System Programming and Compiler Construction

MODULE 5 (4)

COMPILERS : ANALYSIS PHASE

Prof. Sonal Shroff
Computer Engineering Department
TSEC

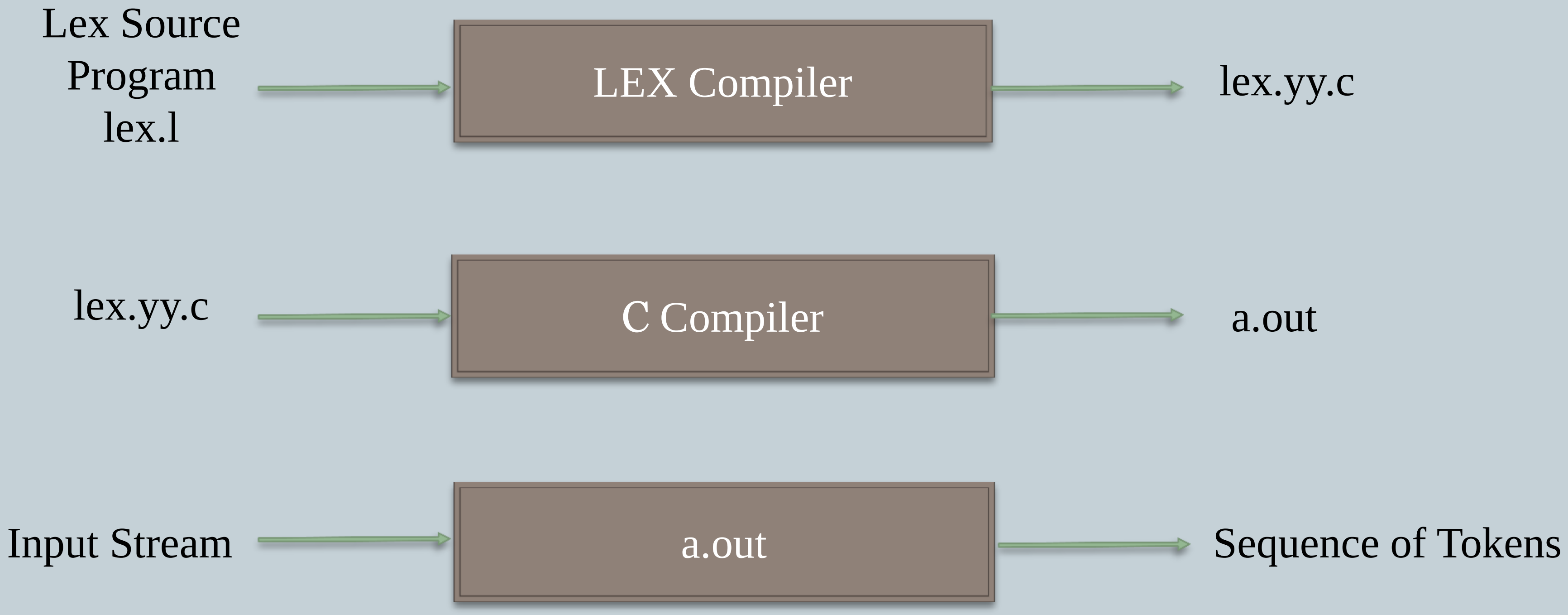
Lexical Analysis

➢ The Lexical-Analyzer Generator Lex

- Allows to specify a lexical analyzer by specifying regular expressions to describe patterns for tokens.
- The tool – Lex compiler
- The input notation – Lex language
- The Lex compiler transforms the input patterns into a transition diagram and generates code – lex.yy.c

Lexical Analysis

- The Lexical-Analyzer Generator Lex



Lex Source Program lex.l → LEX Compiler → lex.yy.c

lex.yy.c → C Compiler → a.out

Input Stream → a.out → Sequence of Tokens

Creating a lexical analyzer with Lex

Lexical Analysis

- The structure of lex program

Lex program consists of three parts:-

```
%{
Declaration
%}

%%
Translation rules
%%

Auxiliary functions
```

Lexical Analysis

- The structure of lex program

%{

Declaration

%}

The declarations section includes

- declarations of variables,
- manifest constants (identifiers declared to stand for a constant)
- regular definitions

Lexical Analysis

- The structure of lex program

%%

Translation rules

%%

Translation Rules are of the form:-

P1 {action 1}

P2 {action 2}

....

