CS 442/542

Lexical Analysis
Scanning
Simplified Compiler Organization

- Scanning
- Parsing
- Code Generation
Lexical Analysis

• Languages
• Finite State Automata (FSA)
• Regular Expressions (RE)
• Algorithms
Languages

• Given an finite alphabet $\Sigma$ a language is a set of strings where each string is a finite sequence of 0 of more symbols for the alphabet.

• Example alphabets
  – $\{0, 1\}$
  – $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
Languages

• Example languages over \{0, 1\}
  – \{00, 01, 10, 11\}
  – \{0, 01, 011, 0111 ...\}
  – \{\epsilon, 1, 11, 111, 1111 ...\}
    • \(\epsilon\) is the empty string
  – \{\}\n  – \{\epsilon\}
    • The empty set and a set containing the empty string are different
Languages

• In this class we are interested in the languages that are used to define programming languages.
• There are two primary types of languages we will look at:
  – Regular languages
  – Context-free languages
• For the two types of languages we will look at notations to specify the language and at abstract machines to recognize strings in the languages.
Definition of a Regular Expression (RE)

- R is a regular expression if R is one of the following
  - 1. a for some a in the alphabet Σ
  - 2. ε
  - 3. ∅
  - 4. if R and S are regular expressions then R | S is a regular expression
  - 5. if R and S are regular expressions then R S is a regular expression
  - 6. if R is a regular expression then R*
Example Regular Expressions over the alphabet \{0, 1\}

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ 0 }</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 (0</td>
<td>1)</td>
</tr>
<tr>
<td>1*</td>
<td>{ x</td>
</tr>
<tr>
<td>(0</td>
<td>1) *</td>
</tr>
<tr>
<td>0*(10<em>10</em>)*</td>
<td>{ x</td>
</tr>
</tbody>
</table>
Definition of a Non-deterministic finite automata (NFA)

- A nondeterministic finite automaton is a 5-tuple \((S, \Sigma, \delta, s_0, S_A)\) where
  - 1. \(S\) is a finite set of states
  - 2. \(\Sigma\) is a finite set of symbols called an alphabet
  - 3. \(\delta: S \times \Sigma \cup \{\varepsilon\} \rightarrow P(S)\) is the transition function
  - 4. \(s_0 \in S\) is the start state
  - 5. \(S_A \subseteq S\) is the set of final or accept states
Example NFAs

(a) NFA for “a”

(b) NFA for “b”

(c) NFA for “ab”

(d) NFA for “a | b”

(e) NFA for “a*”
Example NFAs

(a) NFAs for "a", "b", and "c"

(b) NFA for "b | c"

(c) NFA for "(b | c)"

(d) NFA for "a(b | c)"
Example NFAs

(a) NFA for “a(b | c)∗” (With States Renumbered)
Definition of a Deterministic finite automata (DFA)

• A deterministic finite automaton is a 5 tuple \((S, \Sigma, \delta, s_0, S_A)\) where
  
  – 1. \(S\) is a finite set of states
  – 2. \(\Sigma\) is a finite set of symbols called an alphabet
  – 3. \(\delta: S \times \Sigma \rightarrow S\) is the transition function
  – 4. \(s_0 \in S\) is the start state
  – 5. \(S_A \subseteq S\) is the set of final or accept states
Example DFA

(a) Resulting DFA
DFA

\((S, \Sigma, \delta, s_0, S_A)\)

- \(S = \{d_0, d_1, d_2, d_3\}\)
- \(\Sigma = \{a, b, c\}\)
- \(\delta = \{(d_0, a, d_1), (d_1, b, d_2), (d_1, c, d_3), (d_2, b, d_2), (d_2, c, d_3), (d_3, b, d_2), (d_3, c, d_3)\}\)
- \(S_0 = d_0\)
- \(S_A = \{d_1, d_2, d_3\}\)
Algorithms

• Build a NFA from a RE
• Build a DFA from an NFA
• Build a minimized DFA from a DFA
• Build an RE from a DFA
Algorithms

Kleene’s Construction

DFA Minimization

RE

DFA

NFA

Thompson’s Construction

Subset Construction

Code for a scanner
Scanner

- Input: Stream Characters
- Output: Stream of tokens or words
Scanner Generator

• Input: regular expressions specifying the tokens of a language
• Output: either the minimized DFA or a program that includes the minimized DFA and code that uses the minimized DFA to produce a stream of tokens given an stream of characters as input