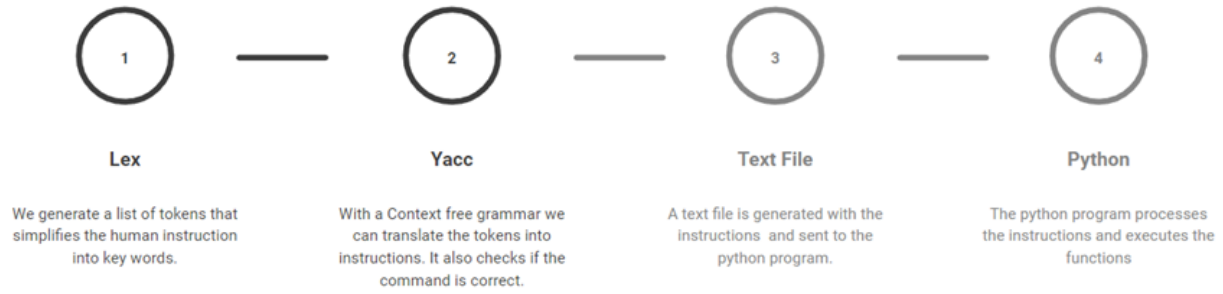
Deliverable I: Basic CPU

In the last century, efficiency has become a crucial concept in the manufacturing field; automatization is everywhere, with an increasing need for specialized equipment. That's where robots come in.

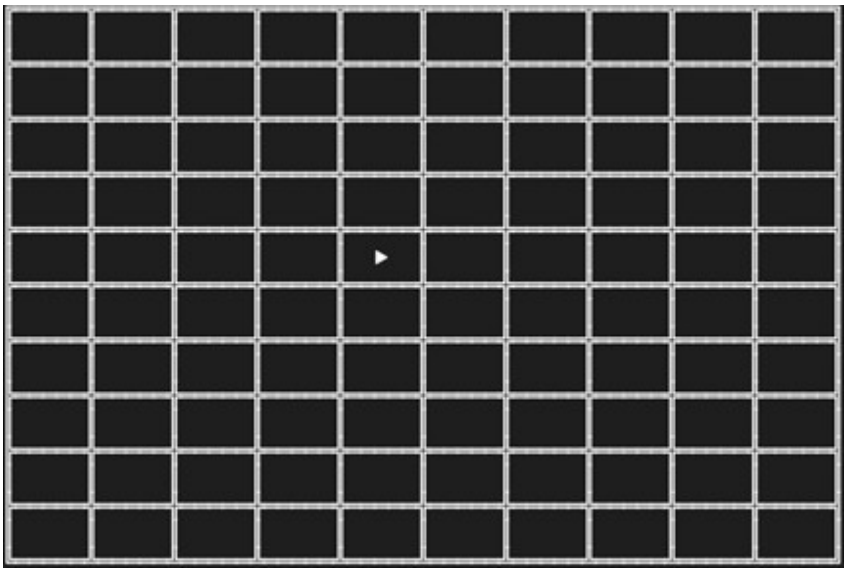
Programming and designing a machine capable of realizing tasks previously made by humans can both decrease manufacture time and increase product quality. Therefore, our mission is to design a compiler capable of processing human instructions, convert them into code that is understandable for the robot, and finally implement the necessary logic for the robot to execute its duty.

To generate a compiler we need to implement different files that interact with each other simplifying human instructions.



First, it is required to define the robot's characteristics and limitations as it is shown in the following list:

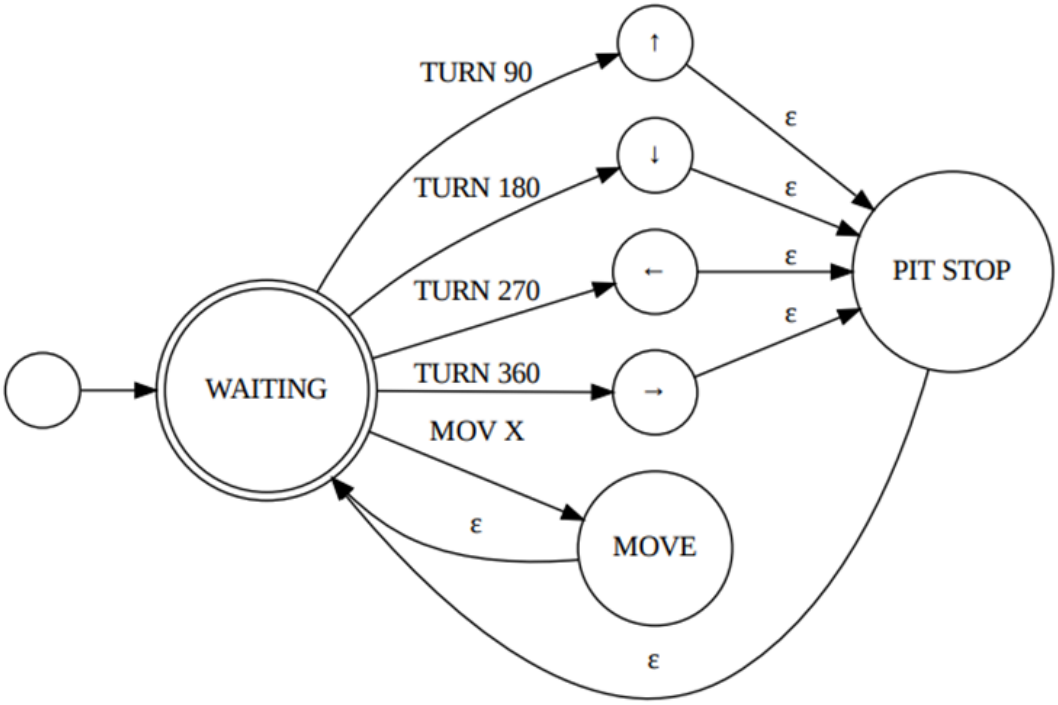
1. The robot is capable of moving only **forward**
2. The robot is capable of **rotation** (90,180,270,360)
3. The robot only understands the following commands:
    1. **MOV X** (where X represents any integer)
    2. **TURN** (90,180,270,360)
4. The robot operates inside a **10 x 10** matrix
5. The robot only follows **polite** instructions



After reviewing the specifications, we were able to generate a ***fine automata***, which dictates the rules and input for the robot.

Machine States

Fine Automata	**Rules**
States $\boldsymbol{Q}$	WAITING, ←, ↑, →, ↓, PIT STOP, MOVE
Alphabet ∑	MOV X, TURN (90,180,270,360), EMPTY
Transition function	Shown in diagram and table
Start State q - >$\boldsymbol{Q}$	WAITING
Accept State F -> $\boldsymbol{Q}$	WAITING



Transition function

STATES	**MOV X**	**TURN 90**	**TURN 180**	**TURN 270**	**TURN 360**	**EMPTY**
WAITING	MOVE	↑	←	↓	→	-
←	-	-	-	-	-	PIT STOP
↑	-	-	-	-	-	PIT STOP
→	-	-	-	-	-	PIT STOP
↓	-	-	-	-	-	PIT STOP
PIT STOP	-	-	-	-	-	WAITING
MOVE	-	-	-	-	-	WAITING

Each state represents a function that the python file executes, with the exception of “WAITING” and “PIT STOP” which are intermediate states between instructions. Consequently, WAITING is the starting and accepting state.

On the other hand, the alphabet is a set of triggers (instructions) that initiate different functions. For example, if the robot is in the WAITING state and receives “MOV 3” it will provoke the MOVE function to start (with 3 as an argument), and then return to the WAITING state.

CPU Simulator

The CPU Simulator in Python works using a coordenates class. This object has overloaded operators that manage the operations with the different coordenates, and the corresponding limits of the matrix. The class uses an integer to determine the Robot's direction, and depending on its value, a different directional coordenate is chosen.

The script reads the instructions and split them by the commas (,). That way the CPU Simulator can determine the action it must do as well as the quantity of times. Each movement is generated and presented in a map with UNICODE characters, representing the current values, direction and position of the robot. If the script recieves a MOVE instruction that exceeds the matrix, the robot will remain in its last position.

Links

Github Main Project link

Github Deliverable I link