CS 4410

Automata, Computability, and Formal Language

Dr. Xuejun Liang

Spring 2019

Chapter 2

Finite Automata

1. Deterministic Finite Accepters
   - Deterministic Accepters and Transition Graphs
   - Languages and Dfas
   - Regular Language
2. Nondeterministic Finite Accepters
   - Definition of a Nondeterministic Accepter
   - Why Nondeterministic
3. Equivalence of Deterministic and Nondeterministic Finite Accepters

Learning Objectives

At the conclusion of the chapter, the student will be able to:

- Describe the components of a deterministic finite accepter (dfa)
- State whether an input string is accepted by a dfa
- Describe the language accepted by a dfa
- Construct a dfa to accept a specific language
- Show that a particular language is regular
- Describe the differences between deterministic and nondeterministic finite automata (nfa)
- State whether an input string is accepted by a nfa
- Construct a nfa to accept a specific language
- Transform an arbitrary nfa to an equivalent dfa

Deterministic Finite Accepters

Definition 2.1

A deterministic finite accepter or dfa is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where Q is a finite set of internal states,

Σ is a finite set of symbols called the input alphabet,

δ: $Q \times \Sigma \to Q$ is a total function called the transition function,

$q_0 \in Q$ is the initial state,

$F \subseteq Q$ is a set of final states.

Transition Graph of a dfa $M = (Q, \Sigma, \delta, q_0, F)$

Vertex labeled with q_i: state $q_i \in Q$,

Edge from q_i to q_j labeled with a: transition $\delta(q_i, a) = q_j$.

Example 2.1 $M = (\{q_0, q_1, q_3\}, \{0,1\}, \delta, q_0, \{q_1\})$, where δ is given by

$$\delta(q_0,0) = q_0, \delta(q_0,1) = q_1, \delta(q_1,0) = q_0,$$

$$\delta(q_1,1) = q_2, \delta(q_2,0) = q_2, \delta(q_2,1) = q_1$$

Languages and Dfas

The extended transition function $\delta^* : Q \times \Sigma^* \rightarrow Q$ can be recursively defined by

$$\delta^*(q, \lambda) = q$$

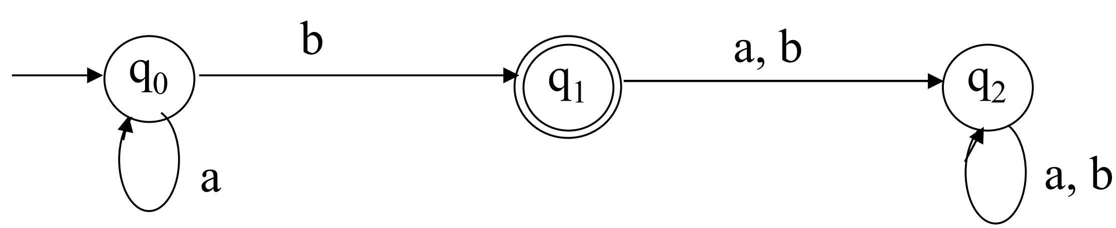
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

Definition 2.2

The language accepted by a dfa $M = (Q, \Sigma, \delta, q_0, F)$ is the set of all strings on Σ accepted by M. In formal notation

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}.$$

Example 2.2 M is given as below, L(M) = ?



Languages and Dfas

Theorem 2.1 Let M = (Q, Σ, δ, q_0, F) a dfa, and let G_M be its associated transition graph. Then $\delta^*(q_i, w) = q_j$ if and only if there is in G_M a walk with label w from q_i to q_j.

Example 2.3 Find a dfa that accepts all strings on Σ = {a, b} starting with the prefix ab.

Example 2.4 Find a dfa that accepts all strings on Σ = {0, 1}, except those containing the substring 001.

Regular Languages

Definition 2.3

A language L is called regular if and only if there exists some deterministic finite accepter M such that L = L(M).

Example 2.5

Show that the language $L=\{awa : w \in \{a, b\}^*\}$ is regular.

Example 2.6

Let $L=\{awa : w \in \{a, b\}^*\}$. Show that L^2 is regular.

