CMSC 330: Organization of Programming Languages

Theory of Regular Expressions
Finite Automata
Regular Expression Review

• Terms
  – Alphabet
  – String
  – Language
  – Regular expression ("regex")

• Operations
  – Concatentation
  – Union
  – Kleene closure

• Ruby vs. theoretical regexes
Previous Course Review

• \{s \mid \text{cond}(s)\} means the set of all strings s such that the given condition applies
• \(s \in A\) means s is an element of the set A
• De Morgan’s Laws:
  \[(A \cap B)^C = A^C \cup B^C\]
  \[(A \cup B)^C = A^C \cap B^C\]
• Quantifiers and Qualifiers
  – Existential quantifier ("there exists"): \(\exists\)
  – Universal quantifier ("for all"): \(\forall\)
  – Qualifier ("such that"): \(\text{s.t.}\)
Implementing Regular Expressions

- We can implement regular expressions by turning them into a *finite automaton*
  - A “machine” for recognizing a regular language
Example

• Machine starts in start or initial state
• Repeat until the end of the string is reached:
  – Scan the next symbol s of the string
  – Take transition edge labeled with s
• The string is accepted if the automaton is in a final or accepting state when the end of the string is reached
Example

\[
\begin{array}{cccccc}
0 & 0 & 1 & 0 & 1 & 1 \\
\hline
0 &  &  &  &  & \\
1 &  &  &  &  & \\
\end{array}
\]
Example

0 0 1 0 1 0

not accepted
What Language is This?

- All strings over $\Sigma = \{0, 1\}$ that end in 1
- What is a regular expression for this language? $(0|1)^*1$
Formal Definition

- A deterministic finite automaton (DFA) is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where
  - \(\Sigma\) is an alphabet
  - \(Q\) is a nonempty set of states
  - \(q_0 \in Q\) is the start state
  - \(F \subseteq Q\) is the set of final states
  - \(\delta : Q \times \Sigma \rightarrow Q\) specifies the DFA's transitions
More on DFAs

- A finite state automaton can have more than one final state:

- A string is accepted as long as there is at least one path to a final state
Our Example, Stated Formally

\[ \Sigma = \{0, 1\} \]
Q = \{S0, S1\}
q_0 = S0
F = \{S1\}
\[ \delta = \{ (S0, 0, S0), (S0, 1, S1), (S1, 0, S0), (S1, 1, S1) \} \]
Another Example

(a,b,c notation shorthand for three self loops)

<table>
<thead>
<tr>
<th>string</th>
<th>state at end</th>
<th>accepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabcc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>acc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>bbc</td>
<td>S2</td>
<td>Y</td>
</tr>
<tr>
<td>aabbb</td>
<td>S1</td>
<td>Y</td>
</tr>
<tr>
<td>aa</td>
<td>S0</td>
<td>Y</td>
</tr>
<tr>
<td>ε</td>
<td>S0</td>
<td>Y</td>
</tr>
<tr>
<td>acba</td>
<td>S3</td>
<td>N</td>
</tr>
</tbody>
</table>
What language does this DFA accept? \(a^*b^*c^*\)

\(S_3\) is a dead state – a nonfinal state with no transition to another state.
Shorthand Notation

• If a transition is omitted, assume it goes to a dead state that is not shown

Language?

Strings over \{0,1,2,3\} with alternating even and odd digits, beginning with odd digit
What Lang. Does This DFA Accept?

a*b*c* again, so DFAs are not unique
These DFAs are equivalent in the sense that they accept the same language.
DFA State Semantics

- $S_0$ = “Haven’t seen anything yet” OR “seen zero or more b’s” OR “Last symbol seen was a b”
- $S_1$ = “Last symbol seen was an a”
- $S_2$ = “Last two symbols seen were ab”
- $S_3$ = “Last three symbols seen were abb”

• Language?
• $(a|b)^*abb$
Review

• Basic parts of a regular expression?
  
  concatenation, |, *, ε, Ø, {a}

• What does a DFA do?
  
  implements regular expressions; accepts strings

• Basic parts of a DFA?
  
  alphabet, set of states, start state, final states, transition function (Σ, Q, q₀, F, δ)
Notes about the DFA definition

• Can not have more than one transition leaving a state on the same symbol
  – the transition function must be a valid function

• Can not have transitions with no or empty labels
  – the transitions must be labeled by alphabet symbols
Nondeterministic Finite Automata (NFA)

- An NFA is a 5-tuple \((\Sigma, Q, q_0, F, \delta)\) where
  - \(\Sigma\) is an alphabet
  - \(Q\) is a nonempty set of states
  - \(q_0 \in Q\) is the start state
  - \(F \subseteq Q\) is the set of final states
  - \(\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Q\) specifies the NFA's transitions
    - Transitions on \(\varepsilon\) are allowed – can optionally take these transitions without consuming any input
    - Can have more than one transition for a given state and symbol

- An NFA accepts \(s\) if there is at least one path from its start to final state on \(s\)
NFA for \((a|b)^*abb\)

- **ba**
  - Has paths to either S0 or S1
  - Neither is final, so rejected

- **babaabb**
  - Has paths to different states
  - One leads to S3, so accepted
Why are NFAs useful?