Pi {action i}

Lexical Analysis

- The structure of lex program

Pattern { Action }

- where each Pi is a regular expression and
- each action i is a program fragment describing what action the lexical analyzer should take when pattern Pi matches a lexeme

Lexical Analysis

- The structure of lex program

Auxiliary functions

- This section holds whatever auxiliary functions are used in the actions
- These functions can be compiled separately and loaded with the lexical analyzer

Lexical Analysis

- The structure of lex program

- When called by the parser, the lexical analyzer begins reading its remaining input, one character at a time
- It reads until it finds the longest prefix of the input that matches one of the patterns Pi.
- It then executes action Ai which returns the control to the parser
- But if it does not then the lexical analyzer proceeds to find additional lexemes, until one of the corresponding actions causes a return to the parser.

Lexical Analysis

➢ The structure of lex program

- The lexical analyzer returns a single value i.e. the token name to the parser.
- To pass an attribute value (additional information about the lexeme), a shared integer variable is set known as **yylval**

Lexical Analysis

- The lex program for the tokens

```
%{
 /* Definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP */
%}
```

Lexical Analysis

- The lex program for the tokens

```
/* Regular definitions */

delim [\t \n]

 ws {delim}+

 letter [A-Za-z]

 digit [0-9]

 id {letter}( {letter} | {digit} )*

 number {digit}+(\ . {digit} +) ? (E [+-] ? {digit}+) ?
```

Lexical Analysis

- The lex program for the tokens

```
%%
{ws}          { /* No action and No Return */ }
if           { return (IF); }
else         { return (ELSE); }
then         { return (THEN); }
{id}          { yylval = (int) installID(); return (ID); }
{number}    { yylval = (int) installNum(); return (NUMBER); }
```

Lexical Analysis

- The lex program for the tokens

```
“<”          { yylval = LT;  return(RELOP); }
“<=”         { yylval = LE;  return(RELOP); }
 “=”         {  yylval = EQ;  return(RELOP); }
“<>”         {  yylval = NE;  return(RELOP); }
“>”          { yylval = GT;  return(RELOP); }
“>=”         { yylval = GE;   return(RELOP); }
%%
```

Lexical Analysis

- The lex program for the tokens

```
int installID() {

/* Function to install the lexeme whose first character is pointed to by yytext,
   and whose length is yyleng, into the symbol table and return a pointer to
   Symbol Table

*/ }

int installNum() {

/* Similar to installID but puts numerical constants into a separate table */ }
```

