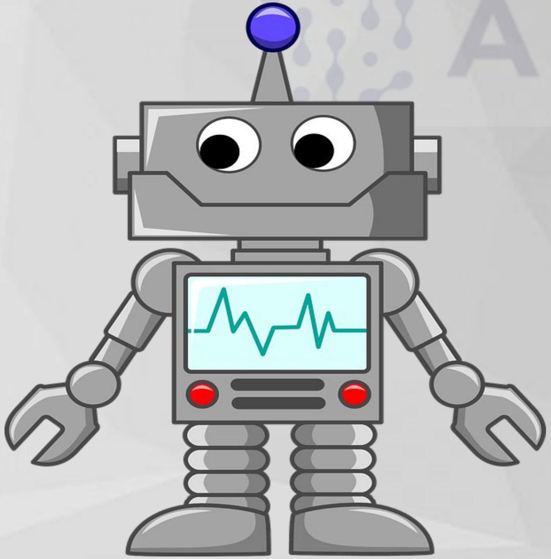
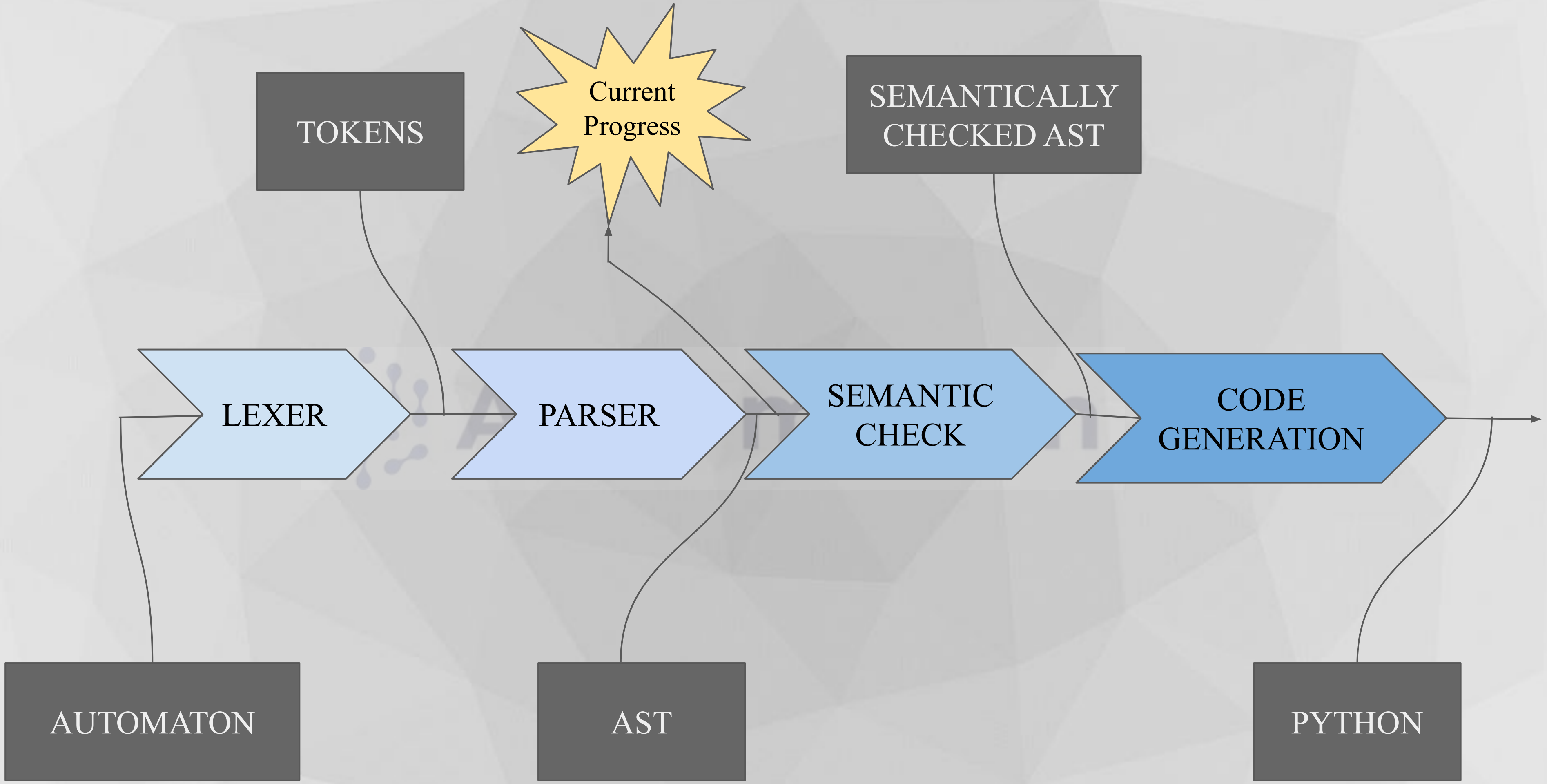
AUTOMATON



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

GRAMMAR

- Just like people require grammar to communicate effectively, a language requires one.
- A language cannot function without grammar.
- Grammar of a language governs the words, sentences, along with their interpretation and combinations, basically the rules of a language.
- Syntax grammar is a set of rules that describes the ordering of symbols in the language.
- A program which is generally represented as a sequence of ASCII characters is transformed into a syntax tree using grammar.