Nondeterministic Finite Accepters

Definition 2.4

A nondeterministic finite accepter or nfa is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where Q, Σ, q_0, and F are as for deterministic finite accepter, but

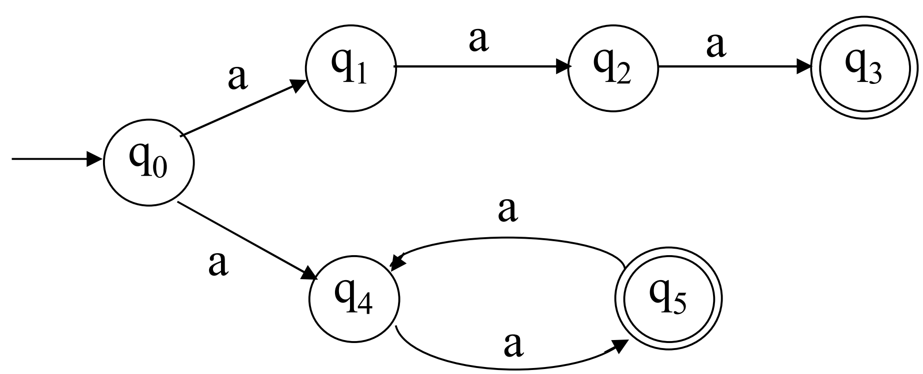
$$\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$$

Transition Graph of an nfa $M = (Q, \Sigma, \delta, q_0, F)$

Vertex labeled with q_i: state $q_i \in Q$,

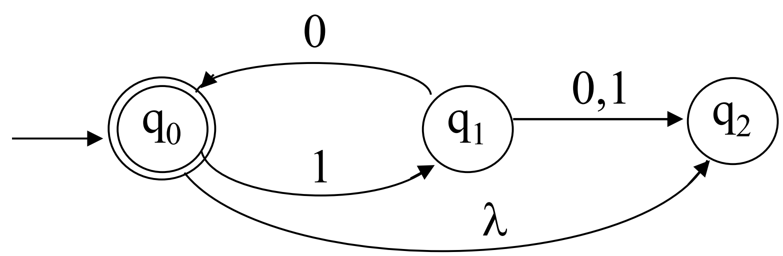
Edge from q_i to q_j labeled with a: $q_j \in \delta(q_i, a)$

Example 2.7 An nfa is shown as below



Nondeterministic Finite Accepters

Example 2.8 An nfa is shown as below



The extended transition function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ can be defined by

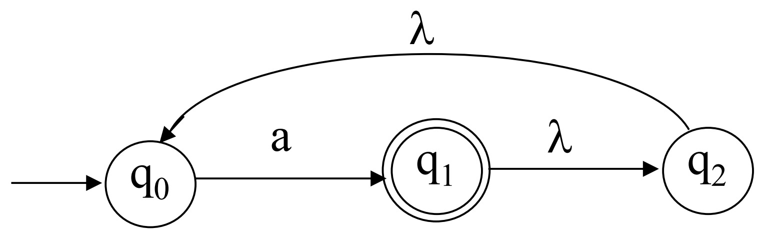
$$\delta^*(q, \lambda) = \{q\} \cup \delta(q, \lambda)$$

$$\delta^*(q, wa) = \cup\{\delta(p, a) : p \in \delta^*(q, w)\}$$

Definition 2.5 (This is a theorem if the above definition is used)
For an nfa, the extended transition function is defined so that $\delta^*(q_i,w)$ contains q_j if and only if there is a walk in the transition graph from q_i to q_j labeled w. This holds for all q_i, $q_j \in Q$ and $w \in \Sigma^*$.

Nondeterministic Finite Accepters

Example 2.9 Consider an nfa, we have



$$\delta^*(q_1, a) = \{q_0, q_1, q_2\}$$

$$\delta^*(q_2, \lambda) = \{q_0, q_2\}$$

$$\delta^*(q_2, aa) = \{q_0, q_1, q_2\}$$

Definition 2.6

The language accepted by an nfa M = (Q, Σ, δ, q_0, F) is the set of all strings on Σ accepted by M. In formal notation

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \varnothing\}.$$

Example 2.10 What is the language accepted by the nfa in Example 2.8

Why Nondeterminism?

Equivalence of Deterministic and Nondeterministic Finite Accepters

Definition 2.7

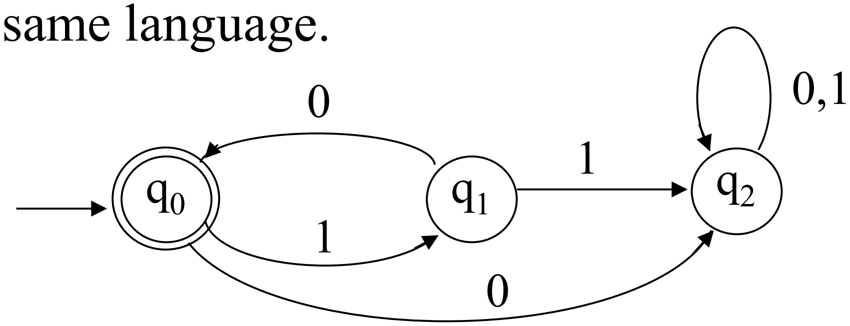
Two finite accepters M_1 and M_2 are said to be equivalent if

$$L(M_1)=L(M_2)$$

That is, if they accept the same language.

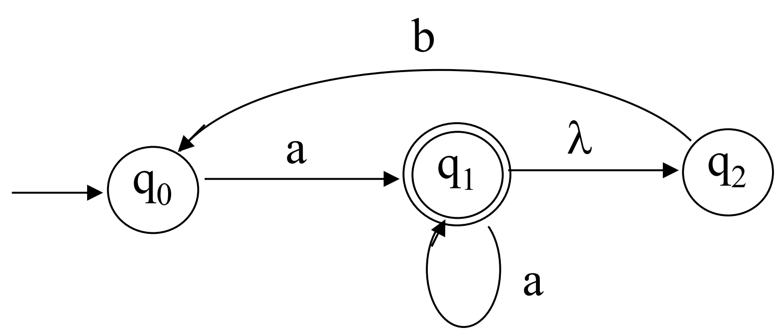
Example 2.11

The dfa is equivalent to the nfa in Example 2.8



Example 2.12

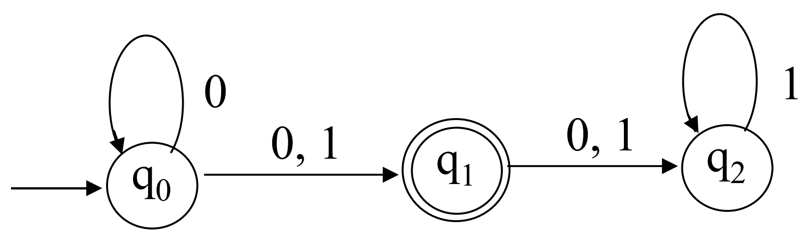
Convert the nfa to an equivalent dfa



Equivalence of Deterministic and Nondeterministic Finite Accepters

Theorem 2.2 Let L be the language accepted by an nfa $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a dfa $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L = L(M_D)$.

Example 2.13
Convert the nfa to an equivalent dfa



Property: Every language accepted by an nfa is regular

Procedure: Nfa_to_Dfa

1. Create a graph G_D with vertex $\{q_0\}$ as the initial vertex
2. Repeat until no more edges are missing
   a. Take any vertex $\{q_i, q_j, .., q_k\}$ of G_D that has no outgoing edge for some $a \in \Sigma$.
   b. Compute $\{q_l, q_m, \ldots, q_n\} = \delta^*(q_i, a) \cup \delta^*(q_j, a) \cup \ldots \cup \delta^*(q_k, a)$
   c. Create a vertex for G_D labeled $\{q_l, q_m, \ldots, q_n\}$ if it does not already exist.
   d. Add to G_D an edge from $\{q_i, q_j, .., q_k\}$ to $\{q_l, q_m, \ldots, q_n\}$ and label it with a
3. Every state of G_D whose label contains any $q_f \in F_N$ is identified as a final vertex.
4. If M_N accepts λ, the vertex $\{q_0\}$ in G_D is also made a final vertex.

The following slides from my compiler class

Conversion of an NFA into a DFA

- The ***subset construction*** algorithm converts an NFA into a DFA using:
  - λ-*closure*$(s) = \{s\} \cup \{\, t \mid s \rightarrow_\varepsilon \ldots \rightarrow_\varepsilon t\}$
  - λ-*closure*$(T) = \cup_{s \in T} \lambda$-*closure*$(s)$
  - *move*$(T, a) = \{\, s \mid t \rightarrow_a s$ and $t \in T\}$
- The algorithm produces:
  - ***Dstates*** -- the set of states of the new DFA consisting of sets of states of the NFA
  - ***Dtran*** -- the transition table of the new DFA

The Subset Construction Algorithm

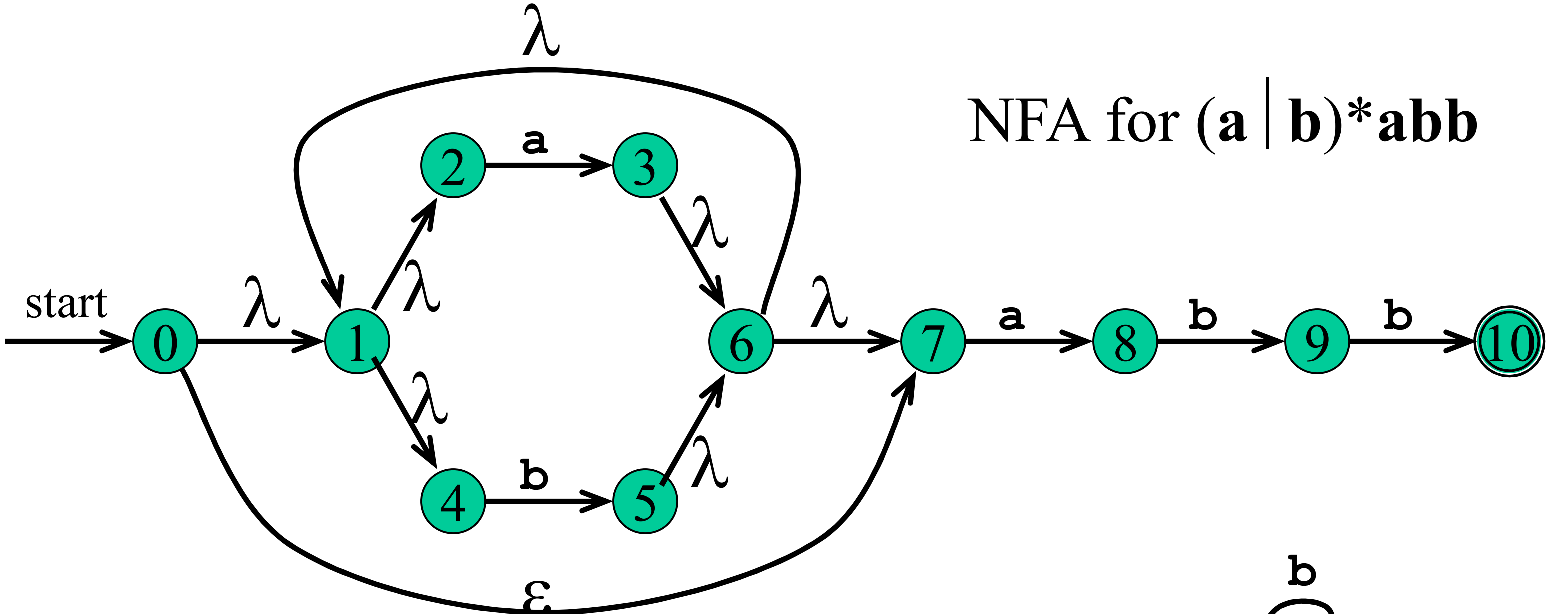
Initially, ε-*closure*(q_0) is the only state in *Dstates* and it is unmarked

while (there is an unmarked state *T* in *Dstates)* **{**
    mark *T*
    for (each input symbol $a \in \Sigma$) {
        U = λ-*closure*(*move*(*T,a*))
        if (*U* is not in *Dstates)*
            add *U* as an unmarked state to *Dstates*
        Dtran[*T*,*a*] := *U*
    }
}

Computing λ-closure(T)

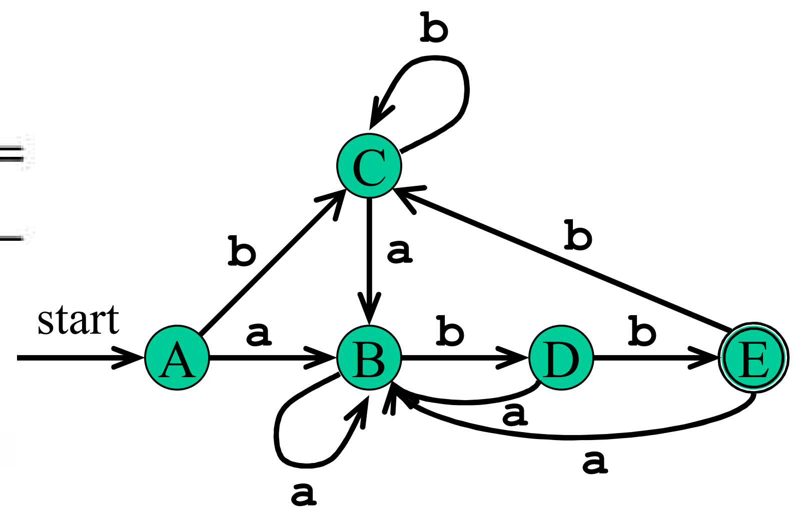
```
push all states of T onto stack;
initialize λ-closure(T) to T;
while ( stack is not empty ) {
        pop t, the top element, off stack;
        for ( each state u with an edge from t to u labeled λ)
              if ( u is not in λ-closure(T) ) {
                      add u to λ-closure(T) ;
                      push u onto stack;
              }
}
```

Subset Construction Example 1

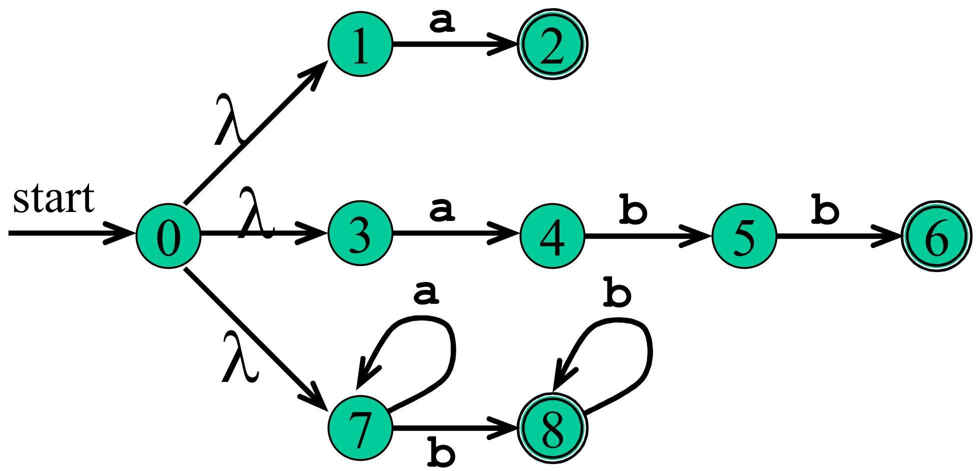


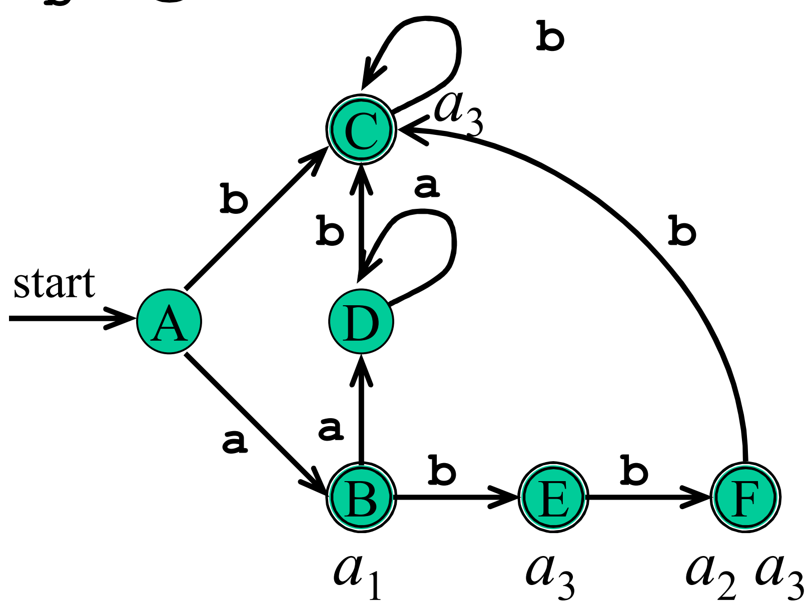
NFA for (**a** | **b**)***abb**

NFA State	DFA State	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 3, 5, 6, 7, 10\}$	E	B	C



Subset Construction Example 2





Dstates
A = {0,1,3,7}
B = {2,4,7}
C = {8}
D = {7}
E = {5,8}
F = {6,8}