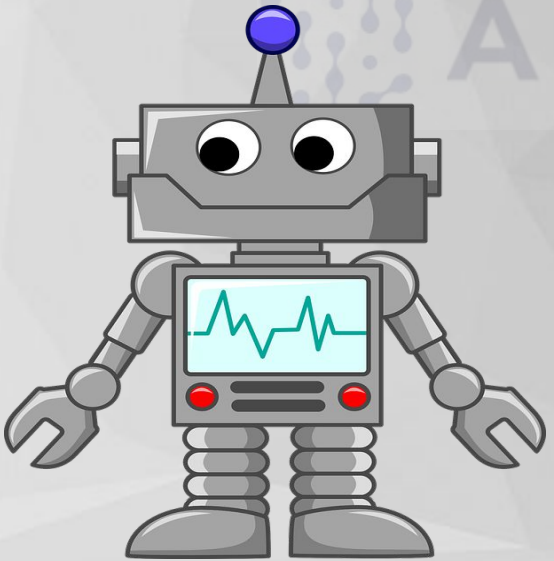
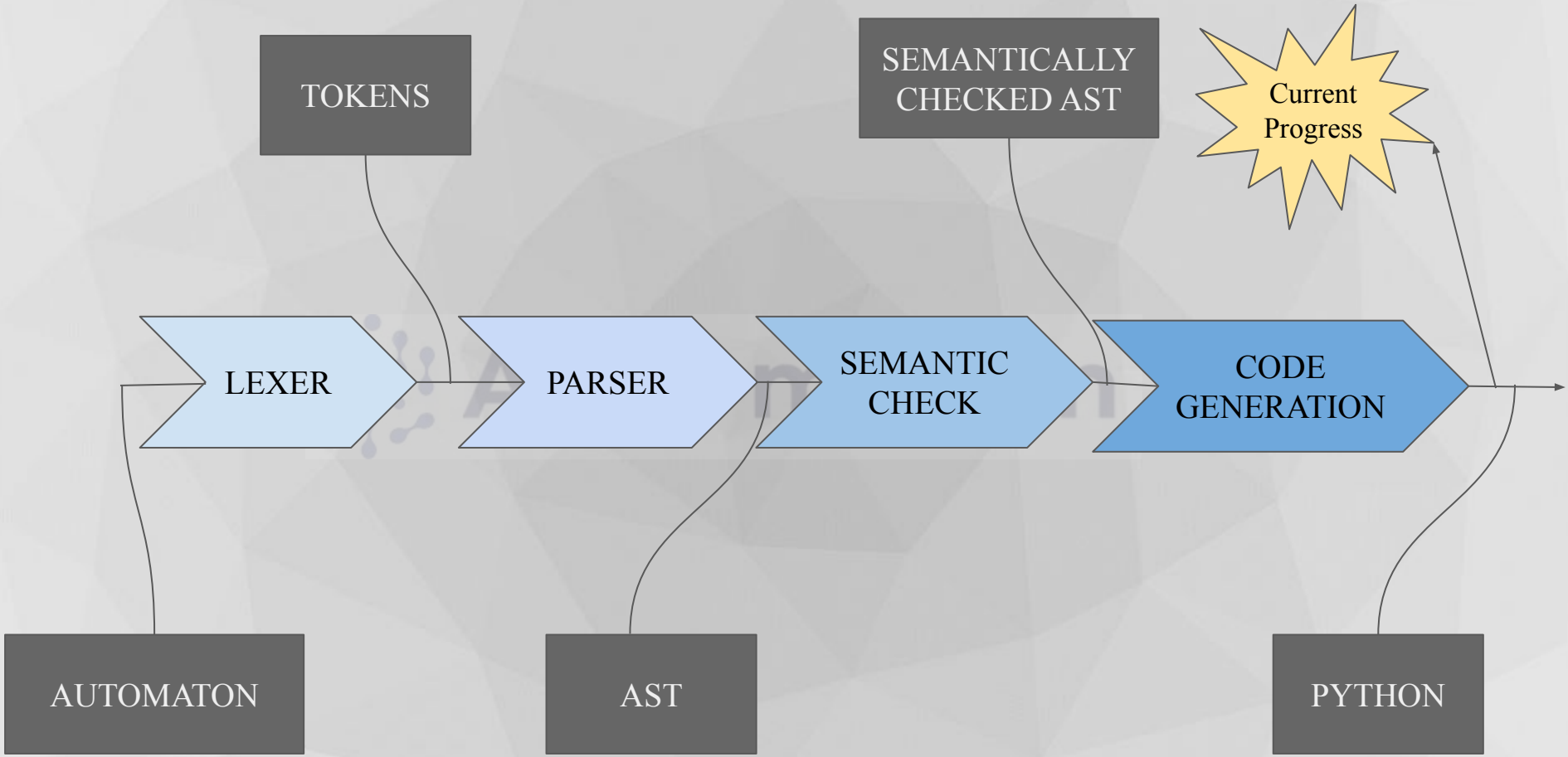
AUTOMATON



