CMSC 330: Organization of Programming Languages

Finite Automata

Last Lecture
- Languages
  - Sets of strings
  - Operations on languages
- Regular expressions
  - Constants
  - Operators
  - Precedence

This Lecture
- Finite automata
  - States
  - Transitions
  - Examples
- Types
  - Deterministic (DFA)
  - Non-deterministic (NFA)

Implementing Regular Expressions
- We can implement a regular expression by turning it into a finite automaton
  - A "machine" for recognizing a regular language

"String" "String" "String" "String"
Yes No

Finite Automata
- 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 $s$
- String is accepted if automaton is in final state when end of string reached

Finite Automata: States
- Start state
  - State with incoming transition from no other state
  - Can have only 1 start state
- Final state
  - State with double circle
  - Can have 0 or more final states
What Language is This?

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

Dead 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
**Practice**

Give the English descriptions and the DFA or regular expression of the following languages:

- ((0|1)(0|1)(0|1)(0|1)(0|1))^*
  - All strings with length a multiple of 5
- (01)^*|(10)^*|(01)^*0|(10)^*1
  - All alternating binary strings

**Practice**

Give the regular expressions and finite automata for the following languages:

- You and your neighbors’ names
- All protein-coding DNA strings (including only ATCG and appearing in multiples of 3)
- All binary strings containing an even length substring of all 1’s
- All binary strings containing exactly two 1’s
- All binary strings that start and end with the same number

**Types of Finite Automata**

- **Deterministic Finite Automata (DFA)**
  - Exactly one sequence of steps for each string
  - All examples so far

- **Nondeterministic Finite Automata (NFA)**
  - May have many sequences of steps for each string
  - More compact

**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
  - How many can there be?
  - \(\delta\) is what's this definition saying that \(\delta\) is?
- A DFA accepts \(s\) if it stops at a final state on \(s\)
Formal Definition: Example

- $\Sigma = \{0, 1\}$
- $Q = \{S_0, S_1\}$
- $q_0 = S_0$
- $F = \{S_1\}$

\[\delta\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_0$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$S_1$</td>
</tr>
</tbody>
</table>

DFA Requirements

- Cannot have more than one transition leaving a state on the same symbol
  - I.e., transition function must be a valid function
- Cannot have transitions with empty labels
  - Transitions must be labeled by alphabet symbols
- NFAs do not have these requirements!
  - DFA is a special case of NFA

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
  - $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ specifies the NFA’s transitions
  - Transitions on $\epsilon$ 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 $S_0$ or $S_1$
  - Neither is final, so rejected
- $babaabb$
  - Has paths to different states
  - One leads to $S_3$, so accepted

Another example DFA

- Language?
  - $(a|b|aba)^*$

NFA for $(a|b|aba)^*$

- $aba$
  - Has paths to states $S_0$, $S_1$
- $ababa$
  - Has paths to $S_0$, $S_1$
  - Need to use $\epsilon$-transition
Relating REs to DFAs and NFAs

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

Reducing Regular Expressions to NFAs

- Goal: Given regular expression $e$, construct NFA: $<e> = (\Sigma, Q, q_0, F, \delta)$
  - Remember regular expressions are defined recursively from primitive RE languages
  - Invariant: $|F| = 1$ in our NFAs
    - Recall $F$ = set of final states

- Base case: $a$
  - $<a> = (\{a\}, \{S0, S1\}, S0, \{S1\}, \{(S0, a, S1)\})$

Reduction (cont.)

- Base case: $\epsilon$
  - $<\epsilon> = (\epsilon, \{S0, S0\}, S0, \{S1\}, \emptyset)$

- Base case: $\emptyset$
  - $<\emptyset> = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$

Reduction: Concatenation

- Induction: $AB$
  - $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
  - $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
  - $<AB> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_0, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \epsilon, q_B)\})$

Reduction: Union

- Induction: $(A|B)$
Reduction: Union (cont.)

- Induction: \((A \cup B)\)

\[
\begin{align*}
\langle A \rangle &= (\Sigma_A, Q_A, q_A, f_A, \delta_A) \\
\langle B \rangle &= (\Sigma_B, Q_B, q_B, f_B, \delta_B) \\
\langle (A \cup B) \rangle &= (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, (S_0, S_1), S_0, (S_1), \\
& \quad \delta_A \cup \delta_B ((S_0, \epsilon, q_A), (S_0, \epsilon, S_1), (f_A, \epsilon, S_1)))
\end{align*}
\]

Reduction: Closure

- Induction: \(A^*\)

\[
\begin{align*}
\langle A \rangle &= (\Sigma_A, Q_A, q_A, f_A, \delta_A) \\
\langle A^* \rangle &= (\Sigma_A, Q_A \cup \{S_0, S_1\}, S_0, \{S_1\}, \\
& \quad \delta_A \cup \{(f_A, \epsilon, S_1), (S_0, \epsilon, q_A), (S_0, \epsilon, S_1), (S_1, \epsilon, S_0)\})
\end{align*}
\]

Reduction: Closure (cont.)

- Induction: \(A^*\)

\[
\begin{align*}
\langle A \rangle &= (\Sigma_A, Q_A, q_A, f_A, \delta_A) \\
\langle A^* \rangle &= (\Sigma_A, Q_A \cup \{S_0, S_1\}, S_0, \{S_1\}, \\
& \quad \delta_A \cup \{(f_A, \epsilon, S_1), (S_0, \epsilon, q_A), (S_0, \epsilon, S_1), (S_1, \epsilon, S_0)\})
\end{align*}
\]

Reduction Complexity

- Given a regular expression \(A\) of size \(n\)... 
  \begin{align*}
  \text{Size} &= \text{# of symbols} + \text{# of operations} \\
  \end{align*}

- How many states does \(\langle A \rangle\) have?
  
  - 2 added for each \(|\), 2 added for each \(*\)
  
  - \(O(n)\)
  
  - That’s pretty good!

Practice

- Draw NFAs for the following regular expressions and languages
  
  - \((0|1)^*110^*\)
  
  - \(101^*111\)
  
  - all binary strings ending in 1 (odd numbers)
  
  - all alphabetic strings which come after “hello” in alphabetic order
  
  - \((ab^*cd^*ejab)d\)

Summary

- Finite automata
  
  - Deterministic (DFA)
  
  - Non-deterministic (NFA)

- Questions
  
  - How are DFAs and NFAs different?
  
  - When does an NFA accept a string?
  
  - How to convert regular expression to an NFA?