# Lexical Analysis

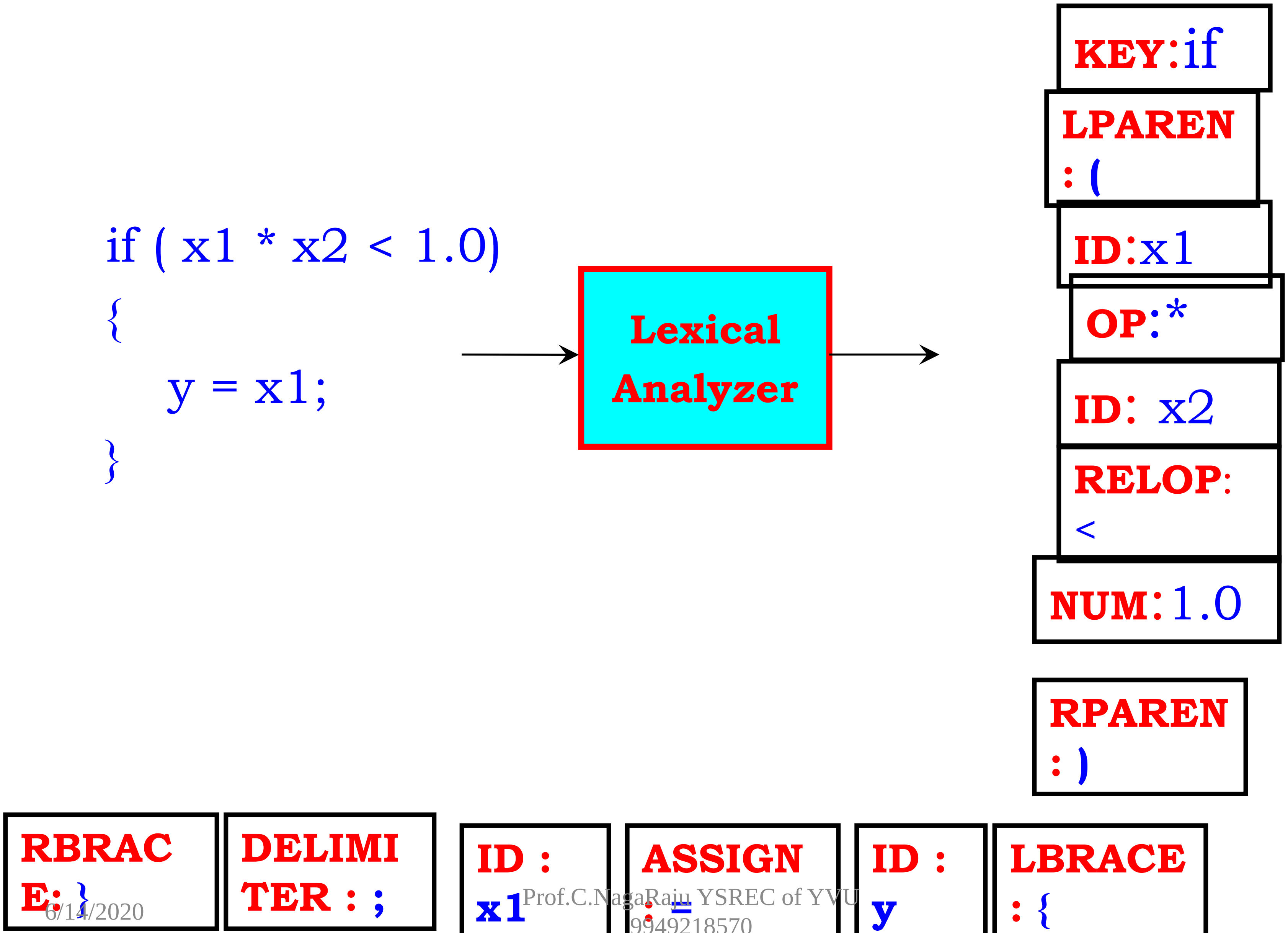
C.Naga Raju
B.Tech(CSE),M.Tech(CSE),PhD(CSE),MIEEE,MCSI,MISTE
Professor
Department of CSE
YSR Engineering College of YVU
Proddatur

# Contents

- Introduction to lexical Analysis
- Specification of tokens
- Recognition of tokens  using transition diagrams
- Regular expressions
- Regular languages
- Examples
- GATE solved problems on lexical analysis

- LEXICAL ANALYSIS OR SCANNER OR LINEAR ANALYSIS
- Lexical analysis is the first phase of a compiler
- It Reads one character at a time from the source program from left to right and generate lexemes
- A lexeme is the process of forming the words based on pattern rules and convert them into tokens
- These tokens are divided into keywords, identifiers, operators, delimiters and punctuation symbols
- each token is represented with pair of values <identifier, number>
- It Recognize the various Tokens with the help of regular expressions and pattern rules. and It classifies the various Tokens
-

# Representation of Lexical Analysis



```
//Consider the program
  int main()  {
  // 2 variables
 int a, b;
 a = 10;
return 0;
       }
```

**'int' 'main' '(' ')' '{' 'int' 'a' ',' 'b' ';' 'a' '=' '10' ';' 'return' '0' ';' '}'**

- It Remove comments and white spaces
- It Interacts with the symbol table
- sends lexical errors to error handling table

Pattern :A pattern is a description form of the lexemes

identifier L(L|d)* (l| _) (L|_|d)*

Lexeme :A lexeme is the process of forming the words using patterns

example: a,b,c,sum ,< ,<=,>,>=,==,!=, &&,|| !, 20,45 ,if,for,break etc.

Token : similar lexemes are grouped into single logical units called as Token.

