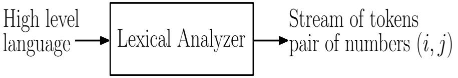
Lecture 2 - Lexical Analysis - Introduction
Compiler Design (CS 3007)

Sumanta Pyne

Assistant Professor
Computer Science and Engineering Department
National Institute of Technology Rourkela
pynes@nitrkl.ac.in

July 30, 2024

Function of lexical analyzer



i: type of token
j: number in the class of token

- Read source program, one character at a time
- Translate a sequence of charaters to tokens (keywords, identifiers, constants, operators)
- Symbol table entry for new identifiers
- Forward tokens to parser
- Report errors like invalid identifier, undefined identifier, multiple declaration, invalid operators, keyword declared as identifier, etc

Function of lexical analyzer

Keywords (token_type 1)

#1	**if**
#2	**then**
#3	**else**
#4	**while**
	⋮

Arithmetic Operators (token_type 2)

#1	$+$
#2	$-$
#3	$*$
#4	$/$
	⋮

Relational Operators (token_type 3)

#1	$<$
#2	$>$
#3	$==$
#4	$!=$
	⋮

Assignment Operators (token_type 4)

#1	$=$
#2	$+=$
#3	$-=$
#4	$*=$
	⋮

Identifiers (token_type 5)

#1	variable
#2	function
#3	array
#4	structure
	⋮

Token separators (token_type 6)

#1	;
#2	blank
#3	’
#4	”
	⋮

if $a < b$ **then** $a = b + c;$ → Lexical Analyzer → Stream of tokens pair of numbers (i, j)

i: type of token
j: number in the class of token

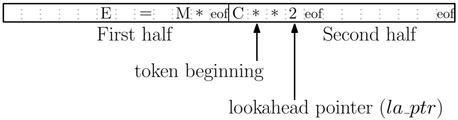
Token	(i, j)
if	$(1, 1)$
blank	$(6, 2)$
a	$(5, 1)$
$<$	$(3, 1)$
b	$(5, 1)$
blank	$(6, 2)$
then	$(1, 2)$
$=$	$(4, 1)$
$+$	$(2, 1)$
c	$(5, 1)$
;	$(6, 1)$

Symbol Table

	Name	Type	Number	⋯
#1	a	5	1	
#2	b	5	1	
#3	c	5	1	
	⋮	⋮	⋮	⋮

Input Buffering

Input buffer




```
la_ptr ← la_ptr + 1;
if la_ptr = eof then begin
   if la_ptr at the end of first half then begin
      reload second half;
      la_ptr ← la_ptr + 1;
   end
   else if la_ptr at the end of second half then begin
      reload first half;
      move forward la_ptr to the beginning of first half
   end
   else
      /*eof within a buffer signifying end of input*/
      terminate lexical analysis
end if
```

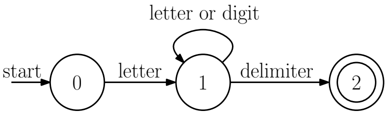
Lookahead code with sentinels

- Lexical analyzer read characters from input buffer
- A pointer marks beginning of token being discovered
- *la_ptr* scans from beginning until the token is discovered

A simple lexical analyzer for identifier

identifier = letter (letter or digit)*

letter = {A-Z,a-z}   digit = {0-9}



Transition diagram for identifier

```
state 0: C ← GETCHAR( );
         if LETTER(C) then goto state 1;
         else FAIL( );
state 1: C ← GETCHAR( );
         if LETTER(C) or DIGIT(C) then goto state 1;
         else if DELIMITER(C) then goto state 2;
         else FAIL( );
state 2: C ← RETRACT( );
         return (id, INSTALL( ));
```

Code for each state in the transition diagram

- GETCHAR() - advances the lookahead pointer and returns the next character
- LETTER(C) - returns **true** if C is a letter
- DIGIT(C) - returns **true** if C is a digit
- DILIMITER(C) - returns **true** if C is a character that can follow an identifier
- INSTALL() - returns a value that is a pointer to the symbol table
- FAIL() - returns error for invalid iput
- RETRACT() - retracts delimiter from the identifier

Designing a lexical analyzer

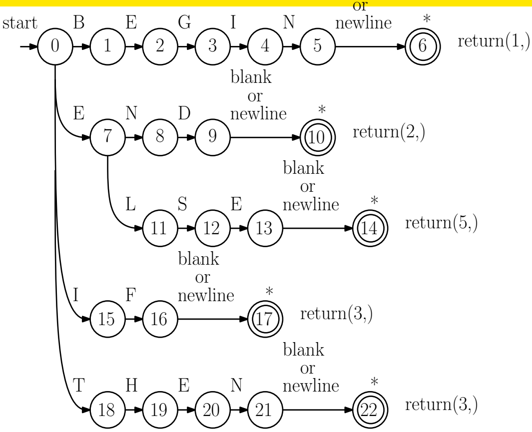
Token	Code	Value
begin	1	–
end	2	–
if	3	–
then	4	–
else	5	–
identifier	6	Pointer to symbol table
constant	7	Pointer to symbol table
$<$	8	1
$<=$	8	2
$=$	8	3
$<>$	8	4
$>$	8	5
$>=$	8	6

Tokens recognized

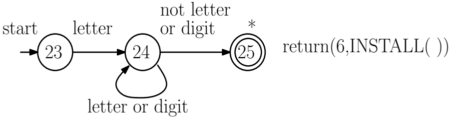
- Defining a token set
- Encoding each class of tokens
- Mapping values to tokens of same class

Transition diagrams for token recognizers

keywords:

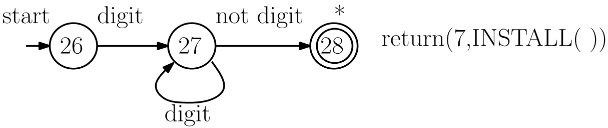


identifier:

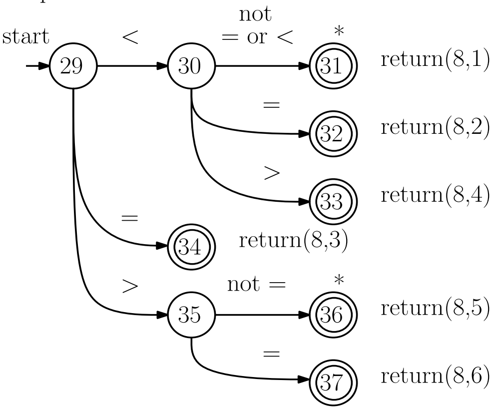


Transition diagrams for token recognizers

constant:



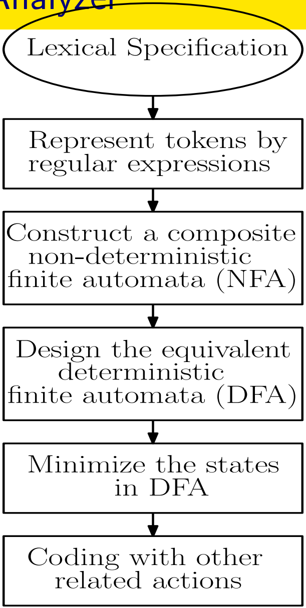
relops:



Combining the token recognizers

- Union of FSMs for identifiers, keywords, constants and relops
- A complex task
- Non-determinism may exist in the combined FSMs
- New transistions to be included
- Follow a proper procedure - simpler and error free
- Combine the token recognizers as an non-dertiministic FSM
- Then obain the equivalent dertiministic FSM
- Minimize the states
- Add actions

Implementation of Lexical Analyzer

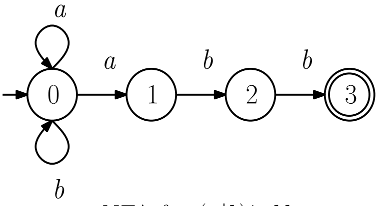


Lexical Specification and Regular Expressions

- An alphabet Σ is a finite set of symbols
- A string w is a finite sequence symbols, i.e. $w \in \Sigma^*$
- A language L is a set of strings over alphabet Σ, i.e. $L = \{w | w \in \Sigma^*\}$
- Regular expressions (REs) over alphabet Σ
  1. ϵ is a RE denoting language $L = \{\epsilon\}$, where ϵ is an empty string.
  2. For each $a \in \Sigma$, a is a RE denoting language $L = \{a\}$.
  3. If R and S are REs denoting languages L_R and L_S, respectively, then
     - i.) $(R)|(S)$ is a RE denoting $L_R \cup L_S$.
     - ii.) $(R) \cdot (S)$ is a RE denoting $L_R \cdot L_S$.
     - iii.) $(R)^*$ is a RE denoting L_R^*.
- Example: following tokens are described by REs
  - keyword = BEGIN | END | IF | THEN | ELSE
  - identifier = letter (letter | digit)*
  - constant = digit$^+$
  - relop = < | <= | = | <> | > | >=

Finite Automata

- A recognizer for a language L.
- Produces YES if an input string $w \in L$, otherwise NO.
- Consider a language $L = \{w | w \in \{a, b\}^* \wedge w \text{ ends with } abb\}$
- Regular expression for L is $R = R_1 \cdot R_2 = (a|b)^* abb$
- A simple recognizer for L is non-deterministic finite automata (NFA)



NFA for $(a|b)^* abb$

State	Input Symbol a	Input Symbol b
0	$\{0, 1\}$	$\{0\}$
1	–	$\{2\}$
2	–	$\{3\}$

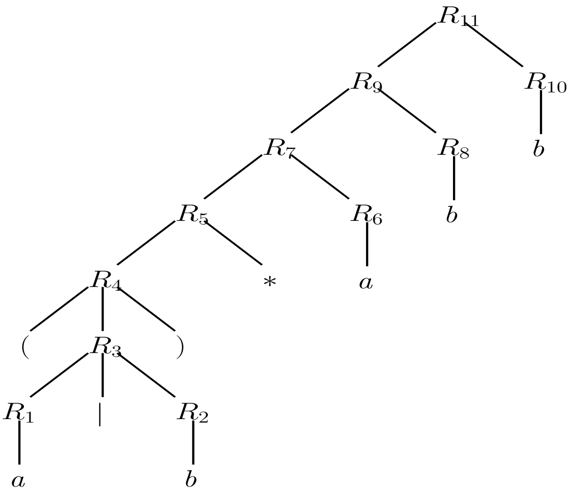
Transition table

State	Remaining Input
0	$aabb$
0	abb
1	bb
2	b
3	ϵ

Accepting $w = aabb$

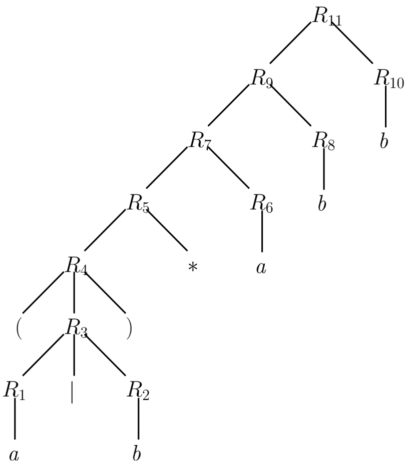
- Non-determinism cannot be implemented
- Convert NFA to deterministic finite automata (DFA)

Constructing NFA from Regular expression

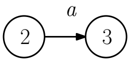


Decomposition tree for $(a|b)^*abb$

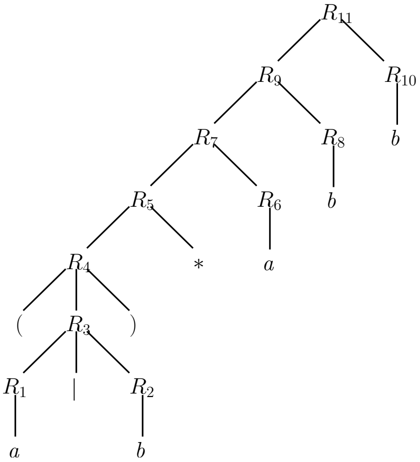
Constructing NFA from Regular expression



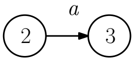
For R_1, N_1 =



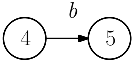
Constructing NFA from Regular expression



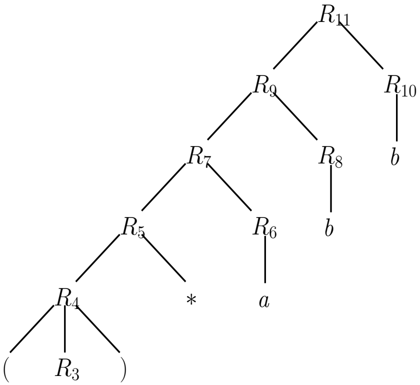
For $R_1, N_1 =$



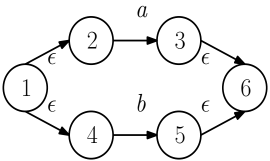
For $R_2, N_2 =$



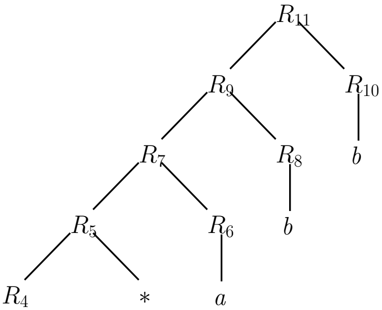
Constructing NFA from Regular expression



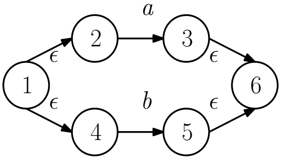
For $R_3 = R_1|R_2$, $N_3 =$



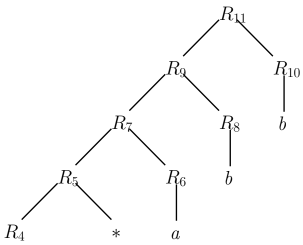
Constructing NFA from Regular expression



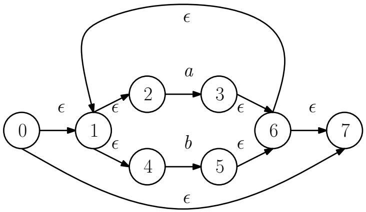
For $R_4 = (R_3)$, $N_4 = N_3 =$



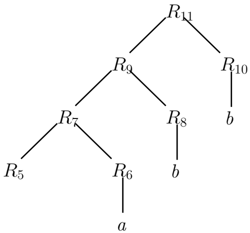
Constructing NFA from Regular expression

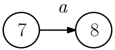


For $R_5 = R_4^*$, $N_5 =$

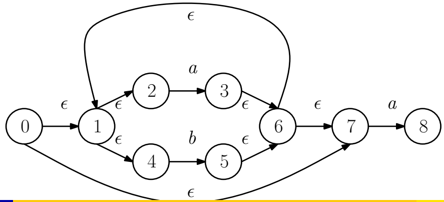


Constructing NFA from Regular expression

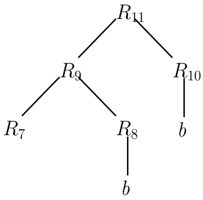


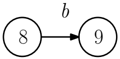
For $R_6 = a, N_6 =$ 

For $R_7 = R_5 \cdot R_6, N_6 =$

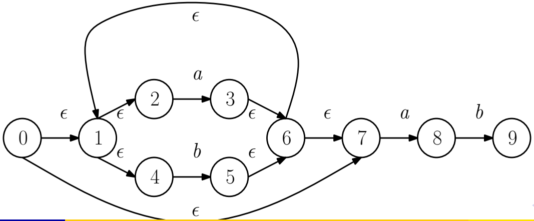


Constructing NFA from Regular expression

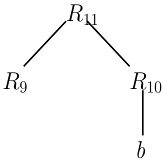


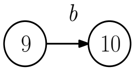
For $R_8 = a, N_8 =$ 

For $R_9 = R_7 \cdot R_8, N_9 =$

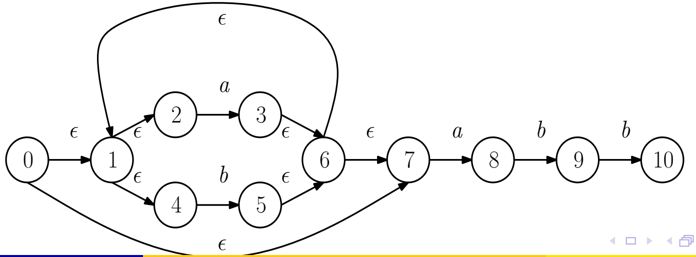


Constructing NFA from Regular expression

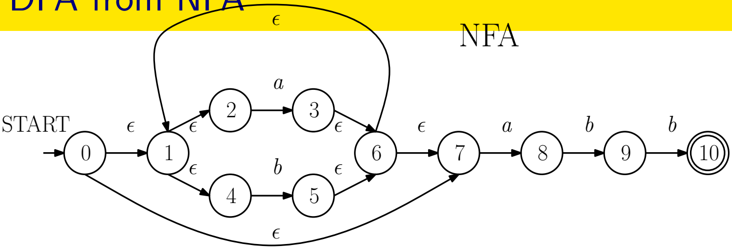


For $R_{10} = b, N_{10} =$ 

For $R_{11} = R_9 \cdot R_{10}, N_{11} =$



Constructing DFA from NFA



$$A = \epsilon - CLOSURE(\{0\}) = \{0, 1, 2, 4, 7\}$$

$$B = \delta(A, a) = \epsilon - CLOSURE(\{3\}) \cup \epsilon - CLOSURE(\{8\}) = \{1, 2, 3, 4, 6, 7, 8\}$$
$$C = \delta(A, b) = \epsilon - CLOSURE(\{5\}) = \{1, 2, 4, 5, 6, 7\}$$

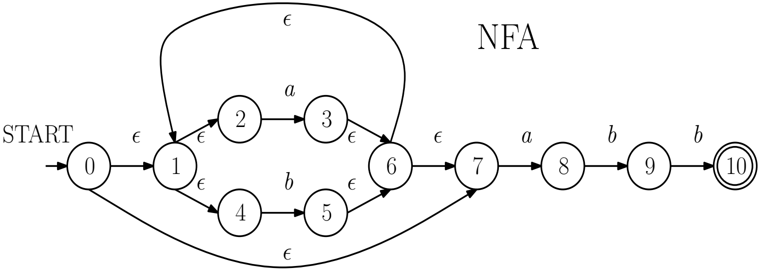
$$\delta(B, a) = \epsilon - CLOSURE(\{3\}) \cup \epsilon - CLOSURE(\{8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$$
$$D = \delta(B, b) = \epsilon - CLOSURE(\{5\}) \cup \epsilon - CLOSURE(\{9\}) = \{1, 2, 4, 5, 6, 7, 9\}$$

$$\delta(C, a) = \epsilon - CLOSURE(\{3\}) \cup \epsilon - CLOSURE(\{8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$$
$$\delta(C, b) = \epsilon - CLOSURE(\{5\}) = \{1, 2, 4, 5, 6, 7\} = C$$

$$\delta(D, a) = \epsilon - CLOSURE(\{3\}) \cup \epsilon - CLOSURE(\{8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$$
$$E = \delta(D, b) = \epsilon - CLOSURE(\{5\}) \cup \epsilon - CLOSURE(\{9\}) = \{1, 2, 4, 5, 6, 7, 10\}$$

$$\delta(E, a) = \epsilon - CLOSURE(\{3\}) \cup \epsilon - CLOSURE(\{8\}) = B$$
$$\delta(E, b) = \epsilon - CLOSURE(\{5\}) = C$$

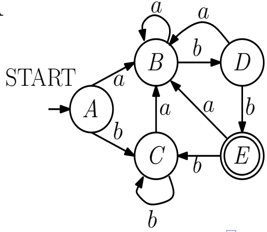
Constructing DFA from NFA



DFA

	NFA State	DFA State	Input a	Input b
Start	$\{0, 1, 2, 4, 7\}$	A	B	C
	$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
	$\{1, 2, 4, 5, 6, 7\}$	C	B	C
	$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
Accept	$\{1, 2, 4, 5, 6, 7, 10\}$	E	B	C

Transition table



Transition diagram

Minimizing number of states in DFA

DFA

	State	Input a	Input b
Start	A	B	C
	B	B	D
	C	B	C
	D	B	E
Accept	E	B	C

DFA with minimum number of states

	State	Input a	Input b
Start	A	B	A
	B	B	D
	D	B	E
Accept	E	B	A

$A = \{A, C\}$

Π_i : partition of set of states at step i

$\Pi_0 = \{\{A, B, C, D\}, \{E\}\}$, final and non-final states

$\delta(A, a) = \delta(B, a) = \delta(C, a) = \delta(D, a) = B \in \{A, B, C, D\}$

$\delta(A, b), \delta(B, b), \delta(C, b) \in \{A, B, C, D\}$ and $\delta(D, b) \in \{E\}$

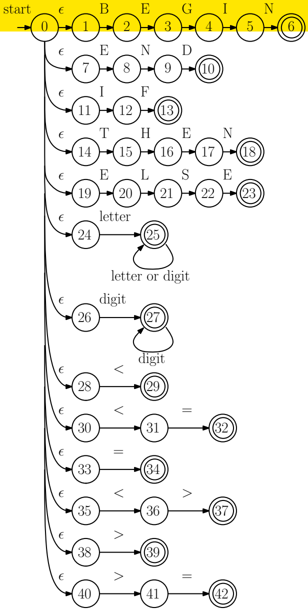
$\Pi_1 = \{\{A, B, C\}, \{D\}, \{E\}\}$

$\delta(A, a) = \delta(C, a) = B$ and $\delta(A, b) = \delta(C, b) = C$

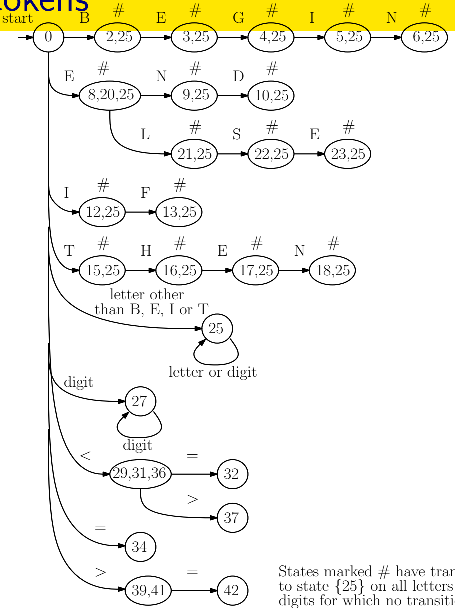
$\delta(B, a) = B$ and $\delta(B, b) = D$

$\Pi_2 = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$, final partition

Combined NFA for tokens



Combined DFA for tokens



States marked # have transitions to state {25} on all letters and digits for which no transition is shown

Thank you