Finite state automaton (FSA)

LING 570
Fei Xia
FSA / FST

• It is an important technique in NLP.

• Multiple FSAs/FSTs can be combined to form a larger, more powerful FSAs/FSTs.

• Any regular language can be recognized by an FSA.

• Any regular relation can be recognized by an FST.
FST Toolkits

- AT&T: http://www.research.att.com/~fsmtools/fsm
- NLTK: http://nltk.sf.net/docs.html
- ISI: Carmel
- ...
- ...
Outline

• Deterministic FSA (DFA)

• Non-deterministic FSA (NFA)

• Probabilistic FSA (PFA)

• Weighted FSA (WFA)
Definition of DFA

An automaton is a 5-tuple = $(\Sigma, Q, q_0, F, \delta)$

- An alphabet input symbols $\Sigma$
- A finite set of states $Q$
- A start state $q_0$
- A set of final states $F$
- A transition function:

$$\delta : Q \times \Sigma \rightarrow Q$$
\[ \Sigma = \{a, b\} \]
\[ S = \{q_0, q_1\} \]
\[ F = \{q_1\} \]
\[ \delta = \{ q_0 \times a \rightarrow q_0, \]
\[ q_0 \times b \rightarrow q_1, \]
\[ q_1 \times b \rightarrow q_1 \} \]

What about \( q_1 \times a \)?
Representing an FSA as a directed graph

• The vertices denote states:
  – Final states are represented as two concentric circles.

• The transitions forms the edges.

• The edges are labeled with symbols.
An example

\[
\begin{array}{cccccc}
q_0 & a & b & b & a & a \\
q_0 & q_0 & q_1 & q_1 & q_2 & q_0 \\
q_0 & q_0 & q_1 & q_1 & q_2 & q_0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
a & b & b & a & b \\
a & b & b & a & b \\
q_0 & q_0 & q_1 & q_1 & q_2 & q_1 \\
\end{array}
\]
DFA as an acceptor

• A string is said to be **accepted** by an FSA if the FSA is in a **final** state when it stops working.
  – that is, there is a path from the initial state to a final state which yields the string.
  – Ex: does the FSA accept “abab”?

• The set of the strings that can be accepted by an FSA is called the language accepted by the FSA.
An example

FSA:

Regular language: \{b, ab, bb, aab, abb, \ldots\}

Regular expression: \(a^* \ b^+\)

Regular grammar:

\[
\begin{align*}
q_0 & \rightarrow a \ q_0 \\
q_0 & \rightarrow b \ q_1 \\
q_1 & \rightarrow b \ q_1 \\
q_1 & \rightarrow \epsilon
\end{align*}
\]
NFA
NFA

• A transition can lead to more than one state.
• There could be multiple start states.
• Transitions can be labeled with $\epsilon$, meaning states can be reached without reading any input.

$\Rightarrow$ now the transition function is:

$$S \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^S$$
NFA example

\[
\begin{array}{cccccccc}
q_0 & q_0 & q_1 & q_1 & q_2 & q_1 \\
q_0 & q_1 & q_2 & q_0 & q_0 & q_0
\end{array}
\]

\[
\begin{array}{cccccccc}
q_0 & q_1 & q_2 & q_0 & q_0 & q_1 \\
q_0 & q_1 & q_2 & q_0 & q_1 & q_2
\end{array}
\]
Relation between DFA and NFA

• DFA and NFA are equivalent.

• The conversion from NFA to DFA:
  – Create a new state for each equivalent class in NFA
  – The max number of states in DFA is $2^N$, where $N$ is the number of states in NFA.

• Why do we need both?
Regular grammar and FSA

- Regular grammar: \((N, \Sigma, P, S)\)

- FSA: \((\Sigma, Q, q_0, F, \delta)\)

- Conversion between the two
Common algorithms for FSA packages

• Converting regular expressions to NFAs

• Converting NFAs to regular expressions

• Determinization: converting NFA to DFA

• Other useful closure properties: union, concatenation, Kleene closure, intersection
So far

• A DFA is a 5-tuple: \((\Sigma, Q, q_0, F, \delta)\)

• A NFA is a 5-tuple: \((\Sigma, Q, I, F, \delta)\)

• DFA and NFA are equivalent.

• Any regular language can be recognized by an FSA.
  
  – Reg lang \(\Leftrightarrow\) Regex \(\Leftrightarrow\) NFA \(\Leftrightarrow\) DFA \(\Leftrightarrow\) Reg grammar
Outline

• Deterministic finite state automata (DFA)

• Non-deterministic finite state automata (NFA)

• Probabilistic finite state automata (PFA)

• Weighted Finite state automata (WFA)
An example of PFA

\[ P(ab^n) = I(q_0) \cdot P(q_0, ab^n, q_1) \cdot F(q_1) \]
\[ = 1.0 \cdot (1.0 \cdot 0.8^n) \cdot 0.2 \]

\[ \sum_x P(x) = \sum_{n=0}^{\infty} P(ab^n) = 0.2 \cdot \sum_{n=0}^{\infty} 0.8^n = 0.2 \cdot \frac{0.8^0}{1-0.8} = 1 \]
Formal definition of PFA

A PFA is \( (Q, \Sigma, I, F, \delta, P) \)

