CMSC 330: Organization of Programming Languages

Finite Automata

Last Lecture

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

Clarifications from last time

- Regular expression: $(0|1|...|9)^*330(0|1|...|9)^*$
  - When talking about formal definition of REs, describe a set of strings exactly
  - When used in Ruby (or other languages) are often used for substring matching

- Implementation of REs
  - Parse from left to right
  - Keep track of whether current symbol is part of RE
  - This brings us to today’s lecture…

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”  “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

Finite Automaton: Example 1

```
0 0 1 0 1 1
```

accepted
Finite Automaton: Example 2

0 0 1 0 1 0
not accepted

What Language is This?

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

\( (0|1)^*1 \)

Finite Automaton: Example 3

(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>aabbc</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>c</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 FA accept?

\( a^*b^*c^* \)

S3 is a dead state – a nonfinal state with no transition to another state
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.

What Lang. Does This FA Accept?

- \(a^*b^*c^*\) again, so DFAs are not unique.

Practice

Give the English descriptions for the FA 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

Finite Automaton: Example 4

- Description for each state:
  - S0 = “Haven’t seen anything yet” OR “seen zero or more b’s” OR “Last symbol seen was a b”
  - S1 = “Last symbol seen was an a”
  - S2 = “Last two symbols seen were ab”
  - S3 = “Last three symbols seen were abb”
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 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
  - the strings recognized by the DFA are over this set
- \(Q\) is a nonempty set of states
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final states
  - How many can there be?
- \(\delta : Q \times \Sigma \rightarrow Q\) specifies the DFA’s transitions
  - What’s this definition saying that \(\delta\) is?

A DFA accepts \(s\) if it stops at a final state on \(s\)

Formal Definition: Example

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

\[
\begin{array}{c|cc}
\delta & 0 & 1 \\
\hline
S_0 & S_0 & S_1 \\
S_1 & S_0 & S_1 \\
\end{array}
\]
### DFA Requirements

- Can not have more than one transition leaving a state on the same symbol
  - I.e., transition function must be a valid function
- Can not 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 S0 or S1
  - Neither is final, so rejected
- **babaabb**
  - Has paths to different states
  - One leads to S3, so accepted

### 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 \(\epsilon\)-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: \(a\)
  - \(<a> = ([a], \{S_0, S_1\}, S_0, \{S_1\}, \{(S_0, a, S_1)\})\)

Reduction (cont.)

- Base case: \(\epsilon\)
  - \(<\epsilon> = (\epsilon, \{S_0\}, S_0, \{S_0\}, \emptyset)\)
- Base case: \(\emptyset\)
  - \(<\emptyset> = (\emptyset, \{S_0, S_1\}, S_0, \{S_1\}, \emptyset)\)
Reduction: Concatenation

Induction: \( AB \)

\[ <A> = (\Sigma_A, Q_A, q_A, f_A, \delta_A) \]
\[ <B> = (\Sigma_B, Q_B, q_B, f_B, \delta_B) \]
\[ <AB> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, f_B, \delta_A \cup \delta_B \cup \{(f_A, \epsilon, q_B)\}) \]

Reduction: Union

Induction: \( (A|B) \)

\[ <A> = (\Sigma_A, Q_A, q_A, f_A, \delta_A) \]
\[ <B> = (\Sigma_B, Q_B, q_B, f_B, \delta_B) \]
\[ <(A|B)> = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(S_0, \epsilon, q_A), (S_0, \epsilon, S_1), (f_A, \epsilon, S_1), (f_B, \epsilon, S_1)\}) \]
Reduction: Closure

Induction: $A^*$

\[ q_A \rightarrow \cdots \rightarrow i_A \]

Reduction: Closure (cont.)

Induction: $A^*$

\[ \cdot <A^* = (\Sigma_A, Q_A \cup \{S_0, S_1\}, S_0, \{S_1\}, \delta_A \cup \{(f_A, \epsilon, S_1), (S_0, \epsilon, q_A), (S_0, \epsilon, S_1), (S_1, \epsilon, S_0)\}) \]

Reduction Complexity

Given a regular expression $A$ of size $n$...

Size = # of symbols + # of operations

How many states does $<A>$ 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)
• $(ab^*c|d^*a|ab)d$
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?