2.2 Deterministic Finite Automata

Now it is time to present the formal notion of a finite automaton, so that we may start to make precise some of the informal arguments and descriptions that we saw in Sections 1.1.1 and 2.1. We begin by introducing the formalism of a deterministic finite automaton, one that is in a single state after reading any sequence of inputs. The term "deterministic" refers to the fact that on each input there is one and only one state to which the automaton can transition from its current state. In contrast, "nondeterministic" finite automata, the subject of Section 2.3, can be in several states at once. The term "finite automaton" will refer to the deterministic variety, although we shall use "deterministic" or the abbreviation *DFA* normally, to remind the reader of which kind of automaton we are talking about.

2.2.1 Definition of a Deterministic Finite Automaton

A *deterministic finite automaton* consists of:

1. A finite set of *states*, often denoted Q.
2. A finite set of *input symbols*, often denoted Σ.
3. A *transition function* that takes as arguments a state and an input symbol and returns a state. The transition function will commonly be denoted δ. In our informal graph representation of automata, δ was represented by arcs between states and the labels on the arcs. If q is a state, and a is an input symbol, then $\delta(q, a)$ is that state p such that there is an arc labeled a from q to p.[2]
4. A *start state*, one of the states in Q.
5. A set of *final* or *accepting* states F. The set F is a subset of Q.

A deterministic finite automaton will often be referred to by its acronym: *DFA*. The most succinct representation of a DFA is a listing of the five components above. In proofs we often talk about a DFA in "five-tuple" notation:

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

where A is the name of the DFA, Q is its set of states, Σ its input symbols, δ its transition function, q_0 its start state, and F its set of accepting states.

2.2.3 Simpler Notations for DFA's

Specifying a DFA as a five-tuple with a detailed description of the δ transition function is both tedious and hard to read. There are two preferred notations for describing automata:

1. A *transition diagram*, which is a graph such as the ones we saw in Section 2.1.

2. A *transition table*, which is a tabular listing of the δ function, which by implication tells us the set of states and the input alphabet.

Transition Diagrams

A *transition diagram* for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows:

a) For each state in Q there is a node.

b) For each state q in Q and each input symbol a in Σ, let $\delta(q, a) = p$. Then the transition diagram has an arc from node q to node p, labeled a. If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc, labeled by the list of these symbols.

c) There is an arrow into the start state q_0, labeled *Start*. This arrow does not originate at any node.

d) Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

Example 2.2: Figure 2.4 shows the transition diagram for the DFA that we designed in Example 2.1. We see in that diagram the three nodes that correspond to the three states. There is a *Start* arrow entering the start state, q_0, and the one accepting state, q_1, is represented by a double circle. Out of each state is one arc labeled 0 and one arc labeled 1 (although the two arcs are combined into one with a double label in the case of q_1). The arcs each correspond to one of the δ facts developed in Example 2.1. □

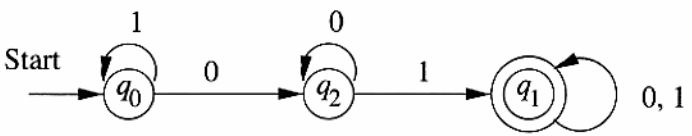


Figure 2.4: The transition diagram for the DFA accepting all strings with a substring 01

Transition Tables

A *transition table* is a conventional, tabular representation of a function like δ that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input a is the state $\delta(q, a)$.

Example 2.3: The transition table corresponding to the function δ of Example 2.1 is shown in Fig. 2.5. We have also shown two other features of a transition table. The start state is marked with an arrow, and the accepting states are marked with a star. Since we can deduce the sets of states and input symbols by looking at the row and column heads, we can now read from the transition table all the information we need to specify the finite automaton uniquely. □

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Figure 2.5: Transition table for the DFA of Example 2.1

2.2.2 How a DFA Processes Strings

The first thing we need to understand about a DFA is how the DFA decides whether or not to "accept" a sequence of input symbols. The "language" of the DFA is the set of all strings that the DFA accepts. Suppose $a_1 a_2 \cdots a_n$ is a sequence of input symbols. We start out with the DFA in its start state, q_0. We consult the transition function δ, say $\delta(q_0, a_1) = q_1$ to find the state that the DFA A enters after processing the first input symbol a_1. We process the next input symbol, a_2, by evaluating $\delta(q_1, a_2)$; let us suppose this state is q_2. We continue in this manner, finding states $q_3, q_4, \ldots, q_n$ such that $\delta(q_{i-1}, a_i) = q_i$ for each i. If q_n is a member of F, then the input $a_1 a_2 \cdots a_n$ is accepted, and if not then it is "rejected."

Example