GROUP 6

V. Sathwik - CS19BTECH11022
D. Sarat Chandra Sai - CS19BTECH11003
Sushma Nalari - CS19BTECH11006
Pochetti Rashmitha - CS19BTECH11019
L. Harsha Vardhan Reddy - CS19BTECH11023
K. Lakshmi Sravya - CS19BTECH11017
K. Sri Teja - CS19BTECH11002


TOKENS
SEMANTICALLY CHECKED AST
Current Progress
LEXER
PARSER
SEMANTIC CHECK
CODE GENERATION
AUTOMATON
AST
PYTHON

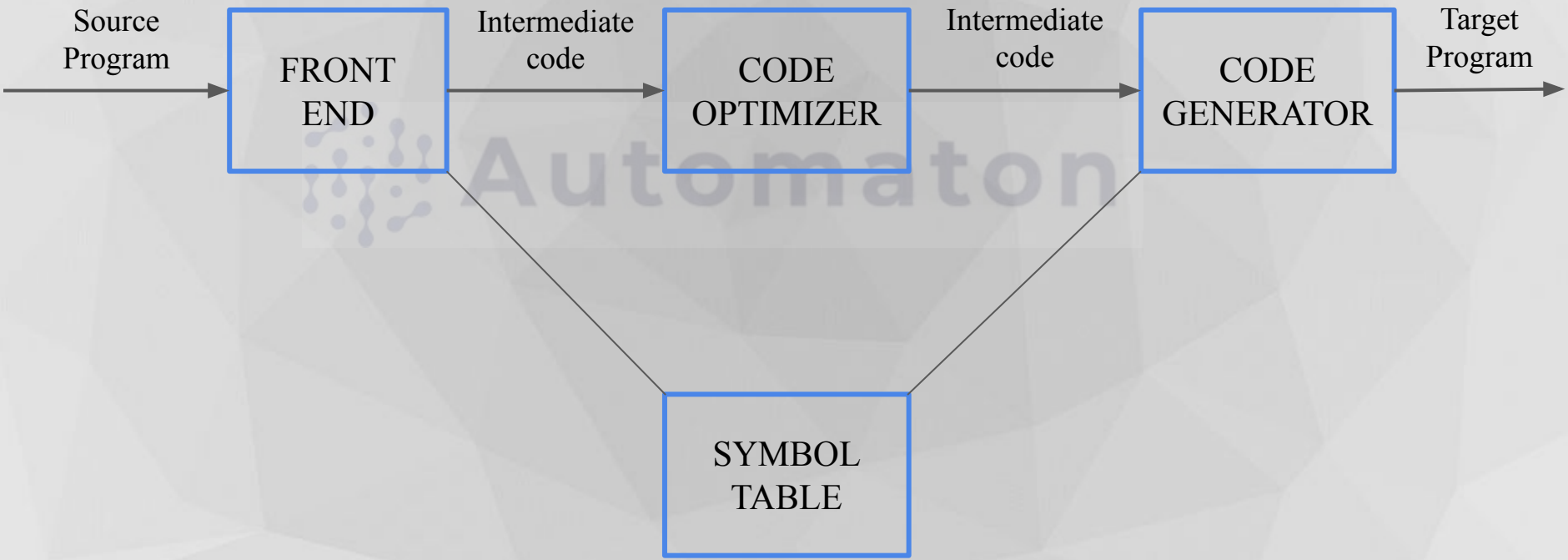
OPTIMIZATION

- Compilers that aim to produce efficient target programs would have an optimization phase prior to code generation
- The optimizer maps the IR into IR from which more efficient code can be generated.
- In general, the code optimization and code generation phases of a compiler (which are often referred as the back end) may make multiple passes over the IR before generating the target program.

CODE GENERATION

- Let us first talk about the requirements that are expected from a code generator. Generally, these expectations are high and include the following:
  - ➢ The semantic meaning of the source program must be preserved in the target program
  - ➢ Target program must be of high quality making effective use of all the available resources of the target machine.
- Now talking about the working of Code Generator. It takes input from the IR(intermediate representation) with supplementary information in symbol table of the source program and produces an equivalent target program as Output.

POSITION OF CODE GENERATOR



CODE GENERATION

- The work of Code generator is to take the semantically checked program and convert it into Python code.
- It converts
  - ➢ DFA into a class
  - ➢ States into class methods.
- It also generates a fair amount of pre-established pure python code that is used to do built-in language functions (wrapped as DFAs), as well as set up the architecture for the main function to run itself (all other functions are run/managed in python by the DFA that calls them).

CODE GENERATOR

- 3 Primary tasks of Code Generator are
  - ➢ Instruction Selection
  - ➢ Register Allocation and Assignment
  - ➢ Instruction Ordering
- Instruction Selection: The task is to choose appropriate target-machine instructions to implement the IR statements.
- Register Allocation and Assignment: The task is to decide what values need to be kept in which registers.
- Instruction Ordering: The task involves decision making in which we need to decide the order in which the execution of instructions should be scheduled.

