

CSci 311, Models of Computation  
Chapter 2  
Finite Automata

H. Conrad Cunningham

29 December 2015

**Contents**

Introduction . . . . .	2
2.1 Deterministic Finite Accepters . . . . .	2
2.1.1 Accepters . . . . .	2
2.1.2 Transition Graphs . . . . .	3
2.1.3 Linz Example 2.1 . . . . .	3
2.1.4 Extended Transition Function for a DFA . . . . .	4
2.1.5 Language Accepted by a DFA . . . . .	4
2.1.6 Linz Example 2.2 . . . . .	4
2.1.7 Linz Example 2.3 . . . . .	5
2.1.8 Linz Example 2.4 . . . . .	5
2.1.9 Regular Languages . . . . .	7
2.1.10 Linz Example 2.5 . . . . .	7
2.1.11 Linz Example 2.6 . . . . .	8
2.2 Nondeterministic Finite Accepters . . . . .	8
2.2.1 Nondeterministic Accepters . . . . .	8
2.2.2 Linz Example 2.7 . . . . .	9
2.2.3 Linz Example 2.8 . . . . .	9
2.2.4 Extended Transition Function for an NFA . . . . .	11

2.2.5	Language Accepted by an NFA . . . . .	11
2.2.6	Linz Example 2.10 (Example 2.8 Revisited) . . . . .	11
2.2.7	Why Nondeterminism . . . . .	12
2.3	Equivalence of DFAs and NFAs . . . . .	12
2.3.1	Meaning of Equivalence . . . . .	12
2.3.2	Linz Example 2.11 . . . . .	13
2.3.3	Power of NFA versus DFA . . . . .	13
2.3.4	Linz Example 2.12 . . . . .	15
2.3.5	Linz Example 2.13 . . . . .	18
2.4	Reduction in the Number of States in Finite Automata . . . . .	19

Copyright (C) 2015, H. Conrad Cunningham

**Acknowledgements:** These lecture notes are for use with Chapter 2 of the textbook: Peter Linz. *Introduction to Formal Languages and Automata*, Fifth Edition, Jones and Bartlett Learning, 2012. The terminology and notation used in these notes are similar to those used in the Linz textbook. This document uses several figures from the Linz textbook.

**Advisory:** The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of December 2015 seems to be a recent version of Firefox from Mozilla.

## Introduction

In chapter 2, we approach automata and languages more precisely and formally than in chapter 1.

A *finite automaton* is an automaton that has no temporary storage (and, like all automata we consider in this course, a finite number of states and input alphabet).

## 2.1 Deterministic Finite Accepters

### 2.1.1 Accepters

**Linz Definition (DFA):** A *deterministic finite accepter*, or *dfa*, is defined by the tuple  $M = (Q, \Sigma, \delta, q_0, F)$  where

- $Q$  is a finite set of *internal states*
- $\Sigma$  is a finite set of symbols called the *input alphabet*

- $\delta : Q \times \Sigma \rightarrow Q$  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*.

**A dfa operates as described in the following pseudocode:** currentState

:=  $q_0$

position input at left end of string

**while more input exists** currentInput := next\_input\_symbol

advance input to right

currentState :=  $\delta(\text{currentState}, \text{currentInput})$

if currentState  $\in F$  then ACCEPT else REJECT

### 2.1.2 Transition Graphs

To visualize a dfa, we use a *transition graph* constructed as follows:

- Vertices represent states
  - labels are state names
  - exactly one vertex for every  $q_i \in Q$
- Directed edges represent transitions
  - label on edge is current input symbol
  - directed edge  $(q, r)$  with label  $a$  if and only if  $\delta(q, a) = r$

### 2.1.3 Linz Example 2.1

The graph pictured in Linz Figure 2.1 represents the dfa  $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$ , where  $\delta$  is represented by

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

$$\delta(q_1, 0) = q_0, \delta(q_1, 1) = q_2$$

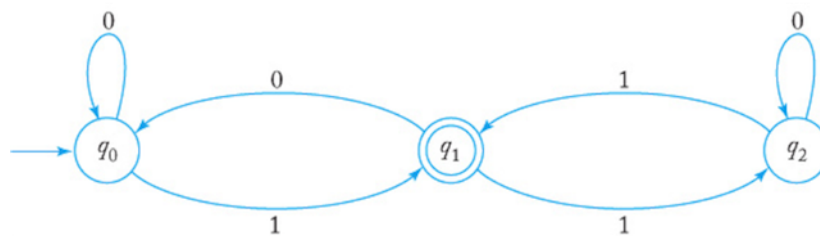
$$\delta(q_2, 0) = q_2, \delta(q_2, 1) = q_1$$

Note that  $q_0$  is the *initial state* and  $q_1$  is the only *final state* in this dfa.

