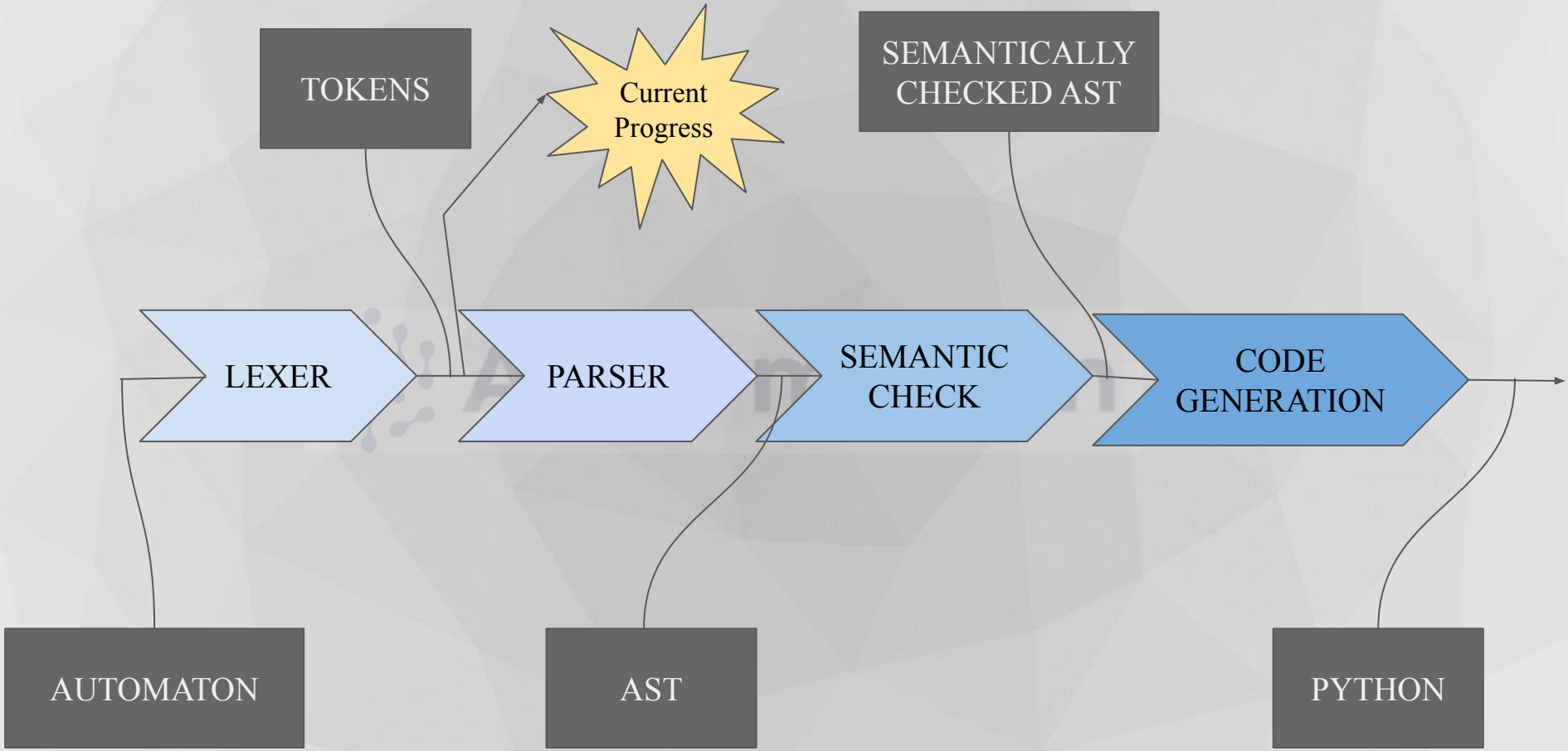
AUTOMATON

GROUP 6

V. Sathwik - CS19BTECH11022
D. Sarat Chandra Sai - CS19BTECH11003
Sushma Nalari - CS19BTECH11006
P. 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

LEXICAL ANALYSIS

- A single element of a programming language is considered as a **token**. It consists of constants, identifiers, separators, keywords, and operators.
- Lexical Analysis is used to tokenize a given source code. It returns all the valid tokens and mentions about the invalid tokens like invalid operators or invalid characters.
- It also removes the comments and white spaces.

