Finite state automaton (FSA)

LING 570

Fei Xia

Week 2: 10/07/09

FSA / FST

- It is one of the most important techniques 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/man.html
- NLTK: http://nltk.sf.net/docs.html
- ISI: Carmel
- …

Outline

- Deterministic FSA (DFA)
- Non-deterministic FSA (NFA)
- Probabilistic FSA (PFA)
- Weighted FSA (WFA)

DFA

Definition of DFA

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

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

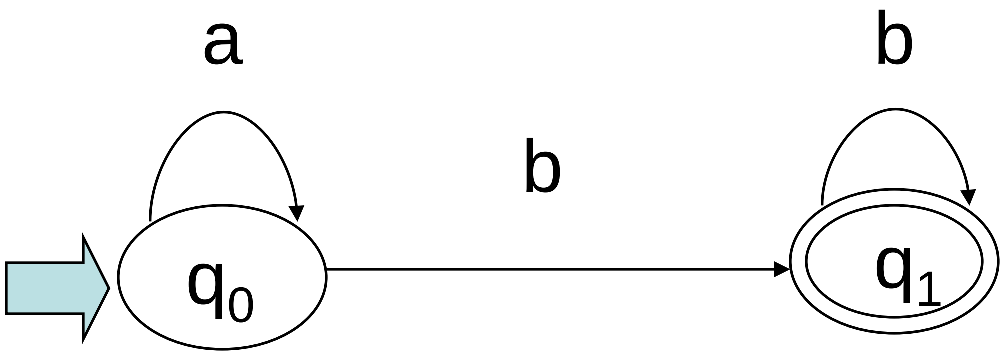
$$\delta : Q \times \Sigma \rightarrow Q$$

$\Sigma = \{a, b\}$

$S = \{q_0, q_1\}$

$F = \{q_1\}$

$$\delta = \{ q_0 \times a \rightarrow q_0,$$
$$q_0 \times b \rightarrow q_1,$$
$$q_1 \times b \rightarrow q_1 \}$$

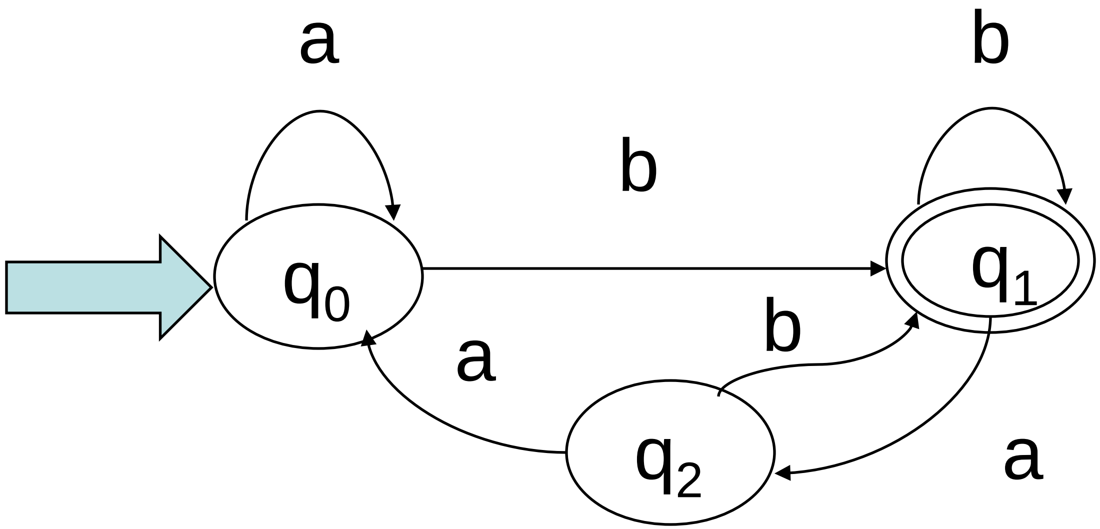


What about $q_1 \times a$?

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



a b b a a

q_0 q_0 q_1 q_1 q_2 q_0

a b b a b

q_0 q_0 q_1 q_1 q_2 q_1

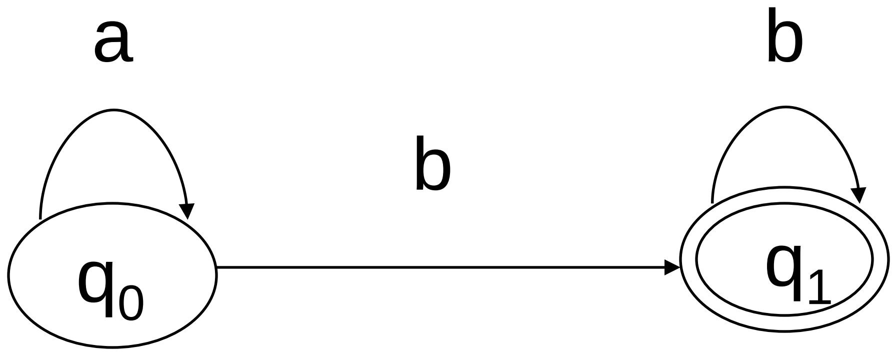
DFA as an acceptor

- 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, …}

Regular expression: $a^* \, b^+$

Regular grammar:

$q_0 \rightarrow a \, q_0$
$q_0 \rightarrow b \, q_1$
$q_1 \rightarrow b \, q_1$
$q_1 \rightarrow \epsilon$

NFA

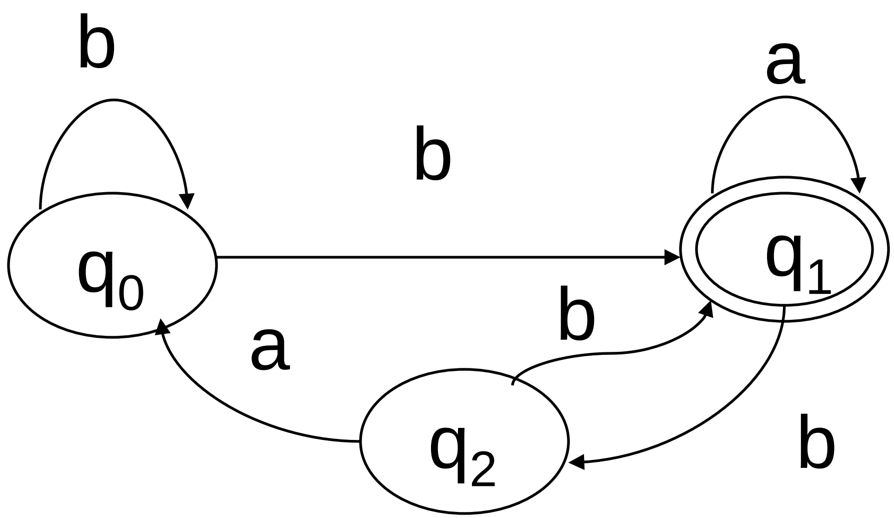
NFA

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

➔ now the transition function is:

$$S \times (\Sigma \bigcup \{\epsilon\}) \rightarrow 2^S$$

NFA example



b	b	a	b	b

q_0 q_0 q_1 q_1 q_2 q_1

q_0 q_1 q_2 q_0 q_0 q_0

b	b	a	b	b

q_0 q_1 q_2 q_0 q_0 q_1

q_0 q_1 q_2 q_0 q_1 q_2

Regular grammar and FSA

- Regular grammar: (N, Σ, P, S)
- FSA: $(\Sigma, Q, q_0, F, \delta)$
- Conversion between them ➔ Q1 in Hw2

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?

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

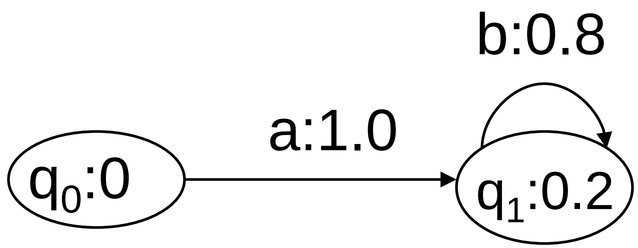
So far

- 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 ⇔ Regex ⇔ NFA ⇔ DFA ⇔ 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



$F(q_0)=0$
$F(q_1)=0.2$

$I(q_0)=1.0$
$I(q_1)=0.0$

$P(ab^n)=I(q_0)*P(q_0,ab^n,q_1)*F(q_1)$
$=1.0*(1.0*0.8^n)*0.2$

$$\sum_x P(x) = \sum_{n=0}^{\infty} P(ab^n) = 0.2*\sum_{n=0}^{\infty} 0.8^n = 0.2*\frac{0.8^0}{1-0.8} = 1$$

Formal definition of PFA

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

- Q: a finite set of N states
- Σ: a finite set of input symbols
- I: Q ➔R$^+$ (initial-state **probabilities**)
- F: Q ➔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:

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

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

Probability of a string:

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

$$P(w_{1,n}) = \sum_{q_{1,n+1}} P(w_{1,n}, q_{1,n+1})$$

PFA

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

- The probability of <u>a path </u>is the multiplication of the arcs on the path.

- The probability of <u>a string </u>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 <u>weight</u>.

- "Sum" and "Multiplication" can have other meanings.

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

- Ex: weight is –log prob
  - "multiplication" ➔ addition
  - "Sum" ➔ power

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, …

Additional slides

An algorithm for deterministic recognition of DFAs

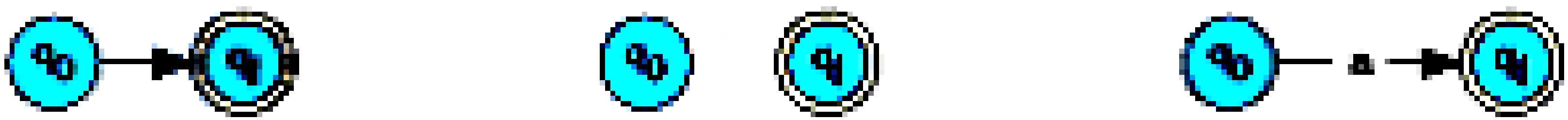
```
function D-RECOGNIZE(tape, machine) returns accept or reject
  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] 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 $\sum$ is a regular expression
  (2) ϵ is a regular expression
  (3) If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, so are
  $(\mathbf{r_1})$, $\mathbf{r_1}\ \mathbf{r_2}$, $\mathbf{r_1} \mid \mathbf{r_2}$, $\mathbf{r_1}^*$
  (4) Nothing else is a regular expression.

Regular expression ➔ NFA

Base case:



(a) r=ϵ (b) r=$\varnothing$ (c) r=a

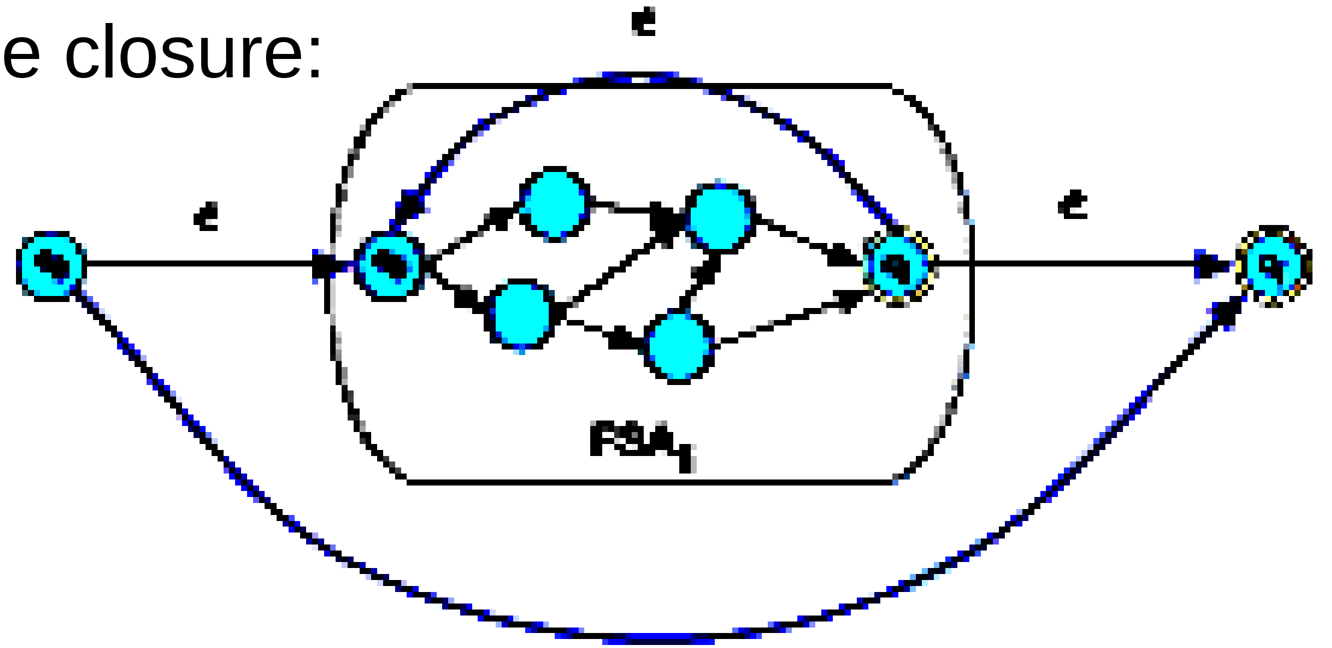
Concatenation: connecting the final states of FSA_1 to the initial state of FSA_2 by an ϵ-translation.

Union: Creating a new initial state and add ϵ-transitions from it to the initial states of FSA_1 and FSA_2.

Kleene closure:

Regular expression → NFA (cont)

Kleene closure:



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

Another PFA example: A bigram language model

P(BOS w_1 w_2 … w_n EOS)

= P(BOS) * P(w_1 | BOS) P(w_2 | w_1) * ….

P(w_n | w_{n-1}) * P(EOS | w_n)

Examples:

I bought two/to/too books

How many states?