PARSER

- PARSER is used for checking/verifying the grammar of our source language.
- An ABSTRACT SYNTAX TREE is a representation of our source code that conveys the structure of our source code.
- Each node in the syntax tree represents a construct occuring in the source code.
- Parser tree can be built using the techniques universal parsing, top-down parsing, and bottom-up parsing.
- Parsing functions take a lexical analyze and a lexer buffer as arguments, which is a function from lexer buffers to tokens and return the semantic attribute of the corresponding entry point.
- Our parser is built using OCAMLYACC.

Ocamlyacc

- The *ocamlyacc* command is used to generate the parser for a given context free grammar. If we execute *ocamlyacc options parser.mly*, it will produce ocaml code for the parser in *parser.ml* file and its interface in the file *parser.mli*.
- The generated *parser.ml* file consists of functions for each of the entry points present in *parser.mly* file. These functions take a lexical analyzer and a lexical buffer as arguments and return the semantic attributes.
- Lexical analyzer functions are generated from ocaml programs of lexer files. Tokens are defined in the interface file *parser.mli*.

STRUCTURE OF .mly FILE

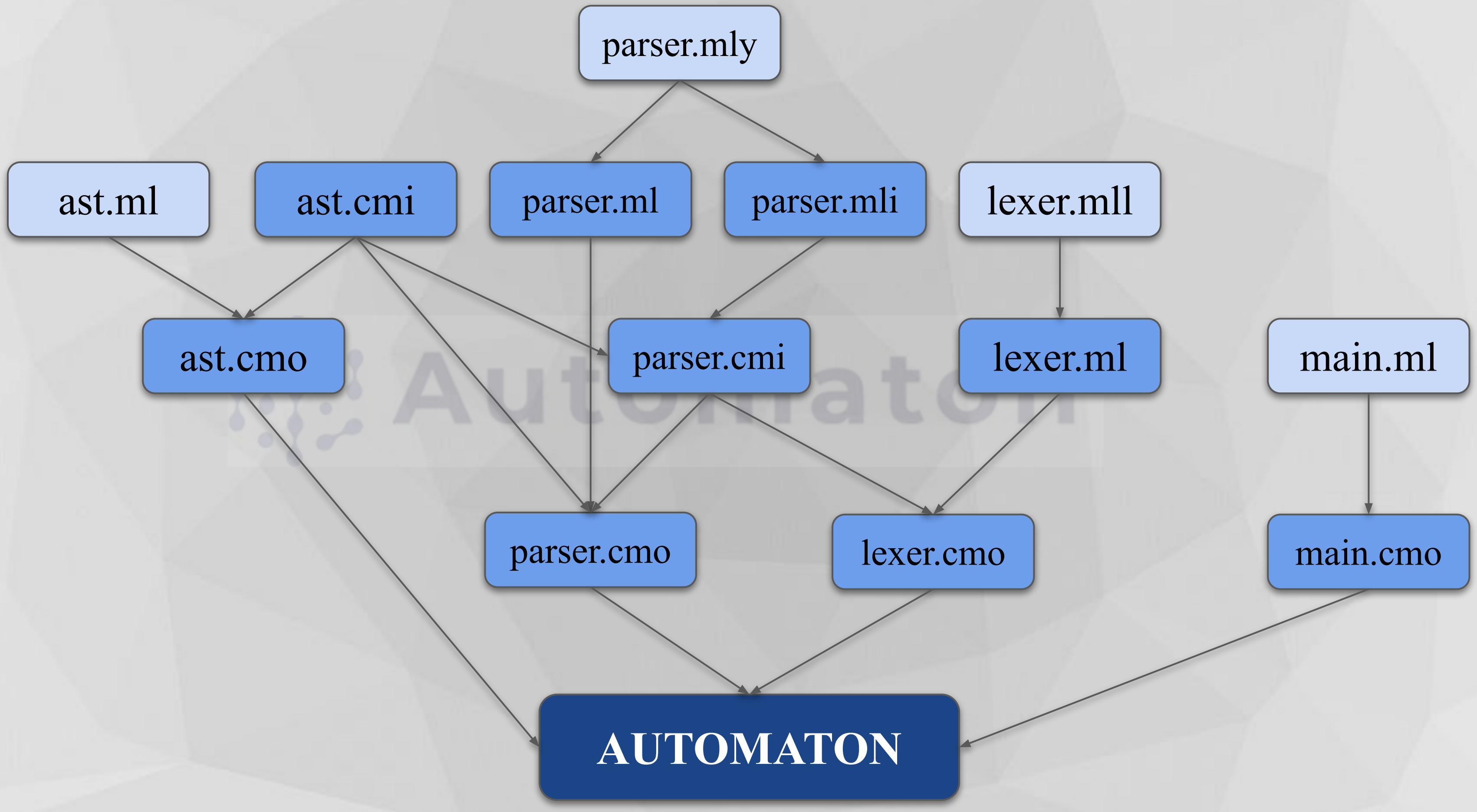
The parser.mly file consists of four parts