LEXER

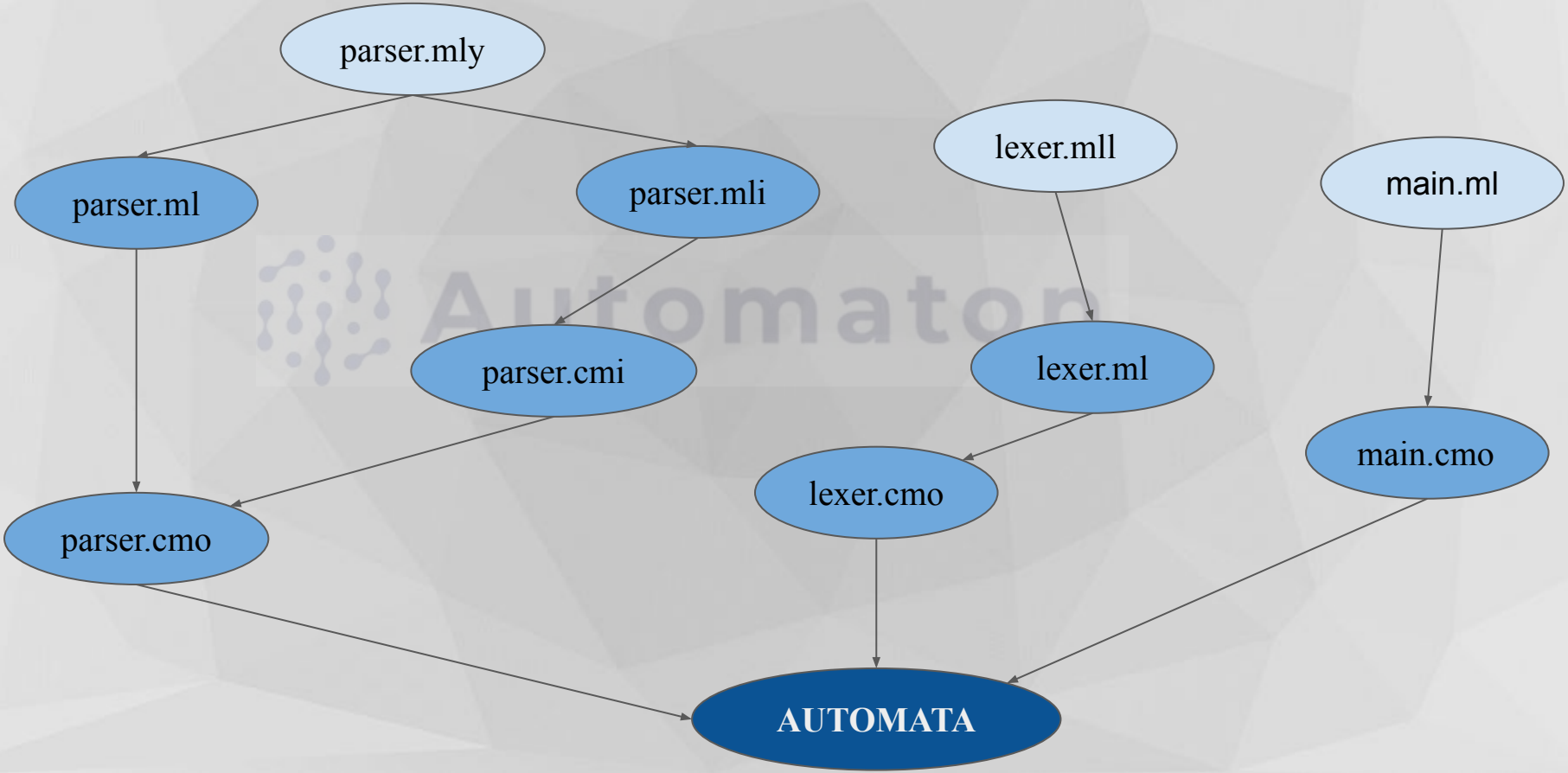
- LEXER is used for lexical analysis. The code for lexer is written using OCAML.
- “input.am” source code is used to take input stream for the lexer.
- The generated tokens are taken by the parser and an Abstract Syntax Tree is generated (will be used later).