- Sometimes an NFA is much easier to understand than its equivalent DFA.
Another example DFA

- Language?
- \((ab|aba)^*\)
NFA for \((ab|aba)^*\)

- **aba**
  - Has paths to states \(S_0, S_1\)
- **ababa**
  - Has paths to \(S_0, S_1\)
  - Need to use \(\varepsilon\)-transition
Relating R.E.'s to DFAs and NFAs

• Regular expressions, NFAs, and DFAs accept the same languages!

DFA \rightarrow NFA

RE \rightarrow DFA

can transform

can transform

can transform

(we’ll discuss this next)
Reducing Regular Expressions to NFAs

• Goal: Given regular expression e, construct NFA: $<e> = (\Sigma, Q, q_0, F, \delta)$
  – Remember, REs are defined inductively; i.e. recursively

• Base case: a

$<a> = (\{a\}, \{S0, S1\}, S0, \{S1\}, \{(S0, a, S1)\})$
Reduction (cont’d)

• Base case: $\varepsilon$

\[\langle \varepsilon \rangle = (\varepsilon, \{S0\}, S0, \{S0\}, \emptyset)\]

• Base case: $\emptyset$

\[\langle \emptyset \rangle = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)\]
Reduction (cont’d)

• Concatenation: AB
Reduction (cont’d)

• Concatenation: $AB$

$$<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$$

$$<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$$

$$<AB> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \varepsilon, q_B)\})$$
Practice

• Draw the NFA for these regular expressions using the reduction method:
  – ab
  – hello

• Write the formal (5-tuple) NFA for the same regular expressions
Reduction (cont’d)

• Union: \((A \cup B)\)
• **Union:** \((A \mid B)\)

- \(<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)\)
- \(<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)\)
- \(<(A \mid B)> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B \cup \{S0,S1\}, S0, \{S1\}, \delta_A \cup \delta_B \cup \{(S0,\epsilon,q_A), (S0,\epsilon,q_B), (f_A,\epsilon,S1), (f_B,\epsilon,S1)\})\)
Practice

• Draw the NFA for these regular expressions using exactly the reduction method:
  – ab|bc
  – hello|hi

• Write the formal NFA for the same regular expressions
Reduction (cont’d)

• Closure: $A^*$
Reduction (cont’d)

• Closure: $A^*$

- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<A^*> = (\Sigma_A, Q_A \cup \{S0,S1\}, S0, \{S1\},$
  $\delta_A \cup \{(f_A,\epsilon,S1), (S0,\epsilon,q_A), (S0,\epsilon,S1), (S1,\epsilon,S0)\})$
Practice

• Draw the NFA for these regular expressions using exactly the reduction method:
  – (ab|bc*)*
  – hello|(hi)*

• Write the formal NFA for the same regular expressions
Reduction Complexity

• Given a regular expression $A$ of size $n$...
  Size = # of symbols + # of operations

• How many states+transitions does $<A>$ have?
  – 2+1 for each symbol
  – 0+1 for each concatenation
  – 2+4 added for each union
  – 2+4 added for each closure
  – $O(n)$
  – That’s pretty good!
\(\epsilon\)-closure

- We say: \(p \xrightarrow{\epsilon} q\)
  - if it is possible to transition from state \(p\) to state \(q\) taking only \(\epsilon\)-transitions
  - if \(\exists\ p, p_1, p_2, \ldots\ p_n, q \in Q\ (p \neq q)\) such that
    \[\{p, \epsilon, p_1\} \in \delta, \{p_1, \epsilon, p_2\} \in \delta, \ldots, \{p_n, \epsilon, q\} \in \delta\]

- For any state \(p\), the \(\epsilon\)-closure of \(p\) is defined as the set of states \(q\) such that
  \[\{q \mid p \xrightarrow{\epsilon} q\}\]
Example

• What’s the $\varepsilon$-closure of S2 in this NFA?

• \{S2, S0\}