- Header: The header part contains open directives and auxiliary functions. These sections of code will be copied as is into ml file.
- Declarations: Declarations are given one per line which starts with % sign.
  - ➢ %tokens declares the constructors as tokens with an attached attribute of the given type.
  - ➢ %start declares the given symbol as entry point for the grammar.start symbol must be given a type with the %type directive below.
  - ➢ It also has %left, %right, %nonassoc which are associative precedences and have associatives to the given symbol. Symbols will have high precedence which declared before these associatives and have low precedence which are declared after these associatives.
- Rules: Rules contain the %prec symbol to override the default precedence and associativity of the rule with the precedence and associativity of the given symbol.
- Trailer: The trailer part goes to the end of the output file. Trailer and header sections of code will be copied as is into ml file.

HOW TO COMPILE AND RUN

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

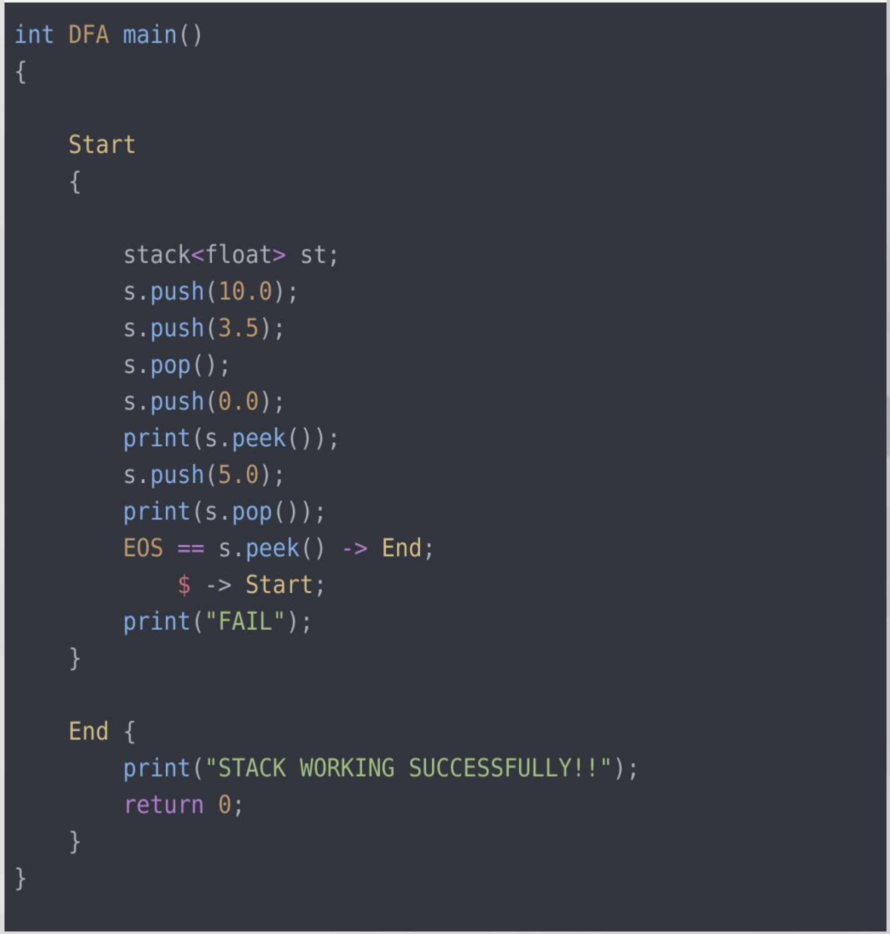
  (Refer to the next slide for better understanding)


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

IMPLEMENTATION

- The tokens of our source code are pre-defined in “lexer.mll”. Using Ocamllex, “lexer.ml” is generated from the above file.
- Context-free grammar specification is defined in “parser.mly”. Using Ocamlyacc, “parser.mli” and “parser.ml” are generated.
- “ast.ml” file is the result of the syntax analysis phase of the compiler. The structure of our program code is represented in this file.
- All of the above which are combination of lexer,parser, and ast generate object file named “parser”, which on executing gives out the names of tokens to stdout, taking input from stdin or “input.am”.

Basic Stack Operations :



```
int DFA main()
{

    Start
    {

        stack<float> st;
        s.push(10.0);
        s.push(3.5);
        s.pop();
        s.push(0.0);
        print(s.peek());
        s.push(5.0);
        print(s.pop());
        EOS == s.peek() -> End;
            $ -> Start;
        print("FAIL");
    }

    End {
        print("STACK WORKING SUCCESSFULLY!!");
        return 0;
    }
}
```

OUTPUT :

```
found int
found float
stack declared
variable declaration
found float
found push
found float
found push
found pop
found float
found push
found peek
found print statement
found float
found push
found pop
found print statement
found peek
found =
change in transition
default transition
found string
found print statement
found transition
found string
found print statement
found digit
found return statement
found transition
```

THANK YOU!!!