- **Q**: a finite set of \( N \) states
- **\( \Sigma \)**: a finite set of input symbols
- **I**: \( Q \rightarrow \mathbb{R}^+ \) (initial-state **probabilities**)
- **F**: \( Q \rightarrow \mathbb{R}^+ \) (final-state **probabilities**)
- \( \delta \subseteq Q \times (\Sigma \cup \{ \varepsilon \}) \times Q \) : the transition relation between states.
- **P**: \( \delta \rightarrow R^+ \) (transition **probabilities**)
Constraints on function:

\[
\sum_{q \in Q} I(q) = 1
\]

\[
\forall q \in Q \quad F(q) + \sum_{a \in \Sigma \cup \{ \varepsilon \}, q' \in Q} P(q, a, q') = 1
\]

Probability of a string:

\[
P(w_{1,n}, q_{1,n+1}) = I(q_1) \cdot F(q_{n+1}) \cdot \prod_{i=1}^{n} p(q_i, w_i, q_{i+1})
\]

\[
P(w_{1,n}) = \sum_{q_{1,n+1}} P(w_{1,n}, q_{1,n+1})
\]
PFA

• Informally, in a PFA, each arc is associated with a probability.

• The probability of a path is the multiplication of the arcs on the path.

• The probability of a string $x$ is the sum of the probabilities of all the paths for $x$.

• Tasks:
  – Given a string $x$, find the best path for $x$.
  – Given a string $x$, find the probability of $x$ in a PFA.
  – Find the string with the highest probability in a PFA
  – …
Weighted finite-state automata (WFA)

- Each arc is associated with a weight.
- “Addition” and “Multiplication” can have other meanings.

\[
\text{weight}(x) = \bigoplus_{s,\ldots,t \in Q} \left( I(s) \otimes F(t) \otimes P(s, x, t) \right)
\]

- In PFA:

\[
P(w_{1,n}, q_{1,n+1}) = I(q_1) \ast F(q_{n+1}) \ast \prod_{i=1}^{n} p(q_i, w_i, q_{i+1})
\]

\[
P(w_{1,n}) = \sum_{q_{1,n+1}} P(w_{1,n}, q_{1,n+1})
\]
A semiring is a set $R$ equipped with two binary operations $+$ (i.e., $\oplus$) and $\cdot$ (i.e., $\otimes$), called addition and multiplication, such that:

1. $(R, +)$ is a commutative monoid with identity element 0:
   - $(a + b) + c = a + (b + c)$
   - $0 + a = a + 0 = a$
   - $a + b = b + a$

2. $(R, \cdot)$ is a monoid with identity element 1:
   - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
   - $1 \cdot a = a \cdot 1 = a$

3. Multiplication left and right distributes over addition:
   - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   - $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$

4. Multiplication by 0 annihilates $R$:
   - $0 \cdot a = a \cdot 0 = 0$
Hw3 bonus points (10 points)

<table>
<thead>
<tr>
<th>Set R</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>0</th>
<th>1</th>
<th>Arc weight</th>
<th>weight (x) is</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 1]</td>
<td>+</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>prob</td>
<td>Prob of x</td>
</tr>
<tr>
<td>[0, 1]</td>
<td>??</td>
<td>??</td>
<td>??</td>
<td>??</td>
<td>prob</td>
<td>Prob of the best path for x</td>
</tr>
<tr>
<td>$R \cup {+\infty, -\infty}$</td>
<td>min</td>
<td>+</td>
<td>??</td>
<td>??</td>
<td>distance</td>
<td>Shortest distance</td>
</tr>
<tr>
<td>$R \cup {+\infty, -\infty}$</td>
<td>max</td>
<td>+</td>
<td>??</td>
<td>??</td>
<td>distance</td>
<td>Longest distance</td>
</tr>
<tr>
<td>N</td>
<td>+</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>??</td>
<td>Number of paths</td>
</tr>
</tbody>
</table>

$weight(x) = \oplus_{s,...,t \in Q} (I(s) \otimes P(s, x, t) \otimes F(t))$

$x$ is the input string. Let’s ignore $I(s)$ and $F(t)$. 
Summary

• DFA and NFA are 5-tuple: \((\Sigma, Q, I, F, \delta)\)
  – They are equivalent
  – Algorithm for constructing NFAs for Regexps

• PFA and WFA are 6-tuple: \((Q, \Sigma, I, F, \delta, P)\)

• Existing packages for FSA/FSM algorithms:
  – Ex: intersection, union, Kleene closure, difference, complementation, …
Two Views of FSAs

• Recognition: An FSA is a model that, given an input string, accepts the string if it is in the language, and rejects otherwise.

• Generation: An FSA $m$ is a model that can generate all and only the strings in $L(m)$. 
Additional slides
An algorithm for deterministic recognition of DFAs

```plaintext
function D-RECOGNIZE(tape, machine) returns accept or reject
  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```
Definition of regular expression

• The set of regular expressions is defined as follows:
  (1) Every symbol of $\Sigma$ is a regular expression
  (2) $\varepsilon$ is a regular expression
  (3) If $r_1$ and $r_2$ are regular expressions, so are $(r_1)$, $r_1r_2$, $r_1|r_2$, $r_1^*$
  (4) Nothing else is a regular expression.
Regular expression \( \rightarrow \) NFA

Base case:

- (a) \( r = \epsilon \)
- (b) \( r = \emptyset \)
- (c) \( r = a \)

Concatenation: connecting the final states of FSA\(_1\) to the initial state of FSA\(_2\) by an \( \epsilon \)-translation.

Union: Creating a new initial state and add \( \epsilon \)-transitions from it to the initial states of FSA\(_1\) and FSA\(_2\).

Kleene closure:
Regular expression $\rightarrow$ NFA (cont)

Kleene closure:

An example: $\backslash d+\(\backslash .\backslash d+\)?(e\-?\backslash d+)?$