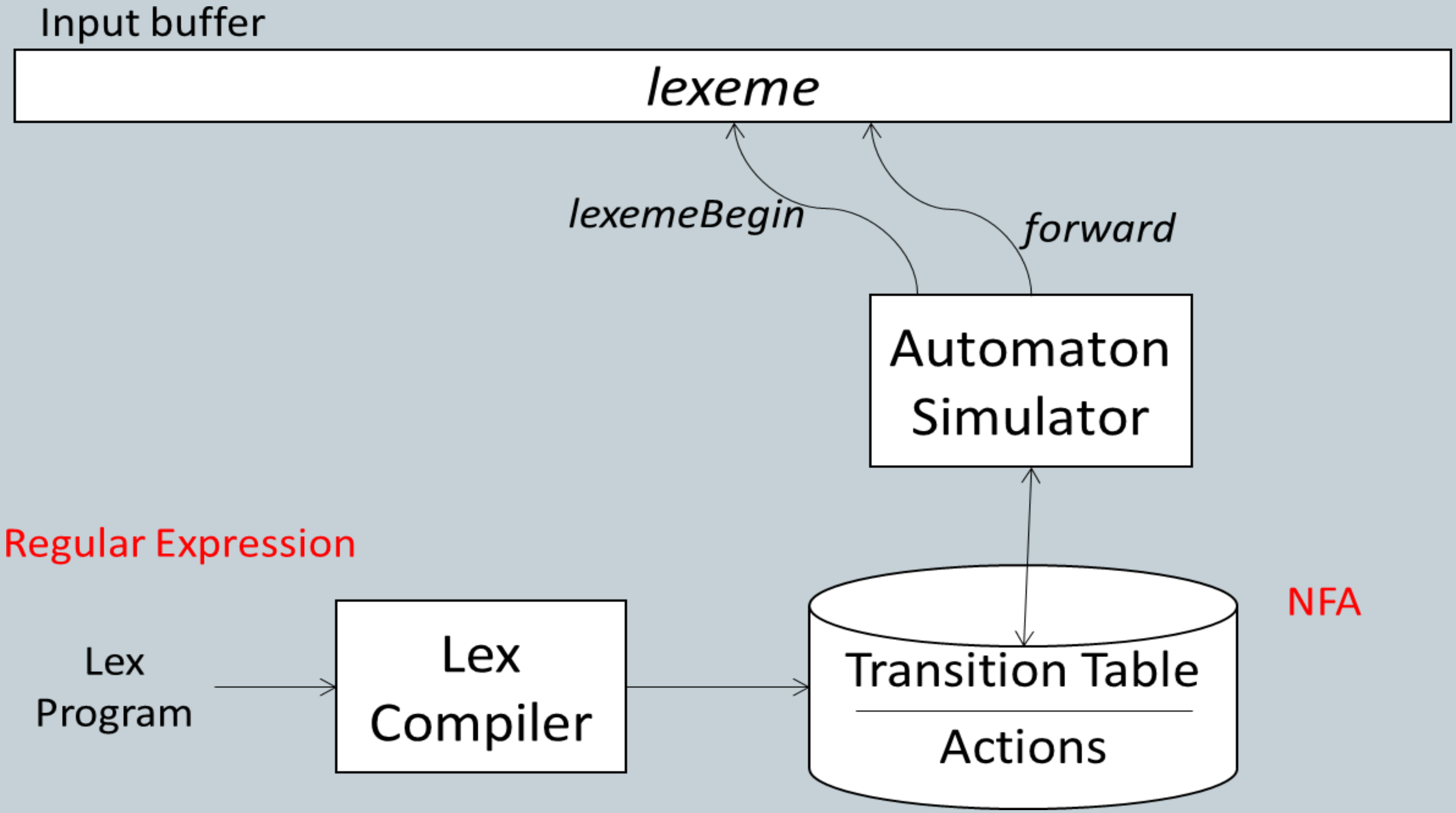
Lexical Analysis

- The lex program for the tokens

Example prog.

Lexical Analysis

- Design of a Lexical-Analyzer Generator



Lexical Analysis

- Design of a Lexical-Analyzer Generator

A Lex program is turned into a transition table and actions which are used by a finite automaton simulator

The program that serves as the lexical analyzer includes a fixed program that simulates an automaton

The rest of the lexical analyzer consists of following component

1. A transition table of the automaton
2. Those functions that are passed directly through Lex to the output
3. The action from the input program which is to be invoked at the appropriate time by Automaton Simulator

Lexical Analysis

- Design of a Lexical-Analyzer Generator

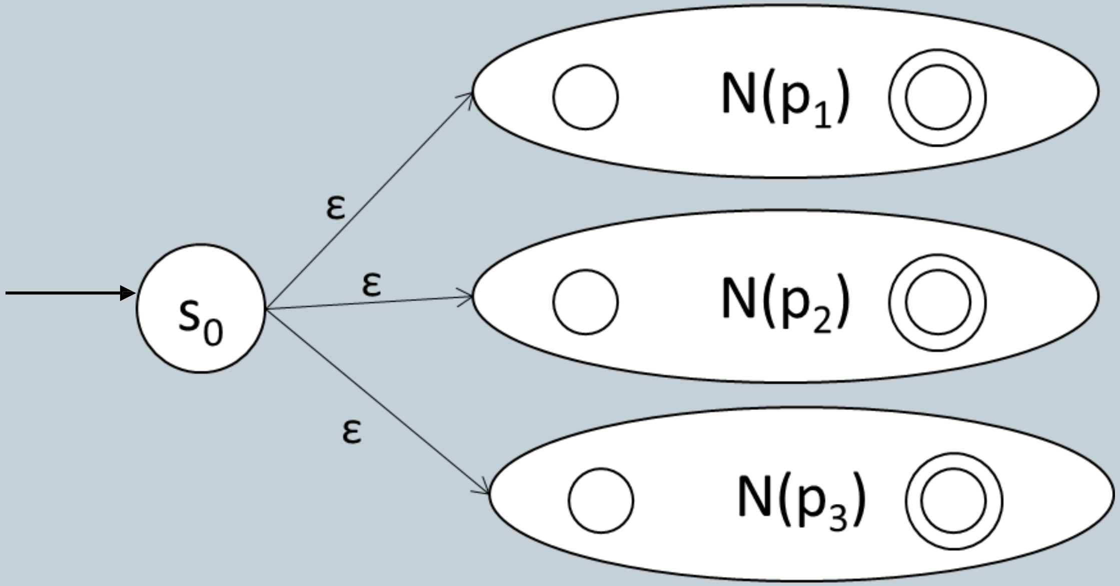
To construct the automaton, take each regular expression pattern in the Lex program and convert it using Algorithm to an NFA

We need a single automaton that will recognize lexemes matching any of the patterns in the program

Hence combine all the NFA s into one by introducing a new start state with ε -transitions to each of the start states of the NFA s Ni for pattern Pi

Lexical Analysis

- Design of a Lexical-Analyzer Generator



Lexical Analysis

Design of a Lexical-Analyzer Generator

Example :

