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)

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

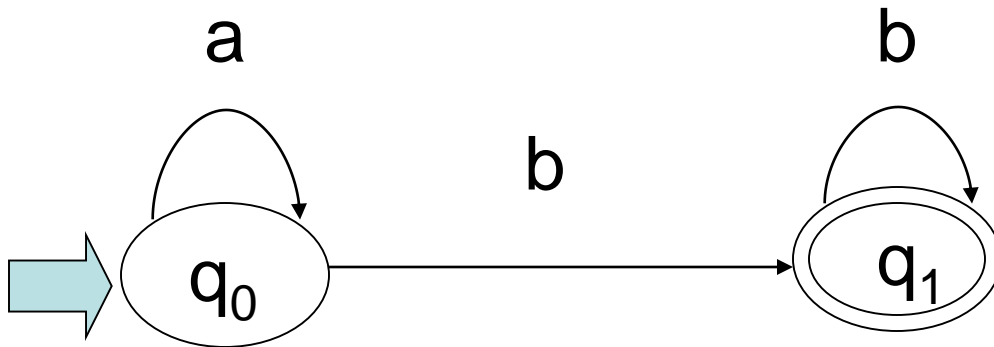- A set of final states F

- A transition function:

$$\delta : Q \times \Sigma \longrightarrow Q$$

Σ = {a, b}
S = {q_0, q_1}
F = {q_1}
δ = { $q_0 \times a \rightarrow q_0$,
     $q_0 \times b \rightarrow q_1$,
     $q_1 \times b \rightarrow q_1$ }

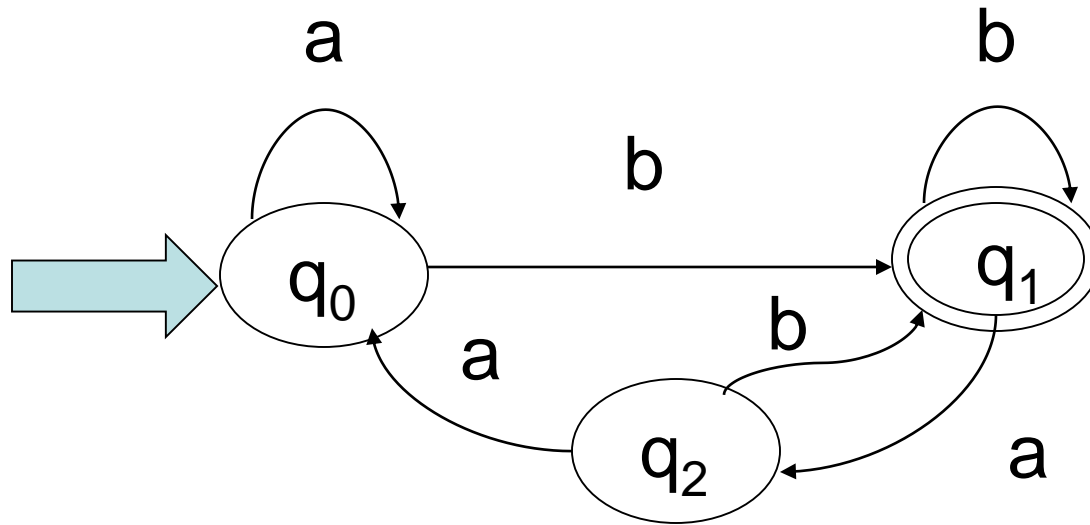a   b

b

q_0 ⟶ 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
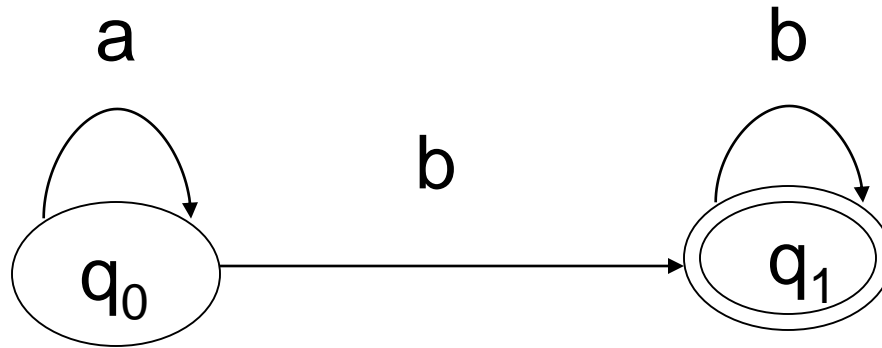
a	b	b	a	b

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

a   b

FSA:

b

q_0 q_1

Regular language: {b, ab, bb, aab, abb, …}

Regular expression: $a^* b^+$

Regular grammar: $q_0 \rightarrow a\ q_0$
                               $q_0 \rightarrow b\ q_1$
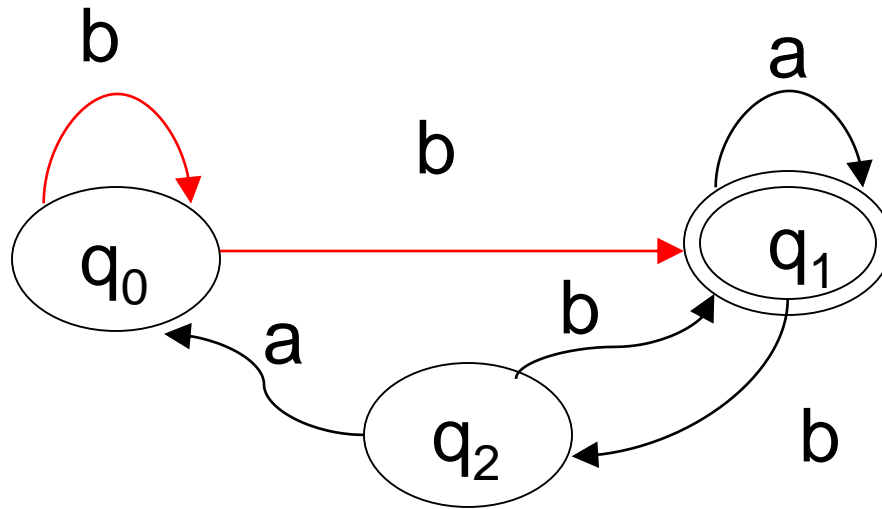 $q_1 \rightarrow b\ q_1$
                               $q_1 \rightarrow \epsilon$

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

14

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, Σ, 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 ⇔ 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

b:0.8

a:1.0

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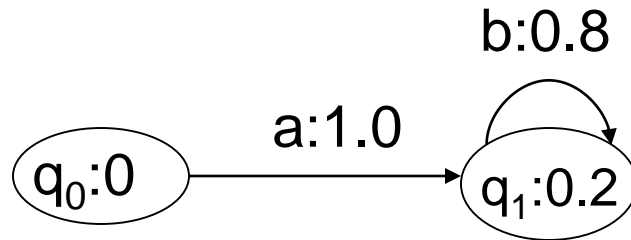
$F(q_0)=0$
$F(q_1)=0.2$

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

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

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

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)**
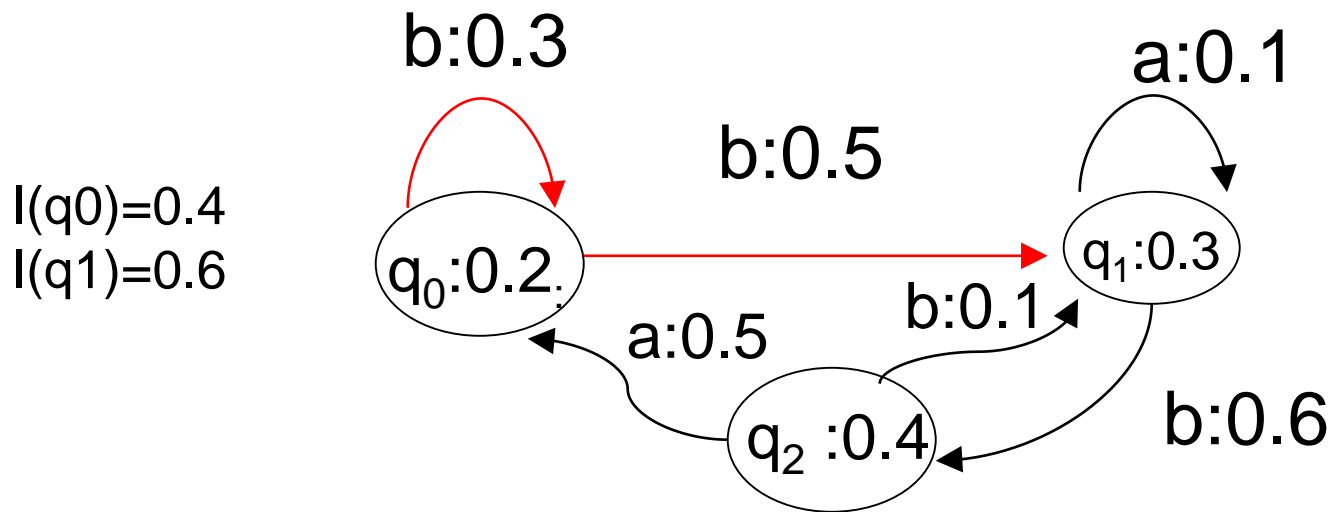
Constraints on function:

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

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

Probability of a string:

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

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

b:0.3

a:0.1

b:0.5

I(q0)=0.4
I(q1)=0.6

q_0:0.2

q_1:0.3

a:0.5

b:0.1

q_2 :0.4

b:0.6

Input: b b b

Prob(input, path)

q_0 q_0 q_0 q_0 I(q0)*P(q0, b, q0) * P(q0,b,q0)*P(q0,b,q0)*F(q0)

q_0 q_0 q_0 q_1 I(q0)*P(q0, b, q0) * P(q0,b,q0)*P(q0,b,q1)*F(q1)

q_1 q_2 q_1 q_2 I(q1)*P(q1, b, q2) * P(q2,b,q1)*P(q1,b,q2)*F(q2)

…

b:0.3

a:0.1

b:0.5

I(q0)=0.4
I(q1)=0.6

q_0:0.2

q_1:0.3

b:0.1

a:0.5

b:0.6

q_2 :0.4
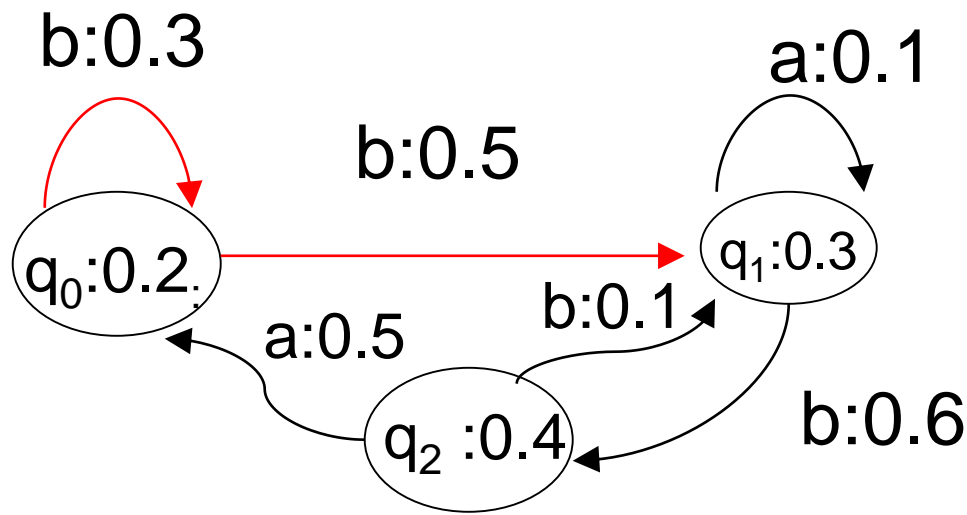
Input: | b | b | b |

Prob(input, path)

q_0 q_0 q_0 q_0 I(q0)*P(q0, b, q0) * P(q0,b,q0) *P(q0,b,q0)*F(q0)

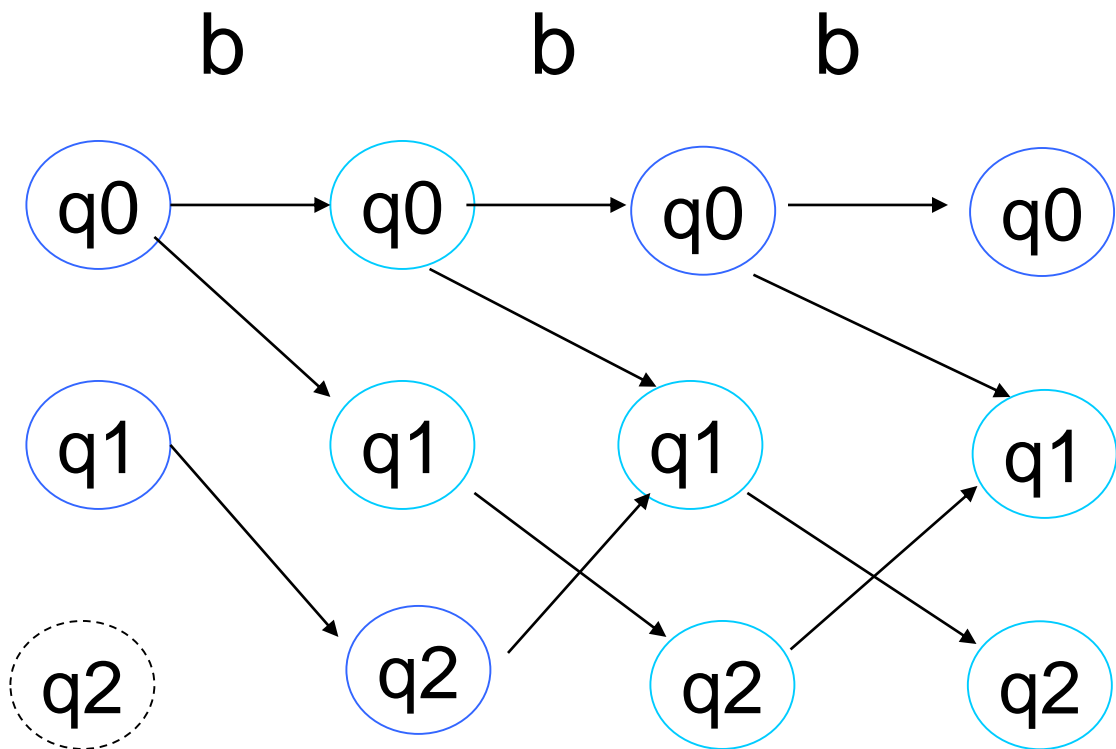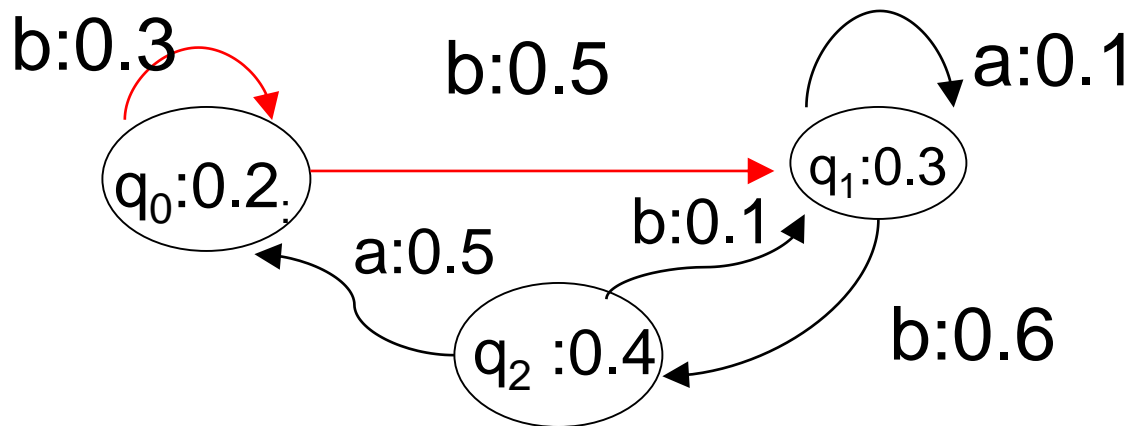q_0 q_0 q_0 q_1 I(q0)*P(q0, b, q0) * P(q0,b,q0) *P(q0,b,q1)*F(q1)

q_1 q_2 q_1 q_2 I(q1)*P(q1, b, q2) * P(q2,b,q1)*P(q1,b,q2)*F(q2)

…

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
  - …

I(q0)=0.4
I(q1)=0.6

b:0.3

b:0.5

a:0.1

q_0:0.2

q_1:0.3

a:0.5

b:0.1

b:0.6

q_2 :0.4

b   b   b

q0 → q0 → q0 → q0

q1   q1   q1   q1

q2   q2   q2   q2

Finding the best path for input x

- Read Section 3.2 of the 2005 PFA paper.

$$\bar{\theta} = \underset{\theta \in \Theta_A(x)}{\arg\max}\ \mathrm{Pr}_A(\theta).$$

The computation of $\mathrm{Pr}_A(x)$ can be efficiently performed by defining a function $\gamma_x(i, q)\ \forall q \in Q,\ 0 \leq i \leq |x|$, as the probability of generating the prefix $x_1 \ldots x_i$ through the best path and reaching state q:

$$\gamma_x(i, q) = \max_{(s_0, s_1, \ldots, s_i) \in \Theta_A(x_1 \ldots x_i)} I(s_0) \cdot \prod_{j=1}^{i} P(s_{j-1}, x_j, s_j) \cdot 1(q, s_i)$$

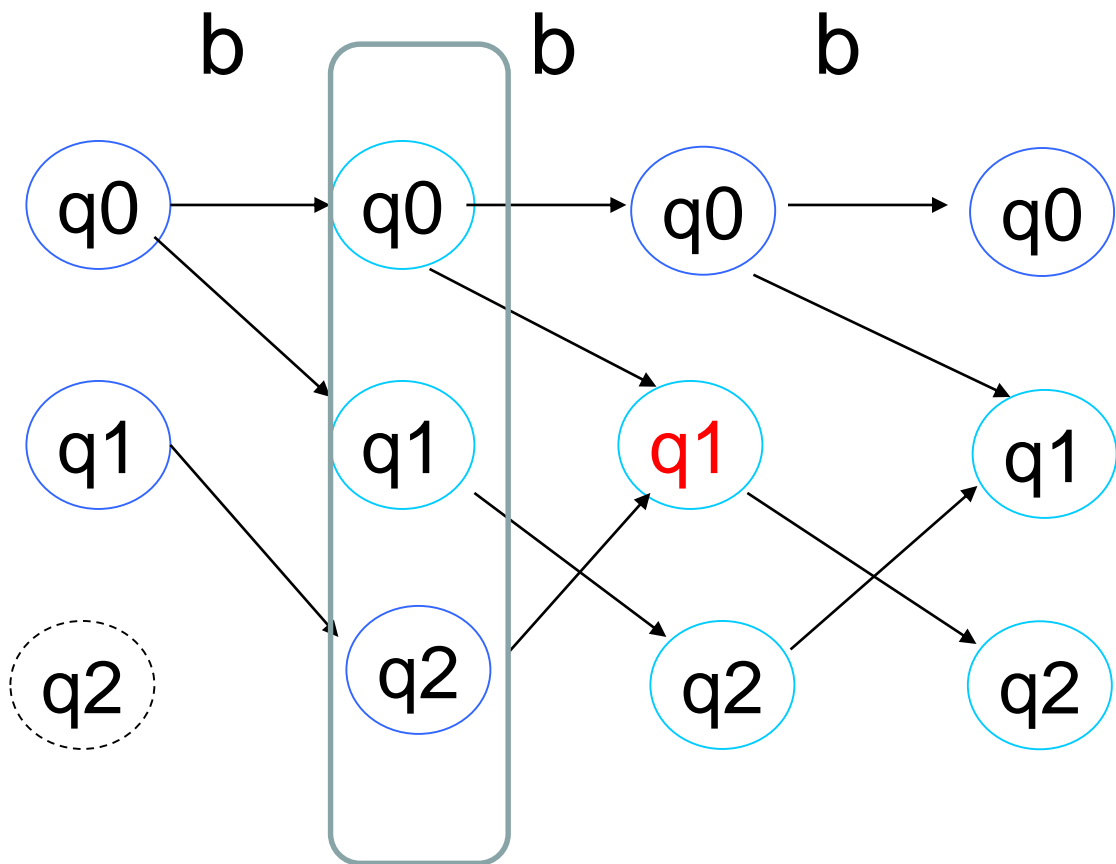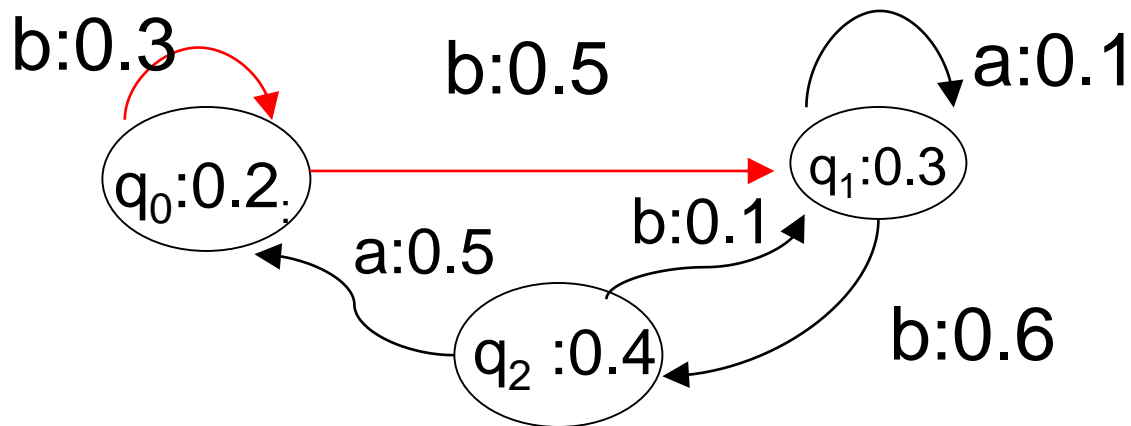where $1(q, q') = 1$ if $q = q'$ and 0 if $q \neq q'$.

Viterbi algorithm:

$$\gamma_x(0, q) = I(q),$$

$$\gamma_x(i, q) = \max_{q' \in Q} \gamma_x(i - 1, q') \cdot P(q', x_i, q), \quad 1 \leq i \leq |x|.$$

Remember to include the final-prob:

$$\widetilde{\mathrm{Pr}}_A(x) = \max_{q \in Q} \gamma_x(|x|, q) \cdot F(q)$$

b:0.3   b:0.5   a:0.1

I(q0)=0.4
I(q1)=0.6

q_0:0.2 q_1:0.3

a:0.5   b:0.1

q_2 :0.4   b:0.6

b   b   b

q0   q0   q0   q0

q1   q1   q1   q1

q2   q2   q2   q2

gamma(2, q1)

- Calculate the gamma function efficiently:
  - Use a two-dimensional array g[i, q], not a recursive function

- Need to remember the best path, not just the highest probability:
  - For each [i, q], remember q'
  - In other words, you need one array for gamma, another for backpointer: b[i, q] = q'

- For hw3 Q3, you need to find the best path and the corresponding output sequence given input x and an FST:
  - Since you know the (q', x_i, q) arc on the best path, you can find the corresponding y_i for that arc.
  - For Hw3 Q3, assume that there is no epsilon-transition in the FST.

Outline

- Deterministic FSA (DFA)

- Non-deterministic FSA (NFA)

- Probabilistic FSA (PFA)

- Weighted FSA (WFA)

Weighted finite-state automata (WFA)

- Each arc is associated with a <u>weight</u>.

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

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

- In PFA:

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

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

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

Semiring

A semiring is a set R equipped with two binary operations + (i.e., $\oplus$) and · (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, ·) is a monoid with identity element 1:

    ❖ (a·b)·c = a·(b·c)

    ❖ 1·a = a·1 = a

(3) Multiplication left and right distributes over addition:

    ❖   a·(b + c) = (a·b) + (a·c)

    ❖ (a + b)·c = (a·c) + (b·c)

(4) Multiplication by 0 annihilates R:

    ❖   0·a = a·0 = 0

Examples of semirings

Set R	$\oplus$	$\otimes$	0	1	Arc weight	weight (x) is
[0, 1]	+	x	0	1	prob	Prob of x
[0, 1]	??	??	??	??	prob	Prob of the best path for x
R ∪ {+∞, -∞}	min	+	??	??	distance	Shortest distance
R ∪ {+∞, -∞}	max	+	??	??	distance	Longest distance
N	+	x	0	1	??	Number of paths

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

x is the input string. Let's ignore I(s) and F(t).

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 r_1 and r_2 are regular expressions, so are

    $(r_1),$ $r_1\ r_2,$ $r_1\mid r_2,$ $r_1{}^*$

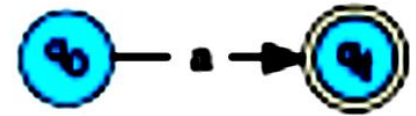   (4) Nothing else is a regular expression.

Regular expression ➔ NFA

Base case:

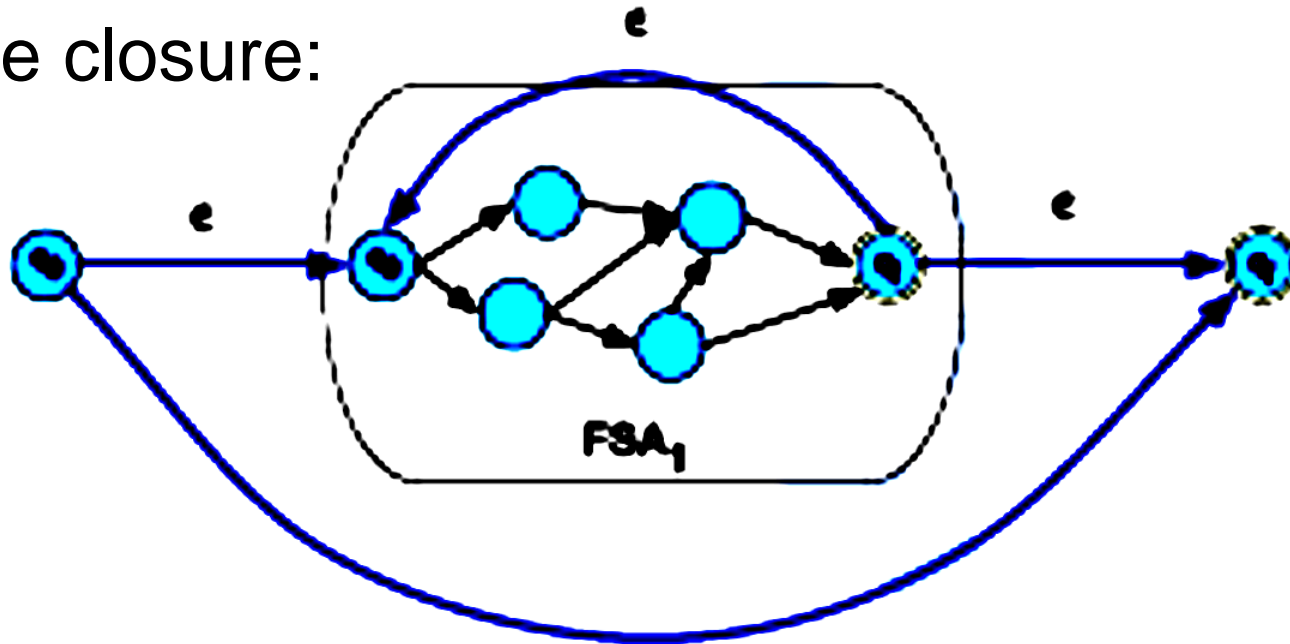**(a) r=ϵ** **(b) r=Ø** **(c) r=a**

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+)?