

## Lecture 3

# Lexical Analysis

# Introduction

- For an English sentence:
  - Read the sentence string
  - Recognize its parts of speech – *noun, verb, adjective, etc...*
  - Match these parts of speech against a grammar
- For computer languages:
  - Extract the words (lexemes) of the input string
  - Classify them according to their roles (parts of speech)
    - This is tokenization – finding token classes
  - Lexical analyzer recognizes the string by matching these tokens against a grammar

# Introduction

- Lexical Structure of a programming language consists of a set of token classes
  - Operators, whitespaces, keywords, identifiers, numbers, (, ), :, = etc...
- Lexical Analysis
  - Classify the program substrings into various token classes and communicate them to the parser.
- Goal – Given the specification for token classes of a language, *automatically construct a lexical analyzer*

# Regular Expressions

- Each token class consists of a set of strings.
- A distinct set of strings constitute a language over an alphabet .
  - Alphabet: A finite set of symbols by which the strings of the language may be formed
- They are *regular languages* (*recognized by FA*)
- Regular languages – defined by regular expressions (RE)
- RE's denote the set of strings recognized by a language by defining their *pattern*
- Automation steps:
  - RE  $\rightarrow$  NFA  $\rightarrow$  DFA  $\rightarrow$  Table-Driven Implementation

# Regular Expressions

- Two basic RE's
  - Single character RE – eg: 'B': A language containing one string which is the single character 'B'
  - Epsilon RE –  $\varepsilon$  : *A language which contains one string, the empty string.*
- Three compound RE's
  - Union – eg.:  $A + B$  or  $A \mid B$  (either A or B)
  - Concatenation – eg.:  $AB$   $\{ab \mid a \in A \wedge b \in B\}$
  - Iteration – eg.:  $A^*$  – 0 or more occurrences of A
    - $A^0 = \varepsilon$
- (A) has the same meaning as A

# Regular Language (Set)

- The regular expression over alphabet  $\Sigma$  are the smallest set of expressions including

$$R = \varepsilon$$

| 'c' for  $c \in \Sigma$  (alphabet)

|  $R + R$  (union)

|  $RR$  (concatenation)

|  $R^*$  (iteration)

# Regular Expressions

- Operator precedence:  $(A)$ ,  $A^*$ ,  $AB$ ,  $A|B$ 
  - So  $ab^*c|d$  is parsed as  $((a(b^*))c)|d$
- Describe the languages denoted by the following REs when alphabet  $(\Sigma) = \{0,1\}$ 
  - $1^*$ ;  $(1+0)1$ ;  $0^*+1^*$ ;  $01(01)^*$ ,  $(0+1)^*$ ,  $11(0+1)^*$
- Short-hands:
  - Character range:  $[a-d]$  for  $a | b | c | d$ ;
  - Atleast one:  $r+$  for  $rr^*$ ;
  - Option:  $r?$  for  $r | \epsilon$
  - Excluded range:  $[^a-z] = \text{Complement of } [a-z]$

# Regular Expressions

- Write RE's for:
  - Strings of 0's and 1's containing 00 as a substring
- Write RE's that:
  - Recognizes our email addresses at IITG
  - Recognizes a number consisting of: One or more digits, followed by an optional fractional part, followed by an optional exponent.



# Lexical Specification of a Language

1. Write REs for the lexemes of each token class:

Number = digit+

Keyword = 'if' + 'else' + ...

Identifier = letter (letter + digit)\*

...

2. Construct R, matching all lexemes for all tokens:

R = Keyword + Identifier + Number + ...

= R1 + R2 + ...

# Lexical Specification of a Language

1. Let input be  $x_1 \dots x_n$

For  $1 \leq i \leq n$  check

$$x_1 \dots x_i \in L(R)$$

2. If success, then we know that:

$$x_1 \dots x_i \in L(R_j) \text{ for some } j$$

3. Remove  $x_1 \dots x_i$  from input and go to (3)

# Lexical Specification of a Language

- How much of the input to use?
  - $x_1 \dots x_i \in L(R)$
  - $x_1 \dots x_j \in L(R), i \neq j$Solution: “Maximal Munch”
- What if more than one token matches?
  - $x_1 \dots x_i \in L(R_x)$
  - $x_1 \dots x_i \in L(R_y)$
  - Solution: Priority ordering – Choose the one listed first
- What to do when no rule matches?
  - Solution: Make special token class / classes for all error strings. Put it last in the priority list.

Thanks