- To identify the following patterns

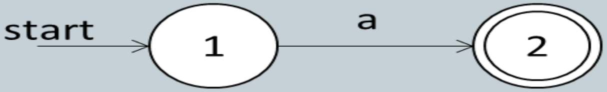
a	{action A_1 for pattern p_1}
abb	{action A_2 for pattern p_2}
a^*b^+	{action A_3 for pattern p_3}

Lexical Analysis

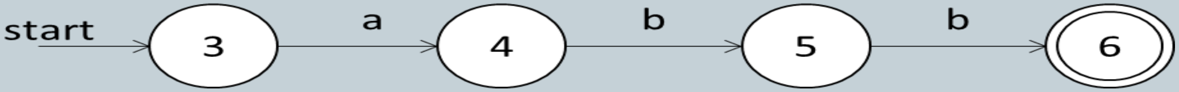
- Design of a Lexical-Analyzer Generator

Example :

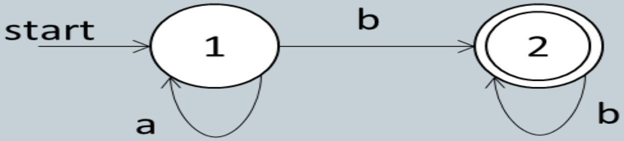
NFA for a



NFA for abb



NFA for a^*b^+

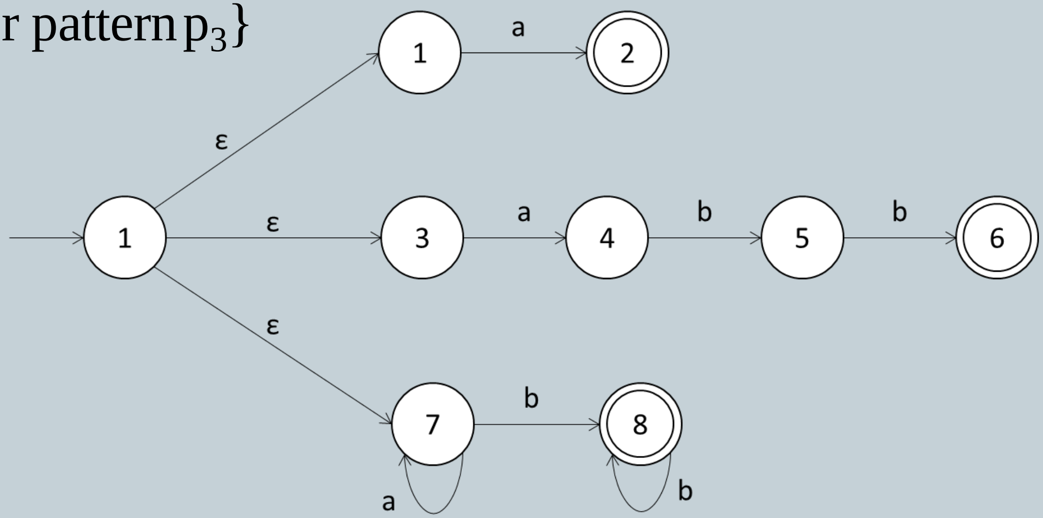


Design of a Lexical-Analyzer Generator

Example :

- To identify the following patterns
  - a {action A_1 for pattern p_1}
  - abb {action A_2 for pattern p_2}
  - a^*b^+ {action A_3 for pattern p_3}

Combined NFA :



Lexical Analysis

➢ Data Structures used in Lexical Analyzer

- Symbol: Identifier used in the Source Program
- Examples: Names of variables, functions and Procedures
- Symbol Table: To maintain information about attributes of symbol
- Operations on Symbol Table:
  1. Add a symbol and its attributes
  2. Locate a symbol’s entry
  3. Delete a symbol’s entry
  4. Access a symbol’s entry

Lexical Analysis

➢ Data Structures used in Lexical Analyzer

Design Goal of Symbol Table

1. The Table's organization should facilitate efficient search
2. Table should be compact (Less Memory)

Time Space Trade off

To improve search efficiency, allocate more memory to symbol table

Organization of Entries:

1. Linear Data Structure
2. Non-Linear Data Structure

Lexical Analysis

- Data Structures used in Lexical Analyzer

Symbol Table Entry Format

- Number of Fields to accommodate attributes of one symbol
- Symbol Field: The symbol to be stored
- Key Field: Basis for the search in table
- Entry of Symbol is called record
- Each entry can be of type fixed length, variable length or hybrid

Lexical Analysis

- Data Structures used in Lexical Analyzer

Symbol Class	Attributes
Variable	Type, Length, number and bounds of dimensions
Procedure	Number of parameters, address of parameter list
Function	Number of parameters, address of parameter list, type and length of returned value
Label	Statement Number