HOW TO COMPILE AND RUN

- The folder “Lexer” consists of the following files:
  - lexer.mll
  - parser.mll
  - 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)

FILES USED AND GENERATED

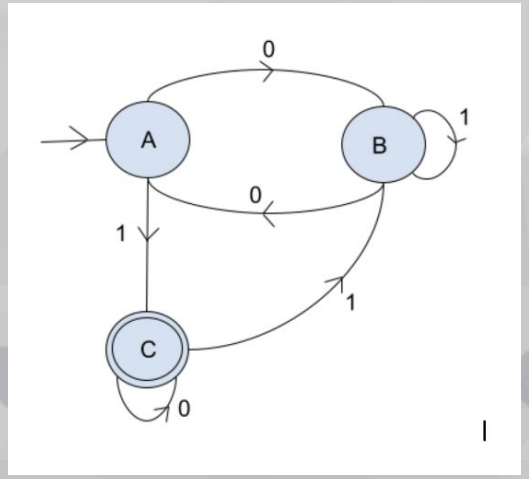


IMPLEMENTATION

- The tokens of our source code are pre-defined in “lexer.mll”. Using Ocamllex, “lexer.ml” is generated from the above file.
- The corresponding identifier/operator/keyword names are printed for a valid token.
- In case of invalid tokens, “invalid token” is returned.
- Without writing grammar, just empty fields are defined in “parser.mly”. Using Ocamlyacc, “parser.mli” and “parser.ml” are generated.
- All of the above which are combination of lexer and parser generate object file named “parser”, which on executing gives out the names of tokens to stdout, taking input from stdin or “input.am”.

Example : Checking whether a DFA accepts a string or not

```
int DFA main()
{
        int n = int(input()) #
        Start {
                $ -> A;
        }
        A {
                (i == n) -> Reject;
                (s[i] == 0 && i++) -> B;
                (s[i] == 1 && i++) -> C;
        }
        B {
                (i == n) -> Reject;
                (s[i] == 1 && i++) -> B;
                (s[i] == 0 && i++) -> A;
        }
        C {
                (i == n) -> Accept;;
                (s[i] == 1 && i++) -> B;
                (s[i] == 0 && i++) -> C;
        }
        Accept {
                print("Given string is accepted by the DFA");
                $->Exit;
        }
        Reject {
                print("GIven string is not accepted by the DFA");
                $->Exit;
        }
        Exit {
                return 0;
        }
}
```



LEXER OUTPUT:

INT
DFA
MAIN FUC
LPAREN
RPAREN
LBRACE
INT
ID
ASSIGN
INT
LPAREN
INPUT
LPAREN
RPAREN
RPAREN
INVALID TOKEN
SSTATE
LBRACE
DEFSTATE
ARROW
STATE
SEMICOLON
RBRACE
STATE
LBRACE
LPAREN
..
..
..
..
..

LESSONS LEARNT

- Exploring and Understanding the internal working of the compiler is new and exciting.
- We had to go through the documentations of different types of parsers/lexers for better understanding and then opted for OCAML.
- Ocaml was hard to understand when we began but seeking help from seniors was resourceful.
- Programming and Debugging in pairs is very productive and “NEVER UNDERESTIMATE THE POWER OF TEAMWORK”.
- Working with an effective mind for 30 mins is far better than struggling with an exhaustive mind for hours.

THANK YOU!!!