For example **relop** is token for all relational operators

examples:

1) a,b,c,sum are represent with common name identifier token
2) <,<=,>,>=,==,!= are represent with common name relop token
3) 20,30 are const token
4) If , for, break are keyword tokens

- Tokens are recognized by regular grammar and Tokens are implemented by finite automata
- Recognition of tokens
- In any programming language reorganization of tokens is the first and most important step:
- Ex:

stmt -> **if** expr **then** stmt
| **if** expr **then** stmt **else** stmt
| ε
expr -> term **relop** term
| term
term -> **id**
| **number**

# Overall

| Regular Expression | Token | Attribute-Value |
|---|---|---|
| ws | - | - |
| **if** | **if** | - |
| **then** | **then** | - |
| **else** | **else** | - |
| **id** | **id** | **pointer to table entry** |
| **num** | **num** | **pointer to table entry** |
| < | **relop** | LT |
| <= | **relop** | LE |
| = | **relop** | EQ |
| < > | **relop** | NE |
| > | **relop** | GT |
| >= | **relop** | GE |

# Lexical Errors

- Some errors are out of power of lexical analyzer to recognize:
  - fi (a == f(x)) …
- However it may be able to recognize errors like:
  - d = 2r
- Such errors are recognized when no pattern for tokens matches a character sequence


Prof.C.NagaRaju YSREC of YVU
9949218570

# Error Recovery

- Panic mode: successive characters are ignored until we reach to a well formed token
- Delete one character from the remaining input
- Insert a missing character into the remaining input
- Replace a character by another character
- Transpose two adjacent characters

# Specification Of Tokens

- In compiler design regular expressions are used to formalize the specification of tokens
- Regular expressions are used for specifying regular languages
- Example:
  - (Letter|_)(letter|_ | digit)*
- Each regular expression is a pattern specifying the form of strings
- One or more instances: (r)+
- Zero or more instances: r*
- Character classes: [abc]

- Components for Construction of the patterns

  digit -> [0-9]

  Capital letter -> [A-Z ]

  Small letters[ a-z_]

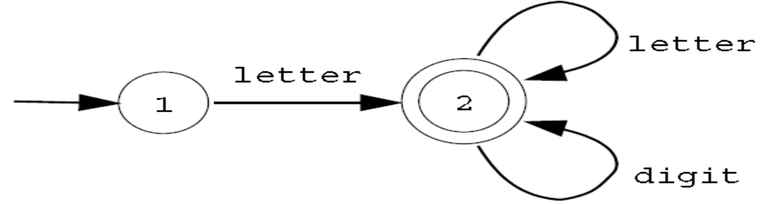
  Key words patterns

- whitespaces: ws -> (blank | tab | newline)+
- These patterns may be represented with

  1)transistion diagrams

  2) Regular expressions

- **Transition Diagram**




✓ Pictorial representation of labeled directed graph called a Transition Diagram

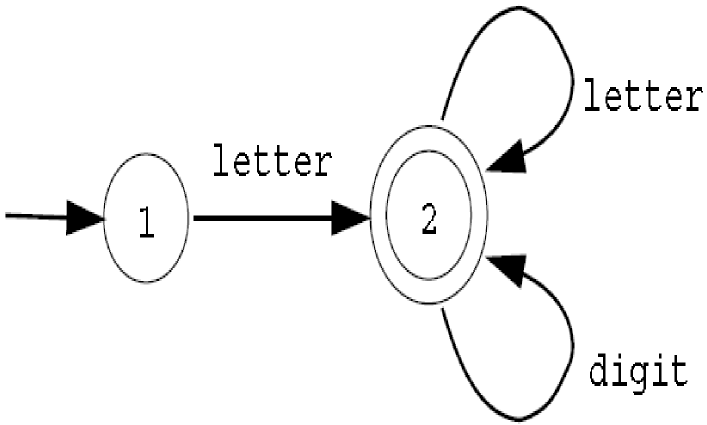
✓ Circles represent states. They represent how much of the input string we have processed.

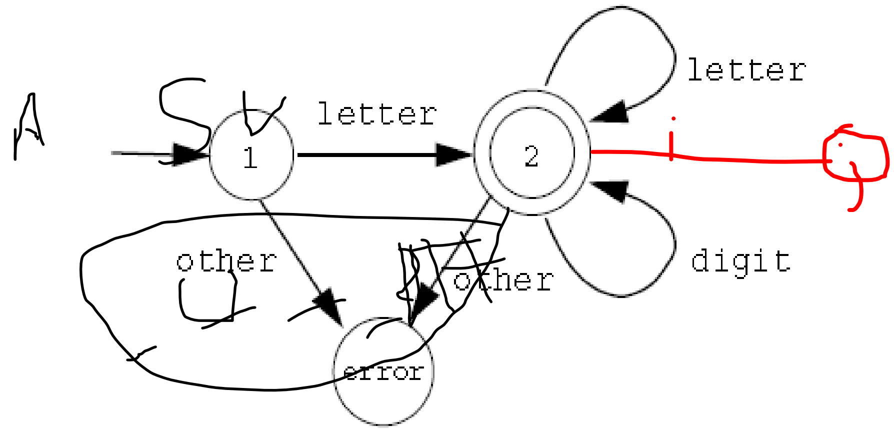
✓ Arrows represent transitions from one state to the next state when the character labeling the arrow is matched.

✓ State 1 is the starting state.

✓ Final or Accepting states are represented by double circles.

# Transition Diagram : Identifier & Identifier with erroneous




❖ It shows that the string of characters "**tmp8**" form a legal identifier

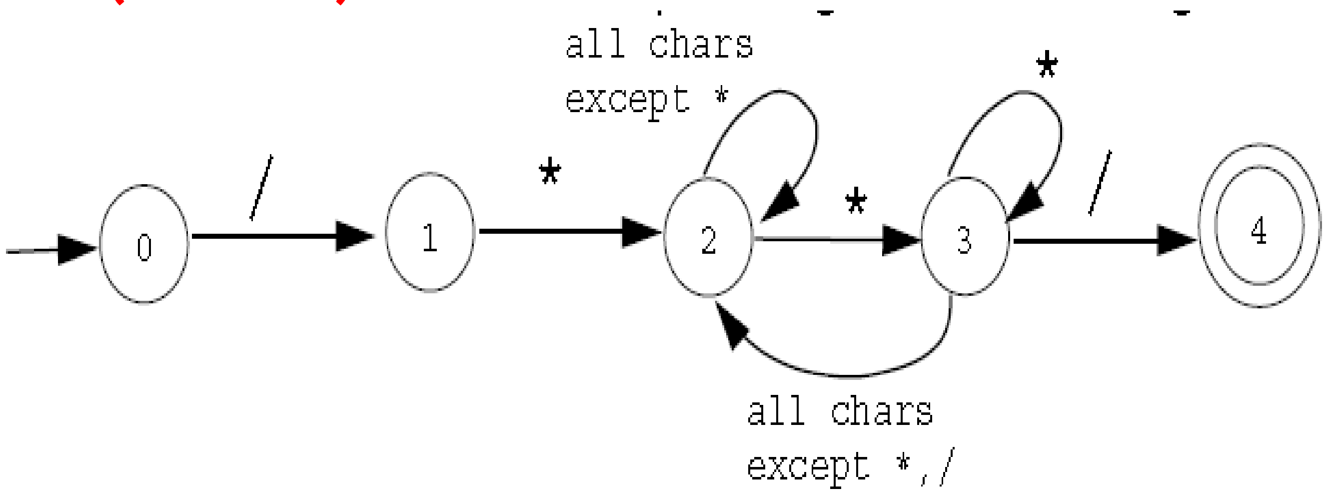
SUM

$$\rightarrow 1 \xrightarrow{t} 2 \xrightarrow{m} 2 \xrightarrow{p} 2 \xrightarrow{8} 2$$

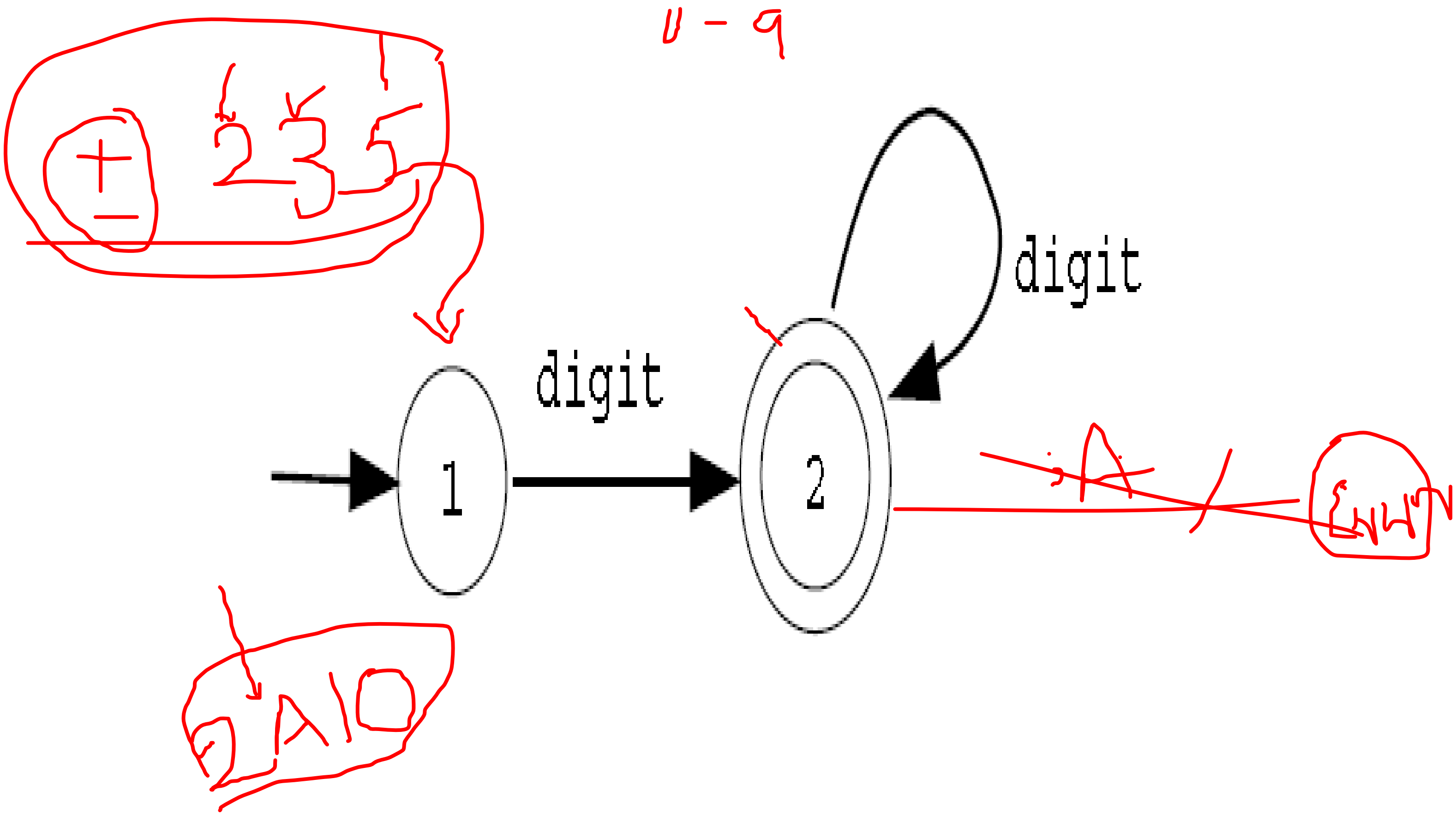
# Transition Diagram : C Comments

❖ C comments are of the form

**/* … (no */s) … */**



❖ Transition diagram for **Natural Numbers**

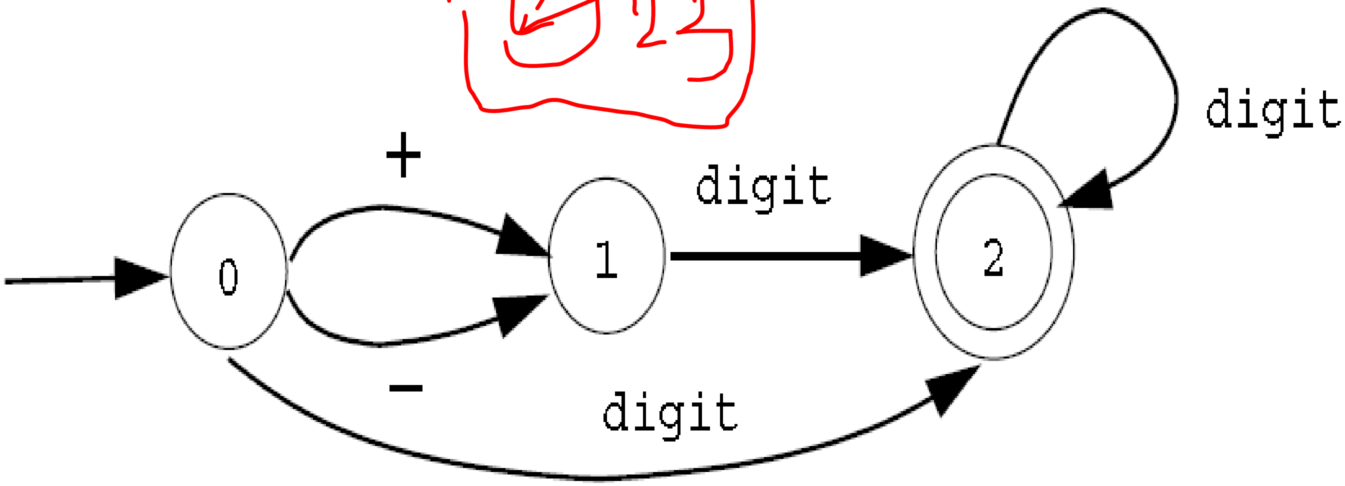


❖ Transition diagram for **Signed Natural Numbers**

-235

+456

623



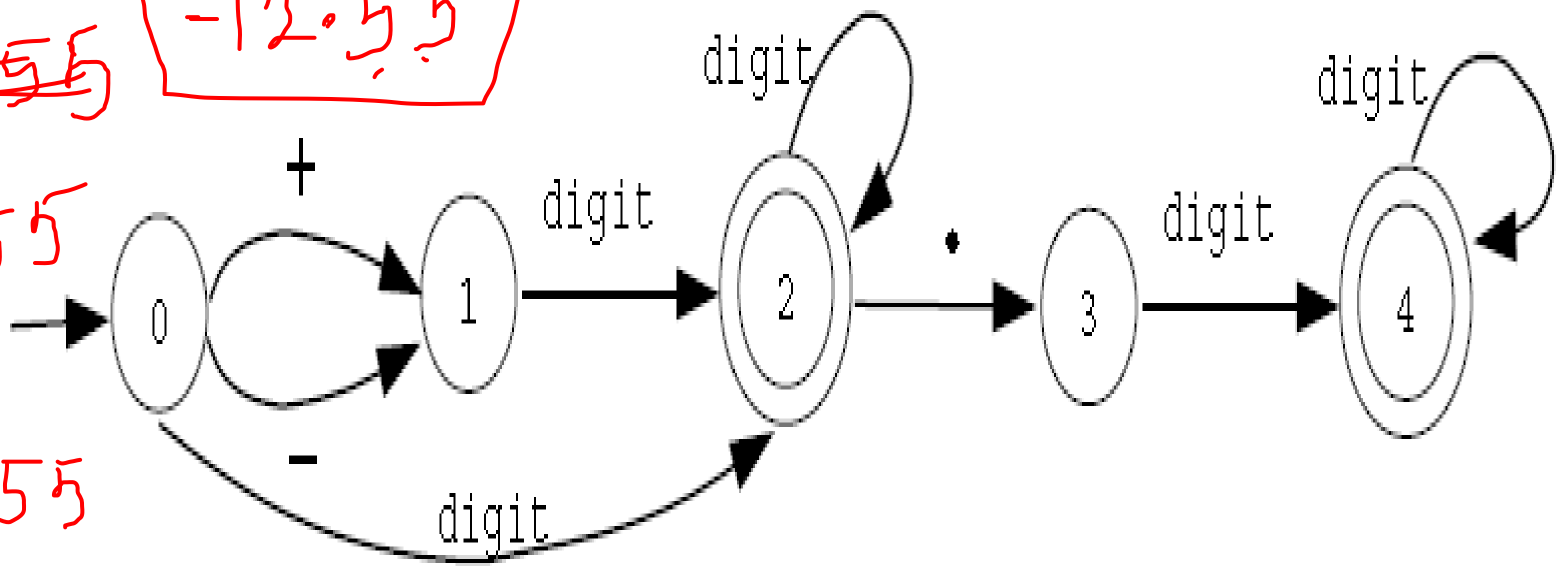

(+|-)d(d)*

## ❖ Transition diagram for **Signed Real Numbers**

~~+12.55~~

$\boxed{-12.55}$

+12.55

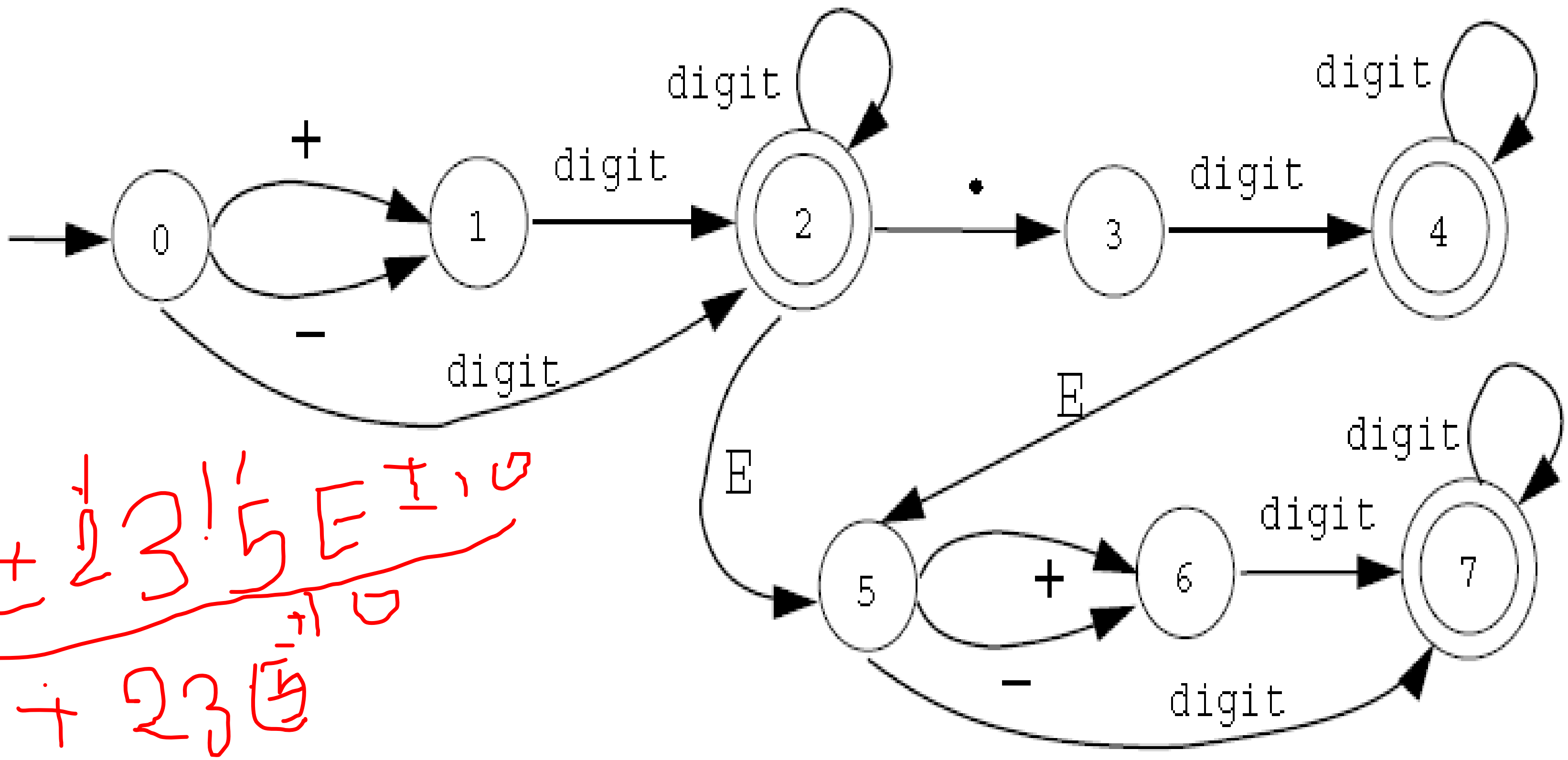



12.55

126

$\boxed{23-4.67}$

❖ Transition diagram for signed **Floating Point numbers**

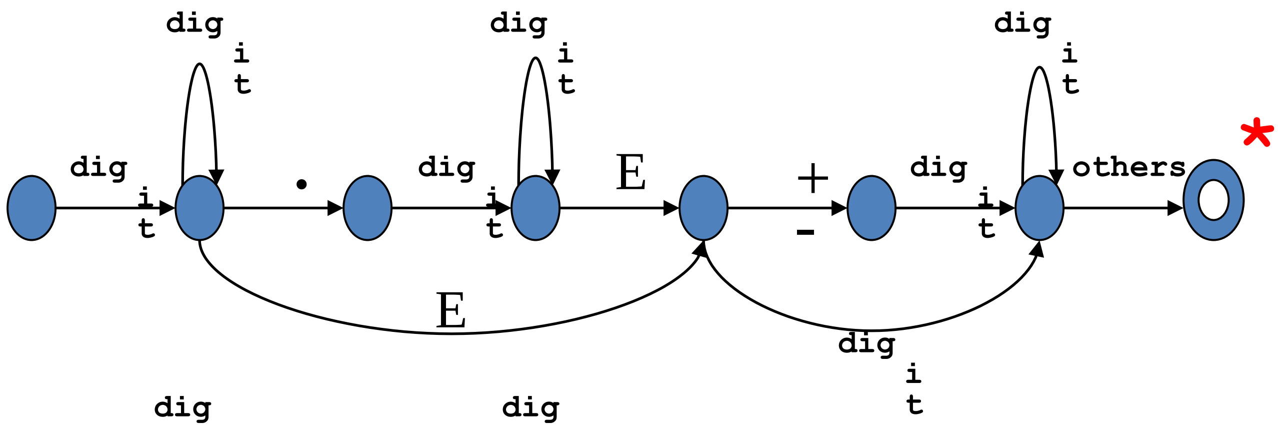


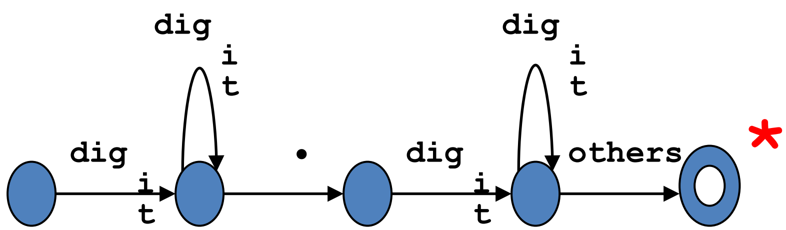

$\pm 23.5E^{\pm 10}$

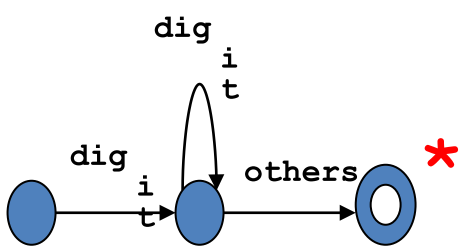
$\pm 23E^{\pm 10}$

$\pm E^{10}$

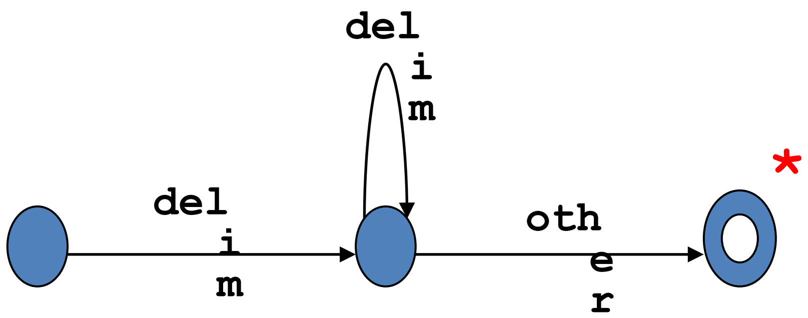
# Transition Diagram : Unsigned floating number



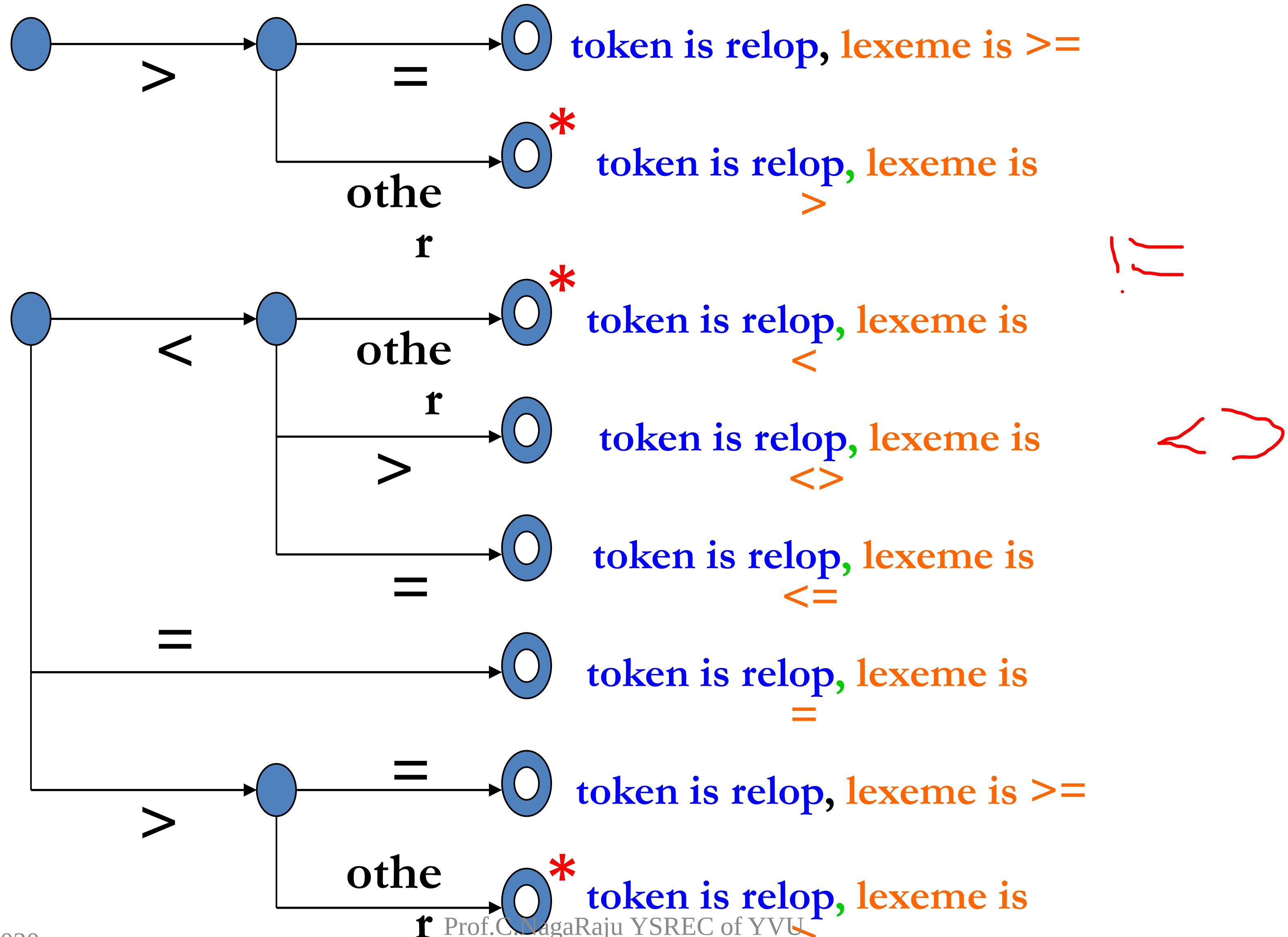



# Transition Diagram : White Spaces



# Transition Diagram : RELOP



# ❖ Regular Expressions

Regular Expressions are used to denote regular languages.

- ❖ An expression is regular if it satisfies the following conditions
- ❖ Let Σ be a Non-empty Alphabet.

1. ∈ is a regular expression
2. ∅ is a regular expression.
3. For each **a ∈ Σ** , **a** is a regular expression.
4. If **R1** and **R2** are regular expressions, then **R1 ∪ R2** is a regular expression.
5. If **R1** and **R2** are regular expressions, then **R1 ∩ R2** is a regular expression.
6. If R is a regular expression, then $\mathbf{R}^*$ is a regular expression.

- Rules for construction of Regular expressions
- where R is regular expression ∅ is empty set and ∈ is null set

**1)∅+R=R+ ∅=R**

**2) ∅.R=R .∅= ∅**

**3) ∅*= ∈**

**4) ∈.R=R. ∈=R**

**5) ∈*= ∈**

**6) ∈+R.R*=R*.R+ ∈=R***

**7)(a+b) *=(a*+b*) ***

**8)(a*.b*)=(a*b*) ***

**9)(a+b*)*=a*(ba)*=b*(ab*)***

## ❖ Regular languages

- ❖ A Languages defined by Regular Expressions are called Regular Languages.
- ❖ A Language is regular if and only if some regular expressions describes it ex: finite automata.
- ❖ Let Σ be a Non-empty Alphabet.

1. The Regular Expression Є describes the language {Є}.
2. The Regular Expression ∅ describes the language ∅.
3. For each a ∊ Σ, the Regular Expression a describes the language {a}.

- **Closure Properties of Regular Languages**

  **Union :** If L1 and L2 are two regular languages, their union $L1 \cup L2$ will also be regular.

- For example, $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$ $L3 = L1 \cup L2 = \{a^n \cup b^n \mid n \geq 0\}$ is also regular.
- **Intersection :** If L1 and L2 are two regular languages, their intersection $L1 \cap L2$ will also be regular.
- For example, $L1=\{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and $L2= \{b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$ $L3 = L1 \cap L2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.
- **Concatenation :** If L1 and L2 are two regular languages, their concatenation L1.L2 will also be regular.
- For example, $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$ $L3 = L1.L2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

- **Kleene Closure :** If L1 is a regular language then its Kleene closure $L1^*$ will also be regular.
- For example,

  $L1 = (a \cup b)$

  $L1^* = (a \cup b)^*$
- **Complement :** If L(G) is regular language, then its complement L'(G) will also be regular language.
- Complement of a language can be found by subtracting strings which are in L(G) from all possible strings.
- For example,

  $L(G) = \{a^n \mid n > 3\}$

  $L'(G) = \{a^n \mid n <= 3\}$

- **Question 1**
- Construct the regular expression over on alphabet S = {a, b} here language has exactly string length of "2"
- Answer:-
- $L_1=\{aa,ab,ba,bb\}$
- $=aa+ab+ba+bb$
- $=a(a+b)+b(a+b)$
- $=(a+b)(a+b)$

- **Question 2**
- Construct the regular expression over on alphabet S = {a, b} where string length is at least "2"
- Answer:-
- $L_1$={aa,ab,ba,bb,aaa-------}
- Example:- 2,3,4,5,6 -------------infinity lengths
- (a+b)(a+b)(a+b)*

- **Question 3**
- Construct the regular expression over on alphabet S = {a, b}    where  string length is at most  "2"
- Answers:-
- At most 2 means 0, 1, 2
- {Є,a,b,aa,ab,ba,bb}
- (a+b+ Є)(a+b+ Є)

- **Question 4**
- Construct the regular expression over on alphabet S = {a, b}   find even length strings
- Answer:-
- L={Є,aa,ab,ba,bb,aaaa,--------}
- $((a+b)(a+b))^*=((a+b)^2)^*=(a+b)^{2x}=(a+b)^{2n}$ ,where n>=0

- **Question 5**
- Construct the regular expression over on alphabet S = {a, b} where string length is odd
- Answer:- $(a+b)^{2n+1} | n0$
- $=(a+b)^{2n}(a+b)$
- $((a+b)^2)^*(a+b)$
- $((a+b)(a+b))^*(a+b)$

- **Question 6**
- Construct the regular expression over on alphabet S = {a, b} where string length which is divisible by'3'
- Answer:-
- L={0,3,6,9,12-------}
- ((a+b)(a+b)(a+b))*

- **Question 7**
- Construct the regular expression over on alphabet $\Sigma = \{a, b\}$ where the string starts with ‘a’.
- Answer:- a(a+b)*
- L={a,aa,ab,aab,aabb-------}

- **Question 8**
- Construct the regular expression over on alphabet S = {a, b} where string contains exactly 2 a's
- Answers:-
- b*ab*ab*

- **Question 9**
- Construct the regular expression over on alphabet S = {a, b} where the string starting and ending with different symbols
- Answers:-a(a+b)*b + b(a+b)*a

- **Question 10**
- Construct the regular expression over on alphabet S = {a, b} where string contains at most 2 a’s
- Answers:- $b^*(\epsilon + a)b^*(\epsilon + a)b^*$


Prof.C.NagaRaju YSREC of YVU
9949218570

- **Question 11**
- Construct the regular expression over on alphabet S = {a, b} where the string contains even a's
- Answers:- $(b^*ab^*ab^*)^*+b^*(b^*ab^*a)^*b^*$
- Where,
- $b^*=\{b,bb,bbb-------\}$

- **Question 12**
- Construct the regular expression over on alphabet S = {a, b} such that no 2a's and 2b's should not come together
- Answer:-
- L={Є,b,bbb,a,ab,aba,abab,ababab---------ba,bab,baba,babab-----------}
- (b+ab)* + (b+ab)*
- (Є,b,bb,bbb----------ab,abab)
- (b+ab)*(Є +a)

## Question 13

Construct the regular expression over on alphabet $\Sigma = \{a, b\}$ such that no 2a's and 2b's should not come together

L={E,a,b,ab,ba,aba,bab-------}

| | Start with | ends with |
|---|---|---|
| {a,aba,ababa----------} | a | a – (ab)*a (or) a(ba)* |
| {ab,abab,ababab-----} | a | b – (ab)* (or) a(ba)*b |
| {ba,baba---------------} | b | a – (ba)* (or) b(ab)*a |
| {babab,bababab------} | b | b – (ba)*b (or) b(ab)* |

=(ab)*a+(ab)*+b(ab)*a+b(ab)*

=(ab)*(a+ Є) + b(ab)*(a+ Є)

=(Є +b)(ab)*(Є +a)

=(Є +a)(ba)*(Є +b)

## ❖ On binary regular expressions

- ❖ 0 is a regular expression.
- ❖ 1 is a regular expression.
- ❖ If 0 and 1 are regular expressions then $0 \cup 1$ is a regular expression.
- ❖ if $0 \cup 1$ is a regular expression, $(0 \cup 1)^*$ is a regular expression.
- ❖ If 1 and 0 are regular expressions, 10 is a regular expression.

- If 10 and 1 are regular expressions, 101 is a regular expression.
- If $(0 \cup 1)^*$ and 101 are regular expressions, $(0 \cup 1)^*101$ is a regular expression.
- If $(0 \cup 1)^*101$ and $(0 \cup 1)^*$ are regular expressions, $(0 \cup 1)^*101(0 \cup 1)^*$ is a regular expression.

- Observe that this language is also described by the regular expression $01^* \cup 1^*$.
- The regular expression $1^*\emptyset$ describes the language $\emptyset$.
- The regular expression $\emptyset^*$ describes the language $\{\epsilon\}$.

# Operations On Languages

- The Concatenation of languages $L_1$ and $L_2$ is

$$L_1L_2 = \{st: s \in L_1, t \in L_2\}$$

- The $N$-th Power of $L^n$ is

$$L^n = \{s_1s_2...s_n: s_1, s_2, ..., s_n \in L\}$$

- The Union of $L_1$ and $L_2$ is

$$L_1 \cup L_2 = \{s: s \in L_1 \text{ or } s \in L_2\}$$

# Example
# String Concatenation

$s = 011$ $t = 101$

$st = 011101$

$ts = 101011$

$ss = 011011$

$sst = 011011101$

$$s = a_1 \ldots a_n \quad t = b_1 \ldots b_m \quad \Rightarrow \quad st = a_1 \ldots a_n b_1 \ldots b_m$$

# Example

$L_1 = \{0, 01\}$

$L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$

any number of 1s

$$
\begin{aligned}
L_1L_2 &= \{0, 01, 011, 0111, \ldots\} \cup \{01, 011, 0111, \ldots\} \\
&= \{0, 01, 011, 0111, \ldots\}
\end{aligned}
$$

0 followed by any number of 1s

$$L_1^{\ 2} = \{00, 001, 010, 0101\}$$

$$
\begin{aligned}
L_2^{\ 2} &= L_2 \\
L_2^{\ n} &= L_2 \quad (n \geq 1)
\end{aligned}
$$

$$L_1 \cup L_2 = \{0, 01, \varepsilon, 1, 11, 111, \ldots\}$$

# Operations on Languages

- The star of $L$ are all strings made up of zero or more chunks from $L$:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$$

  - ✓ This is always infinite, and always contains e

- Example: $L_1 = \{01,\ 0\}$, $L_2 = \{\varepsilon,\ 1,\ 11,\ 111,\ \ldots\}$. What is $L_1{}^*$ and $L_2{}^*$?

# Example

$L_1 = \{0, 01\}$

$L_2 = \{\varepsilon, 1, 11, 111, \ldots\}$

any number of 1s

$L_1^2 = \{00, 001, 010, 0101\}$

$L_1^*$: $0|01|0|0|0|01$ is in $L_1^*$

$00\underline{11}0001$ is not in $L_1^*$

$\underline{1}0010001$ is not in $L_1^*$

$L_1^*$ are all strings that start with 0 and do not contain consecutive 1s

$L_2^2 = L_2$

$L_2^n = L_2 \quad (n \geq 1)$

$$\begin{aligned} L_2^* &= L_2^0 \cup L_2^1 \cup L_2^2 \cup \ldots \\ &= \{\varepsilon\} \cup L_2^1 \cup L_2^2 \cup \ldots \\ &= L_2 \end{aligned}$$

$L_2^* = L_2$

# Constructing Languages With Operations

- Let's say $\Sigma = \{0, 1\}$
- We can construct languages by starting with simple ones, like $\{0\}$, $\{1\}$ and combining them

$\{0\}(\{0\} \cup \{1\})^*$ ⇨ $0(0+1)^*$

all strings that start with 0

$(\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$ ⇨ $01^*+10^*$

0 followed by any number of 1s, or
1 followed by any number of 0s

# Examples

$\Sigma = \{0, 1\}$

$$01^* = 0(1^*) = \{0, 01, 011, 0111, \ldots\}$$

0 followed by any number of 1s

$$(01^*)(01) = \{001, 0101, 01101, 011101, \ldots\}$$

0 followed by any number of 1s and then 01

# Examples

$0+1 = \{0, 1\}$ strings of length 1

$(0+1)^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$ any string

$(0+1)^*010$ any string that ends in 010

$(0+1)^*01(0+1)^*$ any string that contatins the pattern 01

# Examples

$((0+1)(0+1))^*+((0+1)(0+1)(0+1))^*$

all strings whose length is even or a mutliple of 3
= strings of length 0, 3, 6, 9, 12, ...

$((0+1)(0+1))^*$ strings of even length

$(0+1)(0+1)$ strings of length 2

$((0+1)(0+1)(0+1))^*$ strings of length a multiple of 3

$(0+1)(0+1)(0+1)$ strings of length 3

# Examples

$(0+1)(0+1)$ strings of length 2

$(0+1)(0+1)(0+1)$ strings of length 3

$(0+1)(0+1)+(0+1)(0+1)(0+1)$ strings of length 2 or 3

$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$

strings that can be broken in blocks,
where each block has length 2 or 3

# Examples

$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$

strings that can be broken in blocks, where each block has length 2 or 3

$\varepsilon$ ✓ 1 ✗ 10 ✓ 011 ✓ $\underbrace{00}\underbrace{110}$ ✓ $\underbrace{01}\underbrace{10}\underbrace{10}\underbrace{110}$ ✓

this includes all strings except those of length 1

$((0+1)(0+1)+(0+1)(0+1)(0+1))^* =$ all strings except 0 and 1

# Examples

$$\underbrace{(1+01+001)^*}\underbrace{(\varepsilon+0+00)}$$

ends in at most two 0s

there can be at most two 0s between consecutive 1s

---

there are never three consecutive 0s

**Guess:** $(1+01+001)^*(\varepsilon+0+00) = \{x \text{: } x \text{ does not contain } 000\}$

$\varepsilon$     00     01|1|001|01|1|0     001|001|0

# Examples

- Write a regular expression for all strings with two consecutive 0s.

$\Sigma = \{0, 1\}$

(anything) 00 (anything else)

$(0+1)^*00(0+1)^*$

# Examples

- Write a regular expression for all strings that do not contain two consecutive 0s.

$\Sigma = \{0, 1\}$

0110101101010

blocks ending in 1 last block

... at most one 0 in every block ending in 1 $(1 + 01)$

... and at most one 0 in the last block $(\varepsilon + 0)$

$\boxed{(1 + 01)^*(\varepsilon + 0)}$

# Examples

- Write a regular expression for all strings with an even number of 0s.

$\Sigma = \{0, 1\}$

even number of zeros = (two zeros)*

**two zeros** $= 1^*01^*01^*$

$\boxed{(1^*01^*01^*)^*}$

# GATE Questions



Prof.C.NagaRaju YSREC of YVU
9949218570

- 1)The number of tokens in the following C statement is
- printf("i = %d, &i = %x", i, &i);  (GATE 2000)

- A.3
- B.26
- C.10
- D.21

- 1)Printf   2) (   3)"i = %d, &i = %x“  4) ,  5) I  6),  7) & 8) I 9) ) 10) ;

•2) In a compiler, keywords of a language are recognized during (2011)

- A.parsing of the program
- B.the code generation
- C.the lexical analysis of the program
- D.dataflow analysis

- Answer is C

# GATE CS 2011 Lexical analysis

3) The lexical analysis for a modern computer language such as Java needs the power of which one of the following machine models in a necessary and sufficient sense?

A. Finite state automata

B. Deterministic pushdown automata

C. Non-Deterministic pushdown automata

D. Turing Machine

OPTION A

## 4) Consider the following statements:

(I) The output of a lexical analyzer is groups of characters.

(II) Total number of tokens in printf("i=%d, &i=%x", i, &i); are 11.

(III) Symbol table can be implementation by using array and hash table but not tree.

Which of the following statement(s) is/are correct?

A. Only (I)
B. Only (II) and (III)
C. All (I), (II), and (III)
D. None of these

OPTION D

5) Which one of the following statements is FALSE?

A. Context-free grammar can be used to specify both lexical and syntax rules.
B. Type checking is done before parsing.
C. High-level language programs can be translated to different Intermediate Representations.
D. Arguments to a function can be passed using the program stack.

Option B

- 7)The output of a lexical analyzer is

A. A parse tree
B. Intermediate code
C. Machine code
D. A stream of tokens

Option D

- 8) Consider the following statements related to compiler construction :
- I. Lexical Analysis is specified by context-free grammars and implemented by pushdown automata.
- II. Syntax Analysis is specified by regular expressions and implemented by finite-state machine.
- Which of the above statement(s) is/are correct ?
- Only I
- Only II
- Both I and II
- Neither I nor II
- Option D

- 9) Which of the following statement(s) regarding a linker software is/are true ?
- I A function of a linker is to combine several object modules into a single load module.
- II A function of a linker is to replace absolute references in an object module by symbolic references to locations in other modules.
- A) Only I B) Only II C) Both I and II
- D) Neither I nor II

- Option (A) is correct.

10) From the given data below

: a b b a a b b a a b which one of the following is not a word in the dictionary created by LZ-coding (the initial words are a, b)?

A. a b

B. b b

C. b a

D. b a a b

B and D are correct.

- 11) The number of tokens in the following C statement is

  printf("i=%d, &i=%x", i&i);

A. 13

B. 6

C. 10

D. 9

- printf ( "i=%d, &i=%x" , i & i ) ;
- 1 2 3 4 5 6 7 8 9
- Total nine tokens are present. So, correct option is (D)

- 12) In compiler optimization, operator strength reduction uses mathematical identities to replace slow math operations with faster operations. Which of the following code replacements is an illustration of operator strength reduction ?

A. Replace P + P by 2 * P or Replace 3 + 4 by 7.

B. Replace P * 32 by P < < 5

C. Replace P * 0 by 0

D. Replace (P < <4) – P by P * 15

option (B) is correct.

- 13) Debugger is a program that

A. allows to examine and modify the contents of registers
B. does not allow execution of a segment of program
C. allows to set breakpoints, execute a segment of program and display contents of register
D. All of the above

option (C) is correct.

Thank U