IMPLEMENTATION

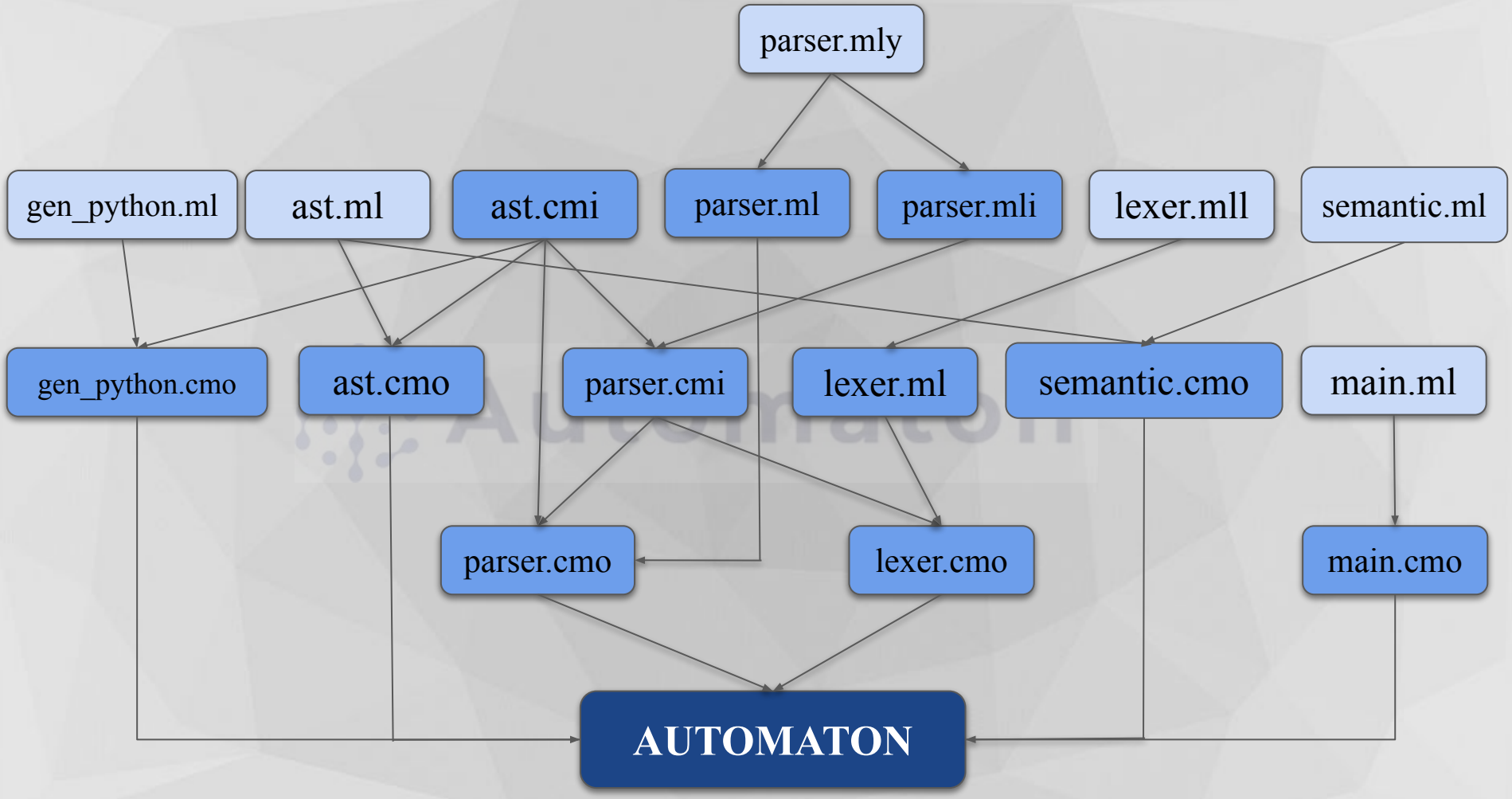
- Using ocamlc -c comand with gen_python.ml file and ast.cmi, gen_python.cmi file will be generated.
- After running "make " to produce the compiler executable, we compile our .sm file with the following Command: `$ ./gen_python "name of output file" < "path to your .sm file"`
- This compiles our Automaton program and produce python code named "name of output file".py.
- We then run this file using the command: `$ python "name of output file".py`
- Stacks can be passed in a command line by separation via commas.
- No spaces should exist between the elements of a stack.
- To pass in a string as a stack of strings, with each string consisting as a single character of the string, surround the string to be passed with “’ (double quotes then single quotes).

`python outputName.py a,b,c || python outputName.py[a,b,c] || python outputName.py “’bitbybit’”`

HOW TO COMPILE AND RUN

- The folder “Code generator” consists of the following files:
  - lexer.mll
  - parser.mll
  - ast.ml
  - semantic.ml
  - gen_python.ml
  - main.ml
  - input.am
  - makefile
- “make” command is used to run these files and generate the required output.

  (Refer to the next slide for better understanding)


parser.mly
gen_python.ml
ast.ml
ast.cmi
parser.ml
parser.mli
lexer.mll
semantic.ml
gen_python.cmo
ast.cmo
parser.cmi
lexer.ml
semantic.cmo
main.ml
parser.cmo
lexer.cmo
main.cmo
AUTOMATON

```
// Testing return types
int DFA get_int()
{
    Start
    {
        return 10;
    }
}

float DFA get_float()
{
    Start
    {
        return 10.0;
    }
}


void DFA main()
{
    Start
    {
        int x = get_int();
        print(x);
        y = get_float();
        print(y);
        return;
    }
}
```

EXPECTED OUTPUT

10
10.0

THANK YOU!!!