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*
  - Syntax 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 → NFA → DFA → 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 – **ε** : A language which contains one string, the empty string.
- Three compound RE's
  - Union – eg.: A + B or A | B (Either A or B)
  - Concatenation – eg.: AB {ab | a ϵ A Λ b ϵ B}
  - Iteration – eg.: A* – 0 or more occurrences of A
    - $A^0 = \boldsymbol{\varepsilon}$
- *(A)* has the same meaning as *A*

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
  - a; (a + b) b; 1*; (0 | 1)*; (0 | 1)(0 | 1); (a*b*)*; (a | b)*baa;

- Short-hands:
  - Character range: [a-d] for a | b | c | d;
  - Atleast one: r^+ for rr*;
  - Option: r? for r | ε
  - Excluded range: [^a-z] = Complement of [a-z]

-

Regular Expressions

- Write RE’s for:
  - Strings of 0’s and 1’s containing 00 as a substring
  - Strings of 0’s and 1’s containing atmost one 0

- 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 + …

  = R_1 + R_2 + …

Lexical Specification of a Language

- 3. Let input be $\mathbf{x_1 \ldots x_n}$
  For **1 <= i <= n** *check*
  $\mathbf{x_1 \ldots x_i} \in \mathbf{L(R)}$

- 4. If success, then we know that:
  $\mathbf{x_1 \ldots x_i \in L(R_j)}$ for some **j**

- 5. Remove $\mathbf{x_1 \ldots x_i}$ from input and go to (3)

Lexical Specification – Issues

- *How much of the input to use?*

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

  $x_1 \ldots x_j \in L(R),\ i \neq j$

  Solution: "Maximal Munch"

- *What if more than one token matches?*

  $x_1 \ldots x_i \in L(R_x)$

  $x_1 \ldots x_i \in L(R_y)$

  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.

Next Lecture

Rest of Lexical Analysis