The dfa in Linz Figure 2.1:

- Accepts 01, 101
- Rejects 00, 100

What about 0111? 1100?



**Linz Fig. 2.1: DFA Transition Graph**

### 2.1.4 Extended Transition Function for a DFA

**Linz Definition:** The *extended transition function*  $\delta^* : Q \times \Sigma^* \rightarrow Q$  is defined recursively:

$$\begin{aligned} \delta^*(q, \lambda) &= q \\ \delta^*(q, wa) &= \delta(\delta^*(q, w), a) \end{aligned}$$

The extended transition function gives the state of the automaton after reading a string.

### 2.1.5 Language Accepted by a DFA

**Linz Definition 2.2 (Language Accepted by DFA):** The language accepted by a dfa  $M = (Q, \Sigma, \delta, q_0, F)$  is  $L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$ .

That is,  $L(M)$  is the set of all strings on the input alphabet accepted by automaton  $M$ .

Note that above  $\delta$  and  $\delta^*$  are total functions (i.e., defined for all strings).

### 2.1.6 Linz Example 2.2

The automaton in Linz Figure 2.2 accepts all strings consisting of arbitrary numbers of  $a$ 's followed by a single  $b$ .

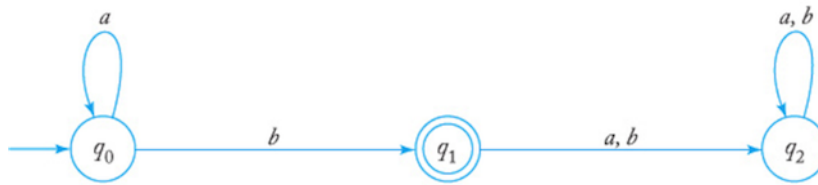
In set notation, the language accepted by the automaton is  $L = \{a^n b : n \geq 0\}$ .

Note that  $q_2$  has two self-loop edges, each with a different label. We write this compactly with multiple labels.

A *trap state* is a state from which the automaton can never “escape”.

Note that  $q_2$  is a trap state in the dfa transition graph shown in Linz Figure 2.2.

Transition graphs are quite convenient for understanding finite automata.



**Linz Fig. 2.2: DFA Transition Graph with Trap State**

For other purposes—such as representing finite automata in programs—a tabular representation of transition function  $\delta$  may also be convenient (as shown in Linz Fig. 2.3).

	<i>a</i>	<i>b</i>
<i>q</i> <sub>0</sub>	<i>q</i> <sub>0</sub>	<i>q</i> <sub>1</sub>
<i>q</i> <sub>1</sub>	<i>q</i> <sub>2</sub>	<i>q</i> <sub>2</sub>
<i>q</i> <sub>2</sub>	<i>q</i> <sub>2</sub>	<i>q</i> <sub>2</sub>

**Linz Fig. 2.3: DFA Transition Table**

### 2.1.7 Linz Example 2.3

Find a deterministic finite accepter that recognizes the set of all string on  $\Sigma = \{a, b\}$  starting with the prefix *ab*.

Linz Figure 2.4 shows a transition graph for a dfa for this example.

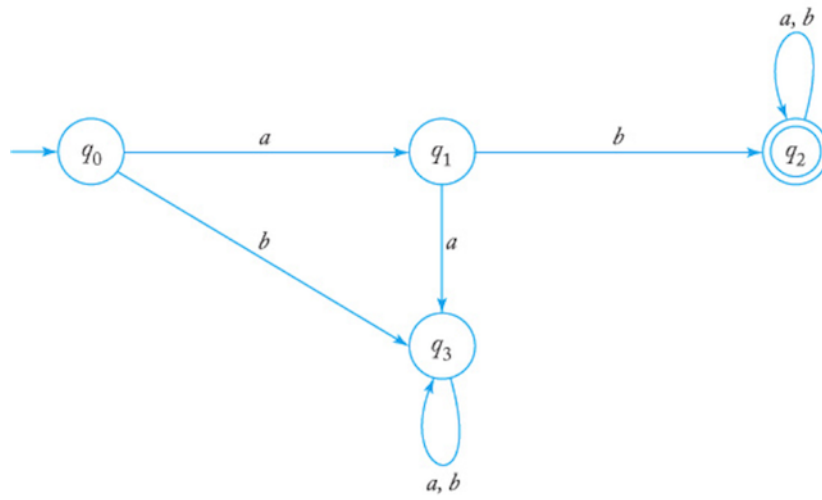
The dfa must accept *ab* and then continue until the string ends.

This dfa has a *final* trap state *q*<sub>2</sub> (accepts) and a *non-final* trap state *q*<sub>3</sub> (rejects).

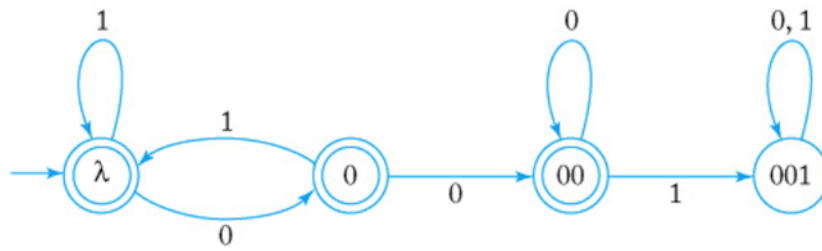
### 2.1.8 Linz Example 2.4

Find a dfa that accepts all strings on  $\{0, 1\}$ , except those containing the substring 001.

- need to “remember” whether last two inputs were 00
- use state changes for “memory”



Linz Fig. 2.4



Linz Fig. 2.5

Linz Figure 2.5 shows a dfa for this example.

Accepts:  $\lambda$ , 0, 00, 01, 010000

Rejects: 001, 000001, 0010101010101

### 2.1.9 Regular Languages

**Linz Definition 2.3 (Regular Language):** A language  $L$  is called *regular* if and only if there exists a dfa  $M$  such that  $L = L(M)$ .

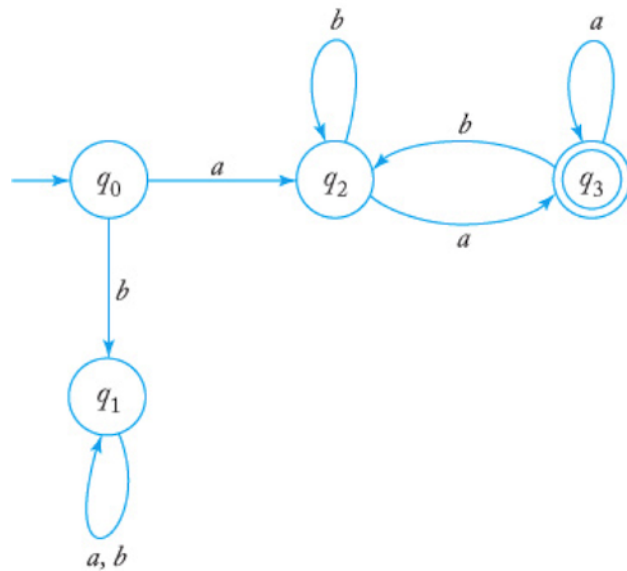
Thus dfas define the *family* of languages called *regular*.

### 2.1.10 Linz Example 2.5

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

- Construct a dfa.
- Check whether begin/end with “a”.
- Am in final state when second  $a$  input.

Linz Figure 2.6 shows a dfa for this example.



**Linz Fig. 2.6**

Question: How would we prove that a languages is not regular?

We will come back to this question in chapter 4.

### 2.1.11 Linz Example 2.6

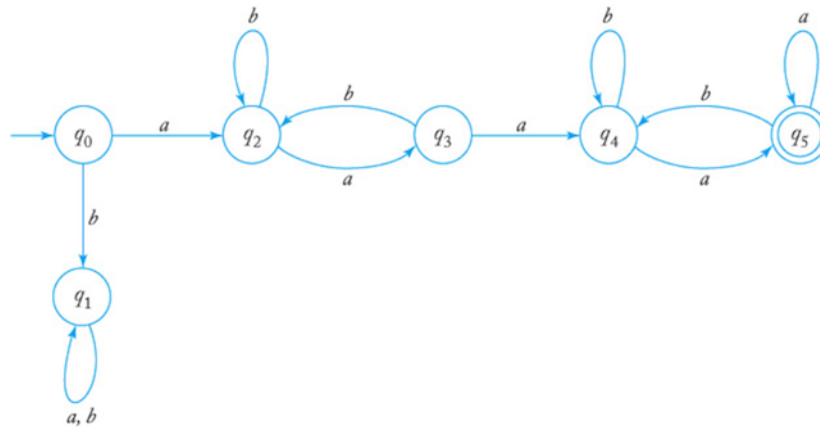
Let  $L$  be the language in the previous example (Linz Example 2.5).

Show that  $L^2$  is regular.

$$L^2 = \{aw_1aaw_2a : w_1, w_2 \in \{a, b\}^*\}.$$

- Construct a dfa.
- Use Example 2.5 dfa as starting point.
- Accept two consecutive strings of form  $awa$ .
- Note that any two consecutive  $a$ 's could start a second string.

Linz Figure 2.7 shows a dfa for this example.



Linz Fig. 2.7

The last example suggests the conjecture that if a language  $L$  then so is  $L^2$ ,  $L^3$ , etc.

We will come back to this issue in chapter 4.

## 2.2 Nondeterministic Finite Acceptors

### 2.2.1 Nondeterministic Acceptors

**Linz Definition 2.4 (NFA):** A *nondeterministic finite acceptor* or *nfa* is defined by the tuple  $M = (Q, \Sigma, \delta, q_0, F)$  where  $Q$ ,  $\Sigma$ ,  $q_0$ , and  $F$  are defined as for deterministic finite acceptors, but



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

Remember for dfas:

- $Q$  is a finite set of *internal states*.
- $\Sigma$  is a finite set of symbols called the *input alphabet*.
- $q_0 \in Q$  is the *initial state*.
- $F \subseteq Q$  is a set of *final states*.

The key differences between dfas and nfas are

1. dfa:  $\delta$  yields a single state  
nfa:  $\delta$  yields a set of states
2. dfa: consumes input on each move  
nfa: can move without input ( $\lambda$ )
3. dfa: moves for all inputs in all states  
nfa: some situations have no defined moves

An nfa *accepts* a string if *some possible* sequence of moves ends in a final state.

An nfa *rejects* a string if *no possible* sequence of moves ends in a final state.

### 2.2.2 Linz Example 2.7

Consider the transition graph shown in Linz Figure 2.8.

Note the nondeterminism in state  $q_0$  with two possible transitions for input  $a$ .

Also state  $q_3$  has no transition for any input.

### 2.2.3 Linz Example 2.8

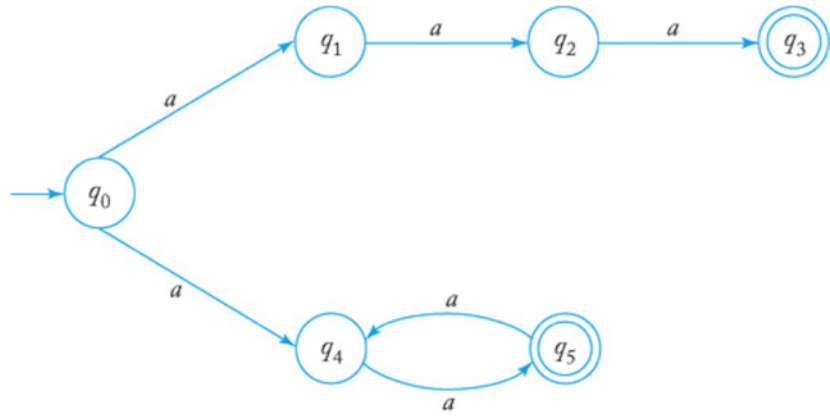
Consider the transition graph for an nfa shown in Linz Figure 2.9.

Note the nondeterminism and the  $\lambda$ -transition.

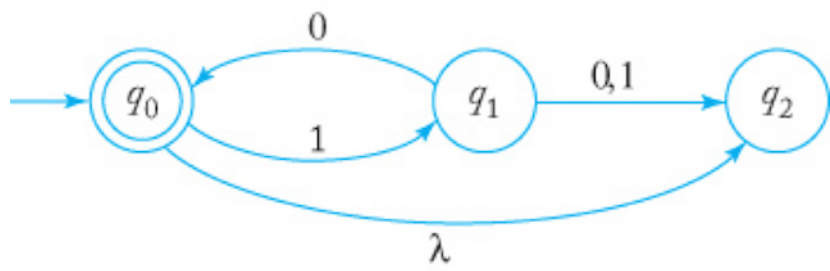
Note: Here  $\lambda$  means the move takes place without consuming any input symbol. This is different from accepting an empty string.

Transitions:

- for  $(q_0, 0)$ ?



Linz Fig. 2.8



Linz Fig. 2.9

- for  $(q_1, 0)$ ?
- for  $(q_2, 0)$ ?
- for  $(q_2, 1)$ ?

Accepts:  $\lambda$ , 10, 1010, 101010

Rejects: 0, 11

### 2.2.4 Extended Transition Function for an NFA

As with dfas, the transition function can be *extended* so its second argument is a string.

Requirement:  $\delta^*(q_i, w) = Q_j$  where  $Q_j$  is the set of all possible states the automaton may be in, having started in state  $q_i$  and read string  $w$ .

**Linz Definition (Extended Transition Function):** For an nfa, the *extended transition function* is defined so that  $\delta^*(q_i, w)$  contains  $q_j$  if there is a walk in the transition graph from  $q_i$  to  $q_j$  labelled  $w$ .

### 2.2.5 Language Accepted by an NFA

**Linz Definition 2.6 (Language Accepted by NFA):** The language  $L$  accepted by the nfa  $M = (Q, \Sigma, \delta, q_0, F)$  is defined

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

That is,  $L(M)$  is the set of all strings  $w$  for which there is a walk labeled  $w$  from the initial vertex of the transition graph to some final vertex.

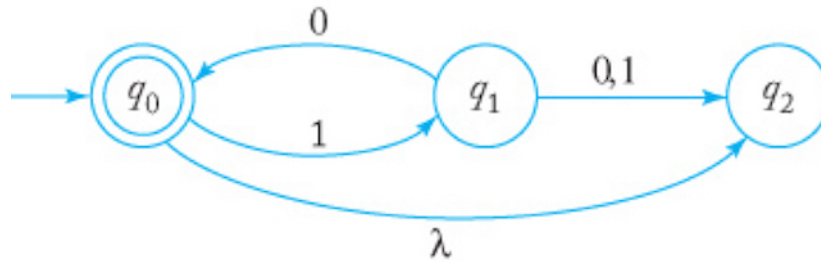
### 2.2.6 Linz Example 2.10 (Example 2.8 Revisited)

Let's again examine the automaton given in Linz Figure 2.9 (Example 2.8).

This nfa, call it  $M$ :

- must end in  $q_0$
- $L(M) = \{(10)^n : n \geq 0\}$

Note that  $q_2$  is a *dead configuration* because  $\delta^*(q_0, 110) = \emptyset$ .



Linz Fig. 2.9 (Repeated)

### 2.2.7 Why Nondeterminism

When computers are deterministic?

- an nfa can model a search or backtracking algorithm
- nfa solutions may be simpler than dfa solutions (can convert from nfa to dfa)
- nondeterminism may model externally influenced interactions (or abstract more detailed computations)

## 2.3 Equivalence of DFAs and NFAs

### 2.3.1 Meaning of Equivalence

When are two mechanisms (e.g., programs) equivalent?

- When they have exactly the same descriptions?
- When they always go through the exact same sequence of steps?
- When the same input generates the same output for both?

The last seems to be the best approach.

**Linz Definition 2.7 (DFA-NFA Equivalence):** Two finite accepters  $M_1$  and  $M_2$  are said to be *equivalent* if  $L(M_1) = L(M_2)$ . That is, if they both accept the same language.

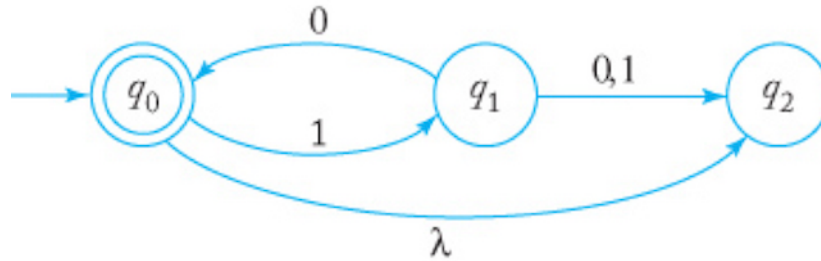
Here “same language” refers to the *input* and “both accept” refers to the *output*.

Often there are many accepters for a language.

Thus there are many equivalent dfa and nfas.

### 2.3.2 Linz Example 2.11

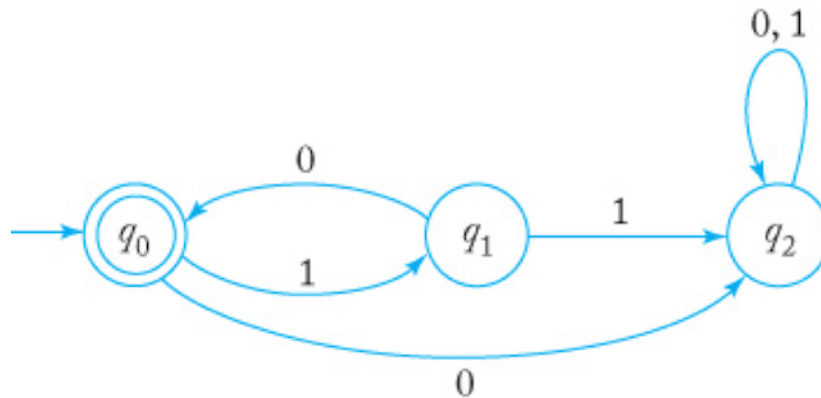
Again consider the nfa represented by the graph in Linz Fig. 2.9. Call this  $M_1$ .



Linz Fig. 2.9 (Repeated): An NFA

As we saw,  $L(M_1) = \{(10)^n : n \geq 0\}$ .

Now consider the dfa represented by the graph in Linz Figure 2.11. Call this  $M_2$ .



Linz Fig. 2.11: DFA Equivalent to Fig. 9 NFA

$L(M_2) = \{(10)^n : n \geq 0\}$ .

Thus,  $M_1$  is equivalent to  $M_2$ .

### 2.3.3 Power of NFA versus DFA

Which is more powerful, dfa or nfa?

Clearly, any dfa  $D$  can be made into a nfa  $N$ .

- Keep the same states.

- Define  $\delta_N(q, a) = \{\delta_D(q, a)\}$ .

Can any nfa  $N$  be made into a dfa  $D$ ?

- Yes, but it is less obvious. (See theorem below.)

Thus, dfas and nfes are “equally powerful”.

**Linz Theorem 2.2 (Existence of DFA Equivalent to NFA):** Let  $L$  be the language accepted by the 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  $L = L(M_D)$ .

A pure mathematician would be content with an existence proof.

But, as computing scientists, we want an *algorithm* for construction of  $M_D$  from  $M_N$ . The proof of the theorem follows from the correctness of the following algorithm.

Key ideas:

- After reading  $w$ ,  $M_N$  will be in the some state from  $\{q_i, q_j, \dots q_k\}$ . That is,  $\delta^*(q_0, w) = \{q_i, q_j, \dots q_k\}$ .
- Label the *dfa state* that has accepted  $w$  with the set of nfa states  $\{q_i, q_j, \dots q_k\}$ . This is an interesting “trick”!

Remember from the earlier definitions in these notes and from discrete mathematics:

- $\Sigma$  is finite (and the same for the nfa and dfa).
- $Q_N$  is finite.
- $\delta_D$  is a total function. That is, every vertex of the dfa graph has  $|\Sigma|$  outgoing edges.
- The maximum number of dfa states with the above labeling is  $|2^{Q_N}| = 2^{|Q_N|}$ . Hence, finite.
- The maximum number of dfa edges is  $2^{|Q_N|}|\Sigma|$ . Hence, finite.

#### Procedure nfa\_to\_dfa

Given a transition graph  $G_N$  for nfa  $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ , construct a transition graph  $G_D$  for a dfa  $M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ . Label each vertex in  $G_D$  with a *subset* of the vertices in  $G_N$ .

1. Initialize graph  $G_D$  to have an initial vertex  $\{q_0\}$  where  $q_0$  is the initial state of  $G_N$ .

2. Repeat the following steps until no more edges are missing from  $G_D$ :
  - a. Take any vertex from  $G_D$  labeled  $\{q_i, q_j, \dots, q_k\}$  that has no outgoing edge for some  $a \in \Sigma$ .
  - b. For this vertex and input, compute  $\delta_N^*(q_i, a), \delta_N^*(q_j, a), \dots, \delta_N^*(q_k, a)$ . (Each of these is a set of states from  $Q_N$ .)
  - c. Let  $\{q_l, q_m, \dots, q_n\}$  be the union of all  $\delta_N^*$  sets formed in the previous step.
  - d. If vertex  $\{q_l, q_m, \dots, q_n\}$  constructed in the previous step (step 2c) is not already in  $G_D$ , then add it to  $G_D$ .
  - e. Add an edge to  $G_D$  from vertex  $\{q_i, q_j, \dots, q_k\}$  (vertex selected in step 2b) to vertex  $\{q_l, q_m, \dots, q_n\}$  (vertex possibly created in step 2d) and label the new edge with  $a$  (input selected in step 2b).
3. Make every vertex of  $G_D$  whose label contains any vertex  $q_f \in F_N$  a final vertex of  $G_D$ .
4. If  $M_N$  accepts  $\lambda$ , then vertex  $\{q_0\}$  in  $G_D$  is also a final vertex.

This, if the loop terminates, it constructs the dfa corresponding to the nfa.

Does the loop terminate?

- Each iteration of the loop adds one edge to  $G_D$ .
- There are a finite number of edges possible in  $G_D$ .
- Thus the loop must terminate.

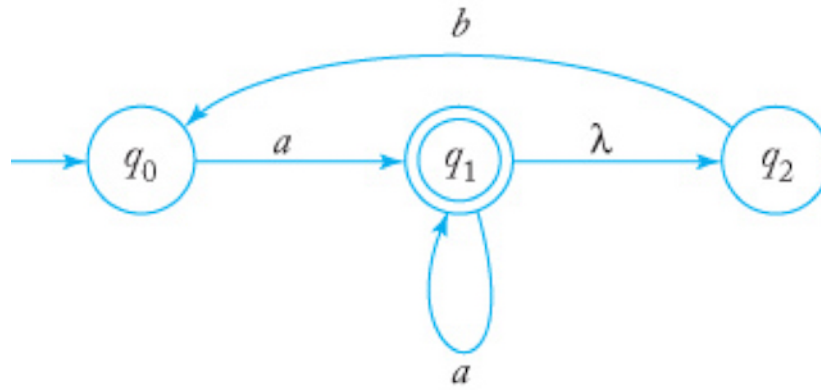
What is the loop invariant? (This is a property always that must hold at the loop test.)

- If there is a walk  $(\{q_0\}, \dots, \{\dots, q_i, \dots\})$  in  $G_D$  labeled  $w$ , then there is a walk  $(q_0, \dots, q_i)$  in  $G_N$  labeled  $w$ .

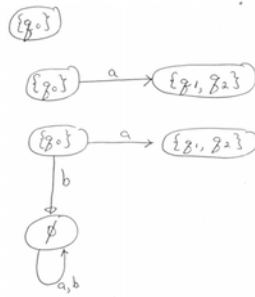
### 2.3.4 Linz Example 2.12

Convert the nfa in Linz Figure 2.12 to an equivalent dfa.

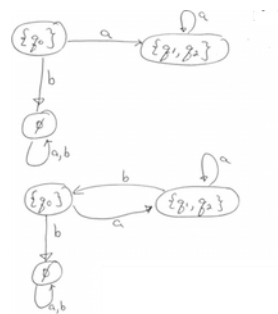
Intermediate steps are shown in Figures 2.12-1 and 2.12-2, with the final results in Linz Figure 2.13.



Linz Fig. 2.12: An NFA

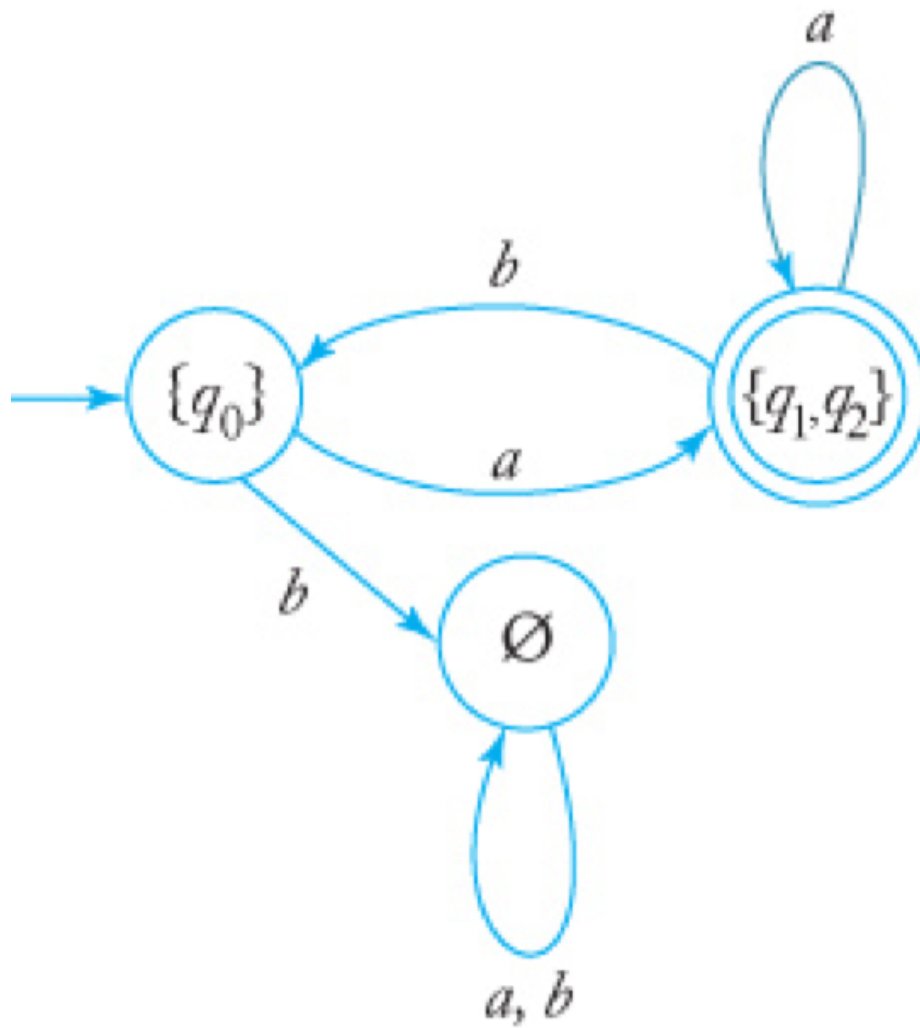


Intermediate Fig. 2.12-1



Intermediate Fig. 2.12-2





Linz Fig. 2.13: Corresponding DFA



Linz Fig. 2.14: An NFA

### 2.3.5 Linz Example 2.13

Convert the nfa shown in Linz Figure 2.14 into an equivalent dfa.

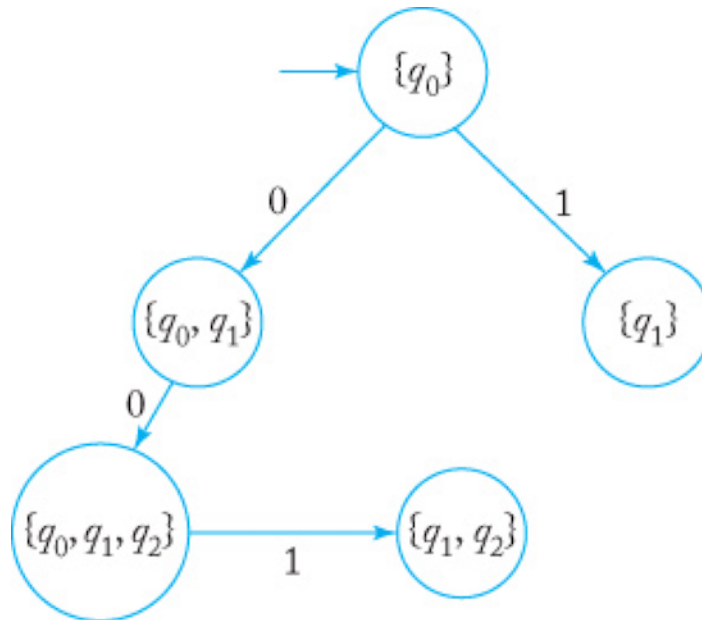
$$\delta_D(\{q_0\}, 0) = \delta_N^*(q_0, 0) = \{q_0, q_1\}$$

$$\delta_D(\{q_0\}, 1) = \delta_N^*(q_0, 1) = \{q_1\}$$

$$\delta_D(\{q_0, q_1\}, 0) = \delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) = \{q_0, q_1, q_2\}$$

$$\delta_D(\{q_0, q_1, q_2\}, 1) = \delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_1, q_2\}$$

The above gives us the partially constructed dfa shown in Linz Figure 2.15.



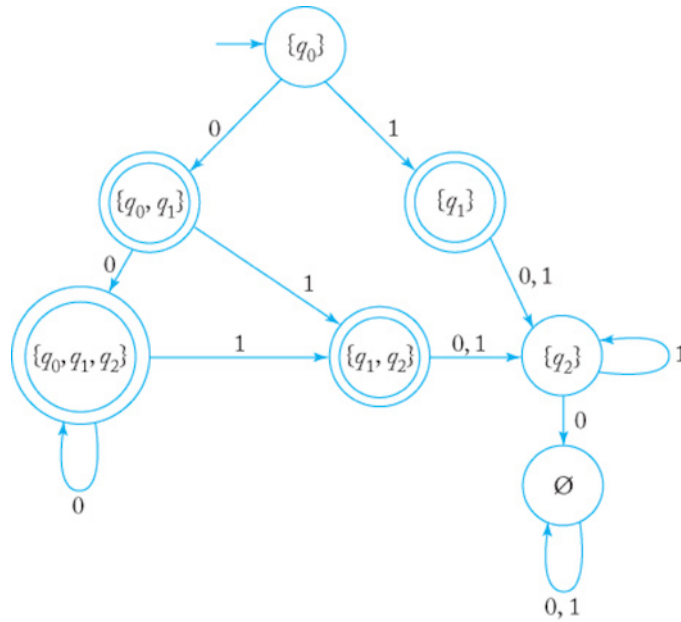
Linz Fig. 2.15

$$\delta_D(\{q_1\}, 0) = \delta_N^*(q_1, 0) = \{q_2\}$$

$$\delta_D(\{q_1\}, 1) = \delta_N^*(q_1, 1) = \{q_2\}$$

$$\begin{aligned} \delta_D(\{q_2\}, 0) &= \delta_N^*(q_1, 0) = \emptyset \\ \delta_D(\{q_2\}, 1) &= \delta_N^*(q_2, 1) = \{q_2\} \\ \delta_D(\{q_0, q_1\}, 1) &= \delta_N^*(q_0, 1) \cup \delta_N^*(q_1, 1) = \{q_1, q_2\} \\ \delta_D(\{q_0, q_1, q_2\}, 0) &= \delta_N^*(q_0, 0) \cup \delta_N^*(q_1, 0) \cup \delta_N^*(q_2, 0) = \{q_0, q_1, q_2\} \\ \delta_D(\{q_1, q_2\}, 0) &= \delta_N^*(q_1, 0) \cup \delta_N^*(q_2, 0) = \{q_2\} \\ \delta_D(\{q_1, q_2\}, 1) &= \delta_N^*(q_1, 1) \cup \delta_N^*(q_2, 1) = \{q_2\} \end{aligned}$$

Now, the above gives us the dfa shown in Linz Figure 2.16.



Linz Fig. 2.16: Corresponding DFA for NFA

## 2.4 Reduction in the Number of States in Finite Automata

This section is not covered in this course.