different direction, represented by the states in the diagram:



Instruction Syntax:

Valid syntax for the **instructions.asm** file

Move instruction:

"mov" keyword followed by a comma separated value

```
mov,2
```

Turn instruction:

"turn" keyword followed comma and valid value: (90,180, 270 or 360)

```
turn, 180
```

Usage:

1. Write instructions into the instructions.asm file with valid syntax
2. Run CPU.py file

Run example:

```
mov, 3
turn, 90
mov, 2
```

```
Robot final position: [2, 3]
Direction: 180
- - - - 0 0 0 0 0 0
0 0 0 - 0 0 0 0 0 0
0 0 0 x 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Lex analyzer

Problem description:

In order for a human operator to use the robot, a language that is more atuned with regular human speech is necesary.

In order to achieve this, a series of tokens that are used by a lexer in order to understand some common words is required.

Accepted Keywords

Keywords accepted and translated to tokens:

- ⟨**NOUN**⟩ → "robot" | "gerald"
- ⟨**KIND_WORD**⟩ → "please" | "kindly"

- ⟨**BLOCKS**⟩ → "block" | "blocks"
- ⟨**DEGREES**⟩ → "degrees"
- ⟨**CONJUNCTION**⟩ → "and"
- ⟨**ADVERB**⟩ → "then" | "subsequently"|"after"|"afterwards"|"next"
- ⟨**POSITION**⟩ → "move" | "advance"
- ⟨**ORIENTATION**⟩ → "turn" | "rotate"
- ⟨**ANGLE**⟩ → "90" |"180"|"270" | "360"
- ⟨**DIRECTION**⟩ → "ahead" | "left"|"right"| "up" |"down"

Run example:

```
robot please move 3 blocks ahead and then turn 90 degrees, then move forward 5
blocks and turn 90 degrees
```

```
ds\Lex_Yacc> ./program text.txt
PASS
```

```
robot moves 2 blocks quickly
```

```
ds\Lex_Yacc> ./program text.txt
FAIL
```

YACC grammar

Problem description:

After translating to tokens, now the job of the parser is to take those tokens and write into the *.asm* file the valid syntax for the CPU to use.

Context Free Grammar

```
⟨STATEMENT_LIST⟩ → ⟨STATEMENT⟩ | ⟨STATEMENT_LIST⟩ ⟨STATEMENT⟩
⟨STATEMENT⟩ → ⟨NOUN_PHRASE⟩  ⟨ROBOT_COMMAND⟩
⟨ROBOT_COMMAND⟩ → ⟨ACTION⟩ | ⟨ACTION⟩  ⟨CONJUNCTION⟩ ⟨ACTION⟩ |⟨ACTION⟩ ⟨CONJUNCTION⟩
⟨ADVERB⟩ ⟨ACTION⟩
⟨NOUN_PHRASE⟩ →  ⟨NOUN⟩  ⟨KIND_WORD⟩
⟨ACTION⟩ →  ⟨MOVEMENT⟩ | ⟨ROTATION⟩ |  ⟨ACTION⟩ ⟨ADVERB⟩ ⟨ACTION⟩ |⟨ACTION⟩ ⟨CONJUNCTION⟩
⟨ACTION⟩
⟨MOVEMENT⟩ → ⟨POSITION⟩ ⟨NUMBER⟩ ⟨BLOCKS⟩ ⟨DIRECTION⟩ | ⟨POSITION⟩ ⟨NUMBER⟩ ⟨BLOCKS⟩ |
⟨POSITION⟩ ⟨BLOCKS⟩ ⟨NUMBER⟩ ⟨DIRECTION⟩
⟨ROTATION⟩ → ⟨ORIENTATION⟩ ⟨ANGLE⟩ ⟨DEGREES⟩ | ORIENTATION⟩ ⟨DIRECTION⟩
```

Valid Sentences

With the tokens already defined, examples of valid sentences are as follows:

```
    * Robot please move 2 blocks ahead
    * Robot please move 3 blocks ahead and then turn 90 degrees, then move 2
blocks
```

Examples of invalid sentences:

```
    * Robot moves 2 blocks
    * Robot moves 2 blocks quickly
    * Move 2 blocks right now
    * Robot  2 blocks moves
    * Moves Robot 2 blocks and turns 89 degrees
```

Run example:

Input:

```
robot please move 3 blocks ahead and then turn 90 degrees, then move 2 blocks
```

OUTPUT instructions.asm: