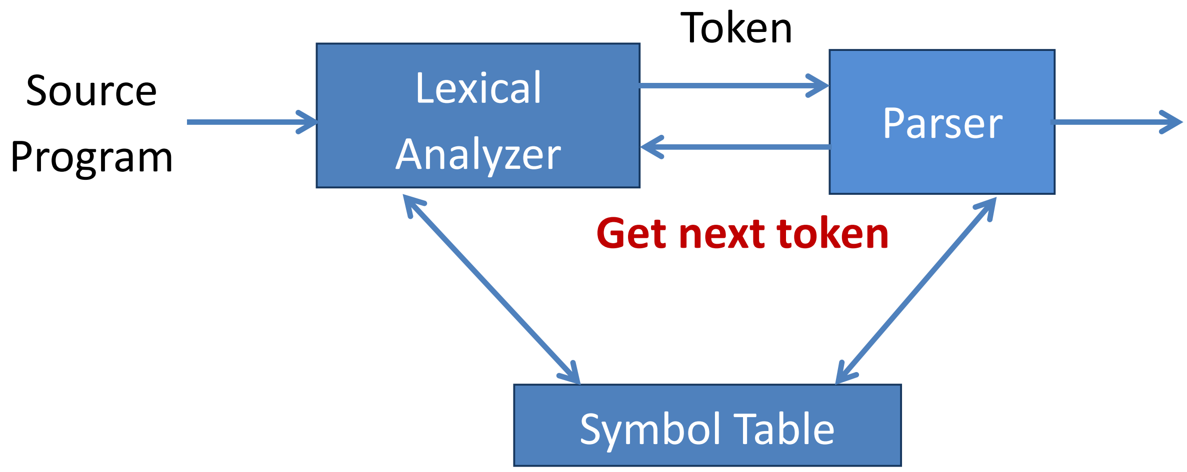
Unit – 1 (Part-2)
Lexical Analysis

Mrs. Ruchi Sharma
ruchi.sharma@bkbiet.ac.in

Topics to be covered

- Interaction of scanner & parser
- Token, Pattern & Lexemes
- Input buffering
- Specification of tokens
- Regular expression & Regular definition
- Transition diagram
- Hard coding & automatic generation lexical analyzers
- Finite automata
- Regular expression to NFA using Thompson's rule
- Conversion from NFA to DFA using subset construction method
- DFA optimization
- Conversion from regular expression to DFA

Interaction of scanner & parser



- Upon receiving a **"Get next token"** command from parser, the lexical analyzer reads the input character until it can identify the next token.
- Lexical analyzer also stripping out comments and white space in the form of blanks, tabs, and newline characters from the source program.

Why to separate lexical analysis & parsing?

1. Simplicity in design.
2. Improves compiler efficiency.
3. Enhance compiler portability.

Token, Pattern & Lexemes

Token

Sequence of character having a collective meaning is known as **token**.

Categories of Tokens:

1. Identifier
2. Keyword
3. Operator
4. Special symbol
5. Constant

Pattern

The set of rules called **pattern** associated with a token.

Example: "*non-empty sequence of digits*", "*letter followed by letters and digits*"

Lexemes

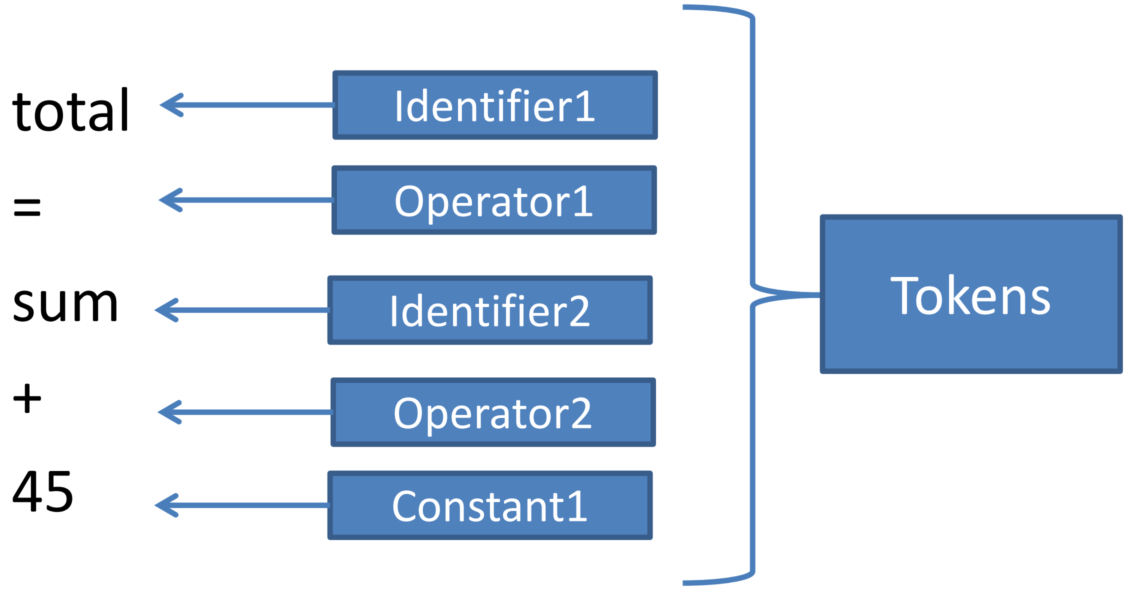
The sequence of character in a source program matched with a pattern for a token is called lexeme.

Example: Rate, DIET, count, Flag

Token, Pattern & Lexemes (Example)

Example: total = sum + 45

Tokens



Lexemes

Lexemes of identifier: total, sum

Lexemes of operator: =, +

Lexemes of constant: 45

Input buffering

Input buffering

There are mainly two techniques for input buffering:

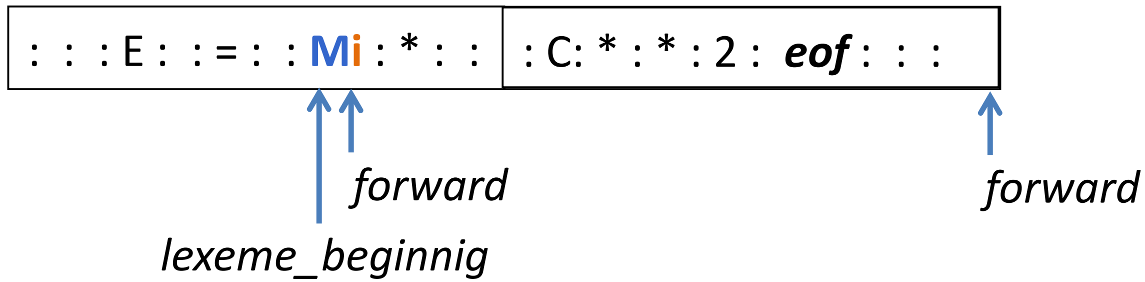
1. Buffer pairs
2. Sentinels

Buffer pairs

- The lexical analysis scans the input string from left to right one character at a time.
- Buffer divided into two N-character halves, where N is the number of character on one disk block.

: : : E : : = : : Mi : * : :	: C: * : * : 2 : ***eof*** : : :

Buffer pairs



- Pointer *Lexeme Begin*, marks the beginning of the current lexeme.
- Pointer *Forward*, scans ahead until a pattern match is found.
- Once the next lexeme is determined, *forward* is set to character at its right end.
- Lexeme Begin is set to the character immediately after the lexeme just found.
- If forward pointer is at the end of first buffer half then second is filled with N input character.
- If forward pointer is at the end of second buffer half then first is filled with N input character.

Buffer pairs

: : : E : : = : : Mi : * : :	: C: * : * : 2 : ***eof*** : : :

forward *lexeme_beginnig* *forward* *forward*

Code to advance forward pointer

if *forward at end of first half* ***then begin***

reload second half;

forward := forward + 1;

end

else if *forward at end of second half* ***then begin***

reload first half;

move forward to beginning of first half;

end

else *forward := forward + 1;*

Sentinels

: : E : : = : : Mi : * : ***eof***	: C: * : * : 2 : ***eof*** : : ***eof***

forward

lexeme_beginnig

- In buffer pairs we must check, each time we move the forward pointer that we have not moved off one of the buffers.
- Thus, for each character read, we make two tests.
- We can combine the buffer-end test with the test for the current character.
- We can reduce the two tests to one if we extend each buffer to hold a sentinel character at the end.
- The sentinel is a special character that cannot be part of the source program, and a natural choice is the character **EOF.**

Sentinels

: : E : : = : : Mi : * : *eof*	: C: * : * : 2 : ***eof*** : : ***eof***

forwa *forward* *forward*

lexeme_beginnig

```
forward := forward + 1;
if forward = eof then begin
        if forward at end of first half then begin
                reload second half;
                forward := forward + 1;
        end
        else if forward at the second half then begin
                reload first half;
                move forward to beginning of first half;
        end
        else terminate lexical analysis;
end
```

Specification of tokens

Strings and languages

Term	Definition
Prefix of s	A string obtained by removing **zero or more trailing symbol** of string S. e.g., ban is prefix of banana.
Suffix of S	A string obtained by removing **zero or more leading symbol** of string S. e.g., nana is suffix of banana.
Sub string of S	A string obtained by **removing prefix and suffix** from S. e.g., nan is substring of banana
Proper prefix, suffix and substring of S	Any nonempty string x that is respectively proper prefix, suffix or substring of S, such that **s≠x**.
Subsequence of S	A string obtained by removing **zero or more not necessarily contiguous symbol** from S. e.g., baaa is subsequence of banana.

Exercise

- Write prefix, suffix, substring, proper prefix, proper suffix and subsequence of following string:

  String: Compiler

Operations on languages

Operation	Definition
Union of L and M Written L U M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M *Written LM*	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L Written L^*	L^* *denotes "zero or more concatenation of" L.*
Positive closure of L *Written* L^+	L^+ *denotes "one or more concatenation of" L.*

Regular expression & Regular definition

Regular expression

- A regular expression is a sequence of characters that define a pattern.

Notational shorthand's

1. One or more instances: +
2. Zero or more instances: *
3. Zero or one instances: ?
4. Alphabets: Σ

Rules to define regular expression

1. ∈ is a regular expression that denotes {∈}, the set containing empty string.
2. If a is a symbol in Σ then a is a regular expression, $L(a) = \{a\}$
3. Suppose r and s are regular expression denoting the languages $L(r)$ and $L(s)$. Then,
   a. $(r)|(s)$ is a regular expression denoting $L(r)\ U\ L(s)$
   b. $(r)(s)$ is a regular expression denoting $L(r)L(s)$
   c. (r)* is a regular expression denoting $\big(L(r)\big)^*$
   d. (r) is a regular expression denoting $L((r))$

The language denoted by regular expression is said to be a regular set.

Regular expression

L = Zero or More Occurrences of a = a^*

*****

$\left.\begin{array}{l} \epsilon \\ a \\ aa \\ aaa \\ aaaa \\ aaaaa..... \end{array}\right\}$ **Infinite …..**

Regular expression

L = One or More Occurrences of a = a^+

+

a

aa

aaa

aaaa

aaaaa.....

Infinite .....

Precedence and associativity of operators

Operator	Precedence	Associative
Kleene *	1	left
Concatenation	2	left
Union |	3	left

Regular expression examples

1. 0 or 1

   $\boldsymbol{Strings}\text{: } \mathbf{0, 1}$ $\quad$ $\boldsymbol{R.E.} = \mathbf{0 \mid 1}$

2. 0 or 11 or 111

   $\boldsymbol{Strings}\text{: } \mathbf{0, 11, 111}$ $\quad$ $\boldsymbol{R.E.} = \mathbf{0 \mid 11 \mid 111}$

3. String having zero or more *a*.

   $\boldsymbol{Strings}\text{: } \epsilon, \boldsymbol{a, aa, aaa, aaaa} \ldots..$ $\quad$ $\boldsymbol{R.E.} = \boldsymbol{a}^*$

4. String having one or more *a*.

   $\boldsymbol{Strings}\text{: } \boldsymbol{a, aa, aaa, aaaa} \ldots..$ $\quad$ $\boldsymbol{R.E.} = \boldsymbol{a}^+$

5. Regular expression over $\Sigma = \{a, b, c\}$ that represent all string of length 3.

   $\boldsymbol{Strings}\text{: } \boldsymbol{abc, bca, bbb, cab, aba} \ldots.$ $\quad$ $\boldsymbol{R.E.} = (\boldsymbol{a}|\boldsymbol{b}|\boldsymbol{c})\ (\boldsymbol{a}|\boldsymbol{b}|\boldsymbol{c})\ (\boldsymbol{a}|\boldsymbol{b}|\boldsymbol{c})$

6. All binary string.

   $\boldsymbol{Strings}\text{: } \mathbf{0, 11, 101, 10101, 1111} \ldots$ $\quad$ $\boldsymbol{R.E.} = (\mathbf{0 \mid 1}) +$

Regular expression examples

7. 0 or more occurrence of either a or b or both

   $\boldsymbol{Strings}: \epsilon, \boldsymbol{a}, \boldsymbol{aa}, \boldsymbol{abab}, \boldsymbol{bab} \ldots$ $\quad$ $\boldsymbol{R.E.} = (\boldsymbol{a} \mid \boldsymbol{b}) *$

8. 1 or more occurrence of either a or b or both

   $\boldsymbol{Strings}: \boldsymbol{a}, \boldsymbol{aa}, \boldsymbol{abab}, \boldsymbol{bab}, \boldsymbol{bbbaaa} \ldots$ $\quad$ $\boldsymbol{R.E.} = (\boldsymbol{a} \mid \boldsymbol{b}) +$

9. Binary no. ends with 0

   $\boldsymbol{Strings}: \mathbf{0}, \mathbf{10}, \mathbf{100}, \mathbf{1010}, \mathbf{11110} \ldots$ $\quad$ $\boldsymbol{R.E.} = (\mathbf{0} \mid \mathbf{1}) * \mathbf{0}$

10. Binary no. ends with 1

   $\boldsymbol{Strings}: \mathbf{1}, \mathbf{101}, \mathbf{1001}, \mathbf{10101}, \ldots$ $\quad$ $\boldsymbol{R.E.} = (\mathbf{0} \mid \mathbf{1}) * \mathbf{1}$

11. Binary no. starts and ends with 1

   $\boldsymbol{Strings}: \mathbf{11}, \mathbf{101}, \mathbf{1001}, \mathbf{10101}, \ldots$ $\quad$ $\boldsymbol{R.E.} = \mathbf{1}\,(\mathbf{0} \mid \mathbf{1}) * \mathbf{1}$

12. String starts and ends with same character

   $\boldsymbol{Strings}: \mathbf{00}, \mathbf{101}, \boldsymbol{aba}, \boldsymbol{baab} \ldots$

   $\boldsymbol{R.E.} = \mathbf{1}\,(\mathbf{0} \mid \mathbf{1}) * \mathbf{1}$ **or** $\mathbf{0}\,(\mathbf{0} \mid \mathbf{1}) * \mathbf{0}$

   $\boldsymbol{a}\,(\boldsymbol{a} \mid \boldsymbol{b})^{*}\,\boldsymbol{a}$ **or** $\boldsymbol{b}\,(\boldsymbol{a} \mid \boldsymbol{b})^{*}\,\boldsymbol{b}$

Regular expression examples

13. All string of a and b starting with a

    $\boldsymbol{Strings: a, ab, aab, abb \ldots}$ $\boldsymbol{R.E. = a(a \mid b)*}$

14. String of 0 and 1 ends with 00

    $\boldsymbol{Strings: 00, 100, 000, 1000, 1100 \ldots}$ $\boldsymbol{R.E. = (0 \mid 1) * 00}$

15. String ends with abb

    $\boldsymbol{Strings: abb, babb, ababb \ldots}$ $\boldsymbol{R.E. = (a \mid b) * abb}$

16. String starts with 1 and ends with 0

    $\boldsymbol{Strings: 10, 100, 110, 1000, 1100 \ldots}$ $\boldsymbol{R.E. = 1(0 \mid 1) * 0}$

17. All binary string with at least 3 characters and 3rd character should be zero

    $\boldsymbol{Strings: 000, 100, 1100, 1001 \ldots}$ $\boldsymbol{R.E. = (0|1)(0|1)0(0 \mid 1)*}$

18. Language which consist of exactly two b's over the set $\Sigma = \{a, b\}$

    $\boldsymbol{Strings: bb, bab, aabb, abba \ldots}$ $\boldsymbol{R.E. = a^* b a^* b a^*}$

Regular expression examples

19. The language with $\Sigma = \{a, b\}$ such that 3rd character from right end of the string is always

$\boldsymbol{Strings: aaa, aba, aaba, abb...}$ $\boldsymbol{R.E. = (a \mid b) * a(a|b)(a|b)}$

19. Any no. of a followed by any no. of b followed by any no. of c

$\boldsymbol{Strings: \epsilon, abc, aabbcc, aabc, abb...}$ $\boldsymbol{R.E. = a^{*} b^{*} c^{*}}$

20. String should contain at least three 1

$\boldsymbol{Strings: 111, 01101, 0101110....}$ $\boldsymbol{R.E. = (0|1)^{*}1\ (0|1)^{*}1\ (0|1)^{*}1\ (0|1)^{*}}$

21. String should contain exactly two 1

$\boldsymbol{Strings: 11, 0101, 1100, 010010, 100100....}$ $\boldsymbol{R.E. = 0^{*}10^{*}10^{*}}$

22. Length of string should be at least 1 and at most 3

$\boldsymbol{Strings: 0, 1, 11, 01, 111, 010, 100....}$ $\boldsymbol{R.E. = (0|1) \mid (0|1)(0|1) \mid (0|1)(0|1)(0|1)}$

23. No. of zero should be multiple of 3

$\boldsymbol{Strings: 000, 010101, 110100, 000000, 100010010....}$ $\boldsymbol{R.E. = (1^{*}01^{*}01^{*}01^{*})^{*}}$

Regular expression examples

24. The language with $\Sigma = \{a, b, c\}$ where a should be multiple of 3

    $\boldsymbol{Strings: aaa, baaa, bacaba, aaaaaa..}$ $\boldsymbol{R.E. = ((b|c)^* a(b|c)^* a(b|c)^* a(b|c)^*)^*}$

25. Even no. of 0

    $\boldsymbol{Strings: 00, 0101, 0000, 100100....}$ $\boldsymbol{R.E. = (1^* 01^* 01^*)^*}$

26. String should have odd length

    $\boldsymbol{Strings: 0, 010, 110, 000, 10010....}$ $\boldsymbol{R.E. = (0|1)\ ((0|1)(0|1))^*}$

27. String should have even length

    $\boldsymbol{Strings: 00, 0101, 0000, 100100....}$ $\boldsymbol{R.E. = ((0|1)(0|1))^*}$

28. String start with 0 and has odd length

    $\boldsymbol{Strings: 0, 010, 010, 000, 00010....}$ $\boldsymbol{R.E. = (0)\ ((0|1)(0|1))^*}$

30. String start with 1 and has even length

    $\boldsymbol{Strings: 10, 1100, 1000, 100100....}$ $\boldsymbol{R.E. = 1(0|1)((0|1)(0|1))^*}$

Regular expression examples

31. All string begins or ends with 00 or 11

$Strings$: $\mathbf{00101, 10100, 110, 01011} \dots$ $R.E. = \mathbf{(00|11)(0\ |\ 1)*\ |\ (0|1)^{*}\ (00|11)}$

32. Language of all string containing both 11 and 00 as substring

$Strings$: $\mathbf{0011, 1100, 100110, 010011} \dots$

$R.E. = \mathbf{((0|1)^{*}00(0|1)^{*}11(0|1)^{*})\ \ |\ \ ((0|1)^{*}11(0|1)^{*}00(0|1)^{*})}$

33. String ending with 1 and not contain 00

$Strings$: $\mathbf{011, 1101, 1011} \dots.$ $R.E. = \mathbf{(1|01)^{+}}$

34. Language of C identifier

$Strings: area, i, redious, grade1 \dots.$ $R.E. = (\_ + L)(\_\ +\ L\ + D)^{*}$

$where\ L\ is\ Letter$ & **D is digit**

Regular definition

- A regular definition gives names to certain regular expressions and uses those names in other regular expressions.
- Regular definition is a sequence of definitions of the form:

  $d_1 \rightarrow r_1$
  $d_2 \rightarrow r_2$
  ……
  $d_n \rightarrow rn$

  Where d_i is a **distinct name** & r_i is a **regular expression**.
- Example: Regular definition for identifier

  letter → A|B|C|……….|Z|a|b|……….|z

  digit → 0|1|…….|9|

  id→ letter (letter | digit)*

Regular definition example

- Example: Unsigned Pascal numbers

  3

  5280

  39.37

  6.336E4

  1.894E-4

  2.56E+7

Regular Definition

digit → 0|1|…..|9

digits → digit digit*

optional_fraction → .digits | ϵ

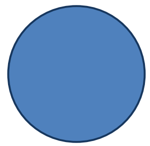
optional_exponent → (E(+|-|ϵ)digits)|ϵ

num → digits optional_fraction optional_exponent

Transition diagram

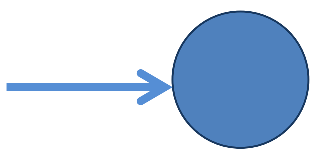
Transition diagram

- A stylized flowchart is called transition diagram.

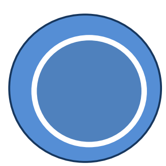


is a state

is a transition

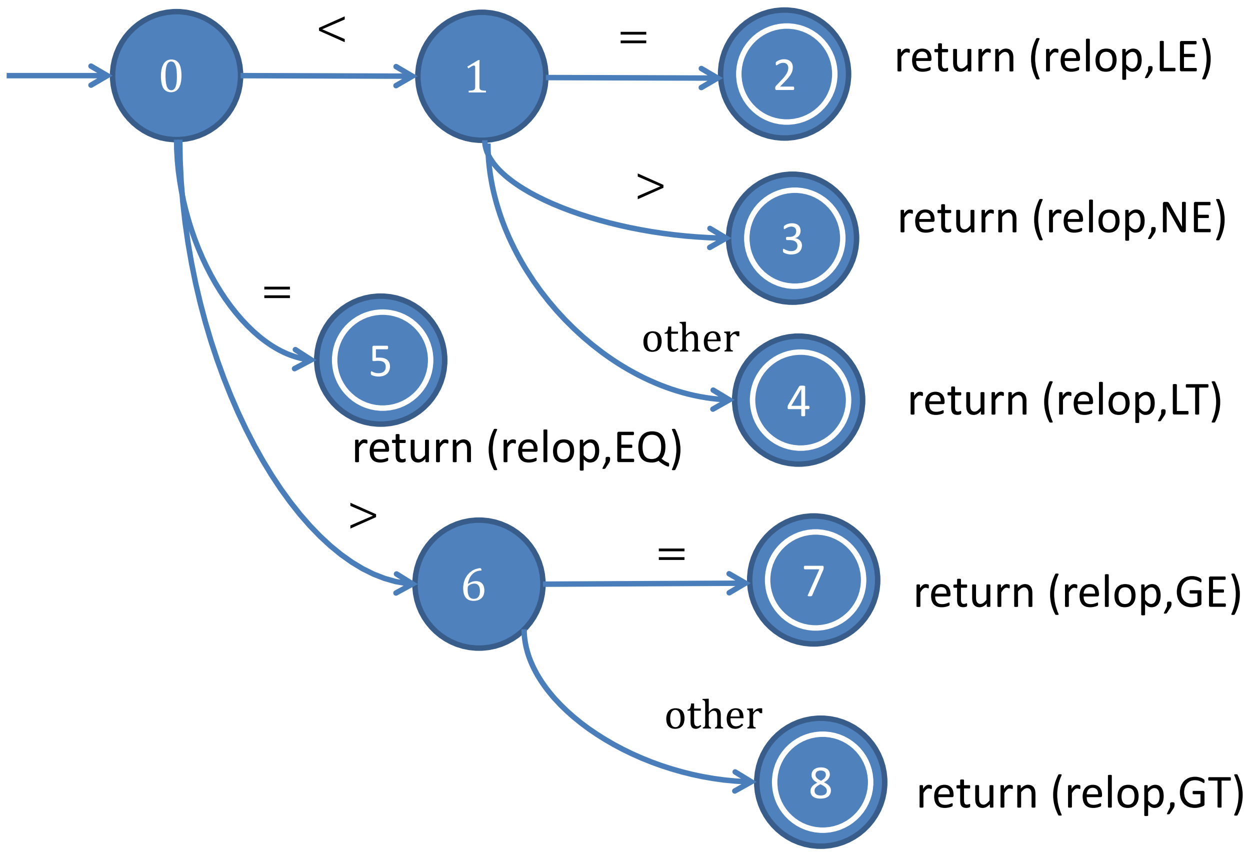


is a start state



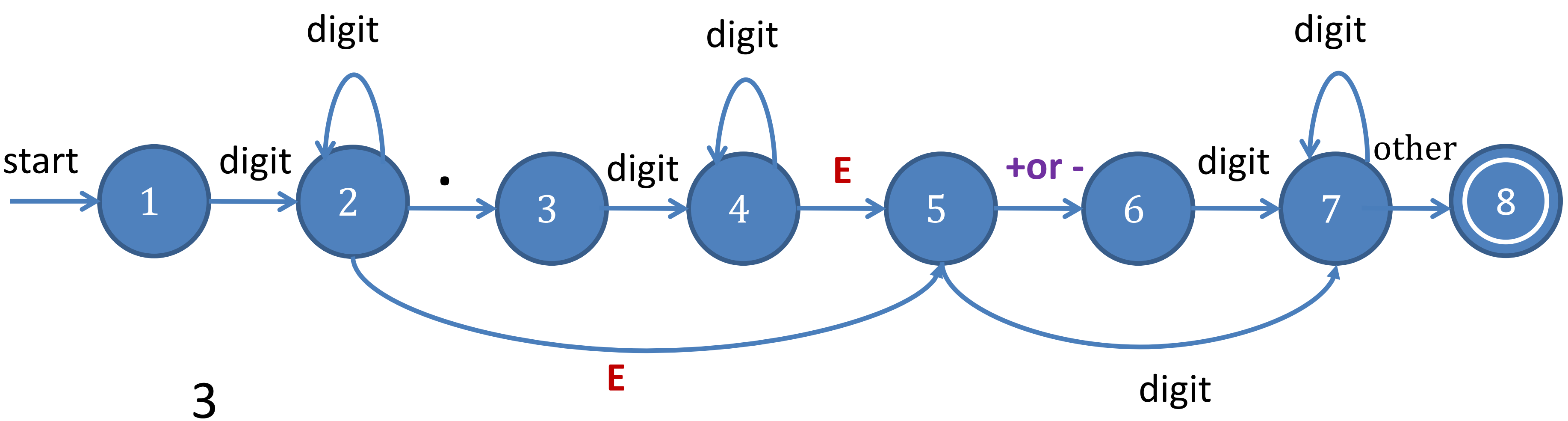
is a final state

Transition diagram example: Relational opera



Transition diagram example: Unsigned number

Transition diagram for unsigned number in pascal

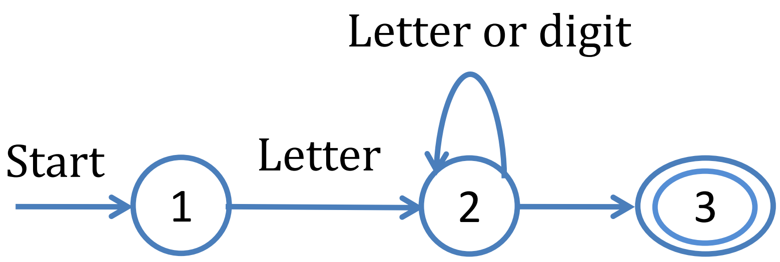


3
5280
39.37
1.894 **E - 4**
2.56 **E + 7**
45 **E + 6**
96 **E** 2

Hard coding & automatic generation lexical analyzers

Hard coding and automatic generation lexical analyzers

- Lexical analysis is about identifying the pattern from the input.
- To recognize the pattern, transition diagram is constructed.
- It is known as hard coding lexical analyzer.
- Example: to represent identifier in ‘C’, the first character must be letter and other characters are either letter or digits.
- To recognize this pattern, hard coding lexical analyzer will work with a transition diagram.



Hard coding and automatic generation lexical analyzers

- The automatic generation lexical analyzer takes special notation as input.
- For example, lex compiler tool will take regular expression as input and finds out the pattern matching to that regular expression.

End of Part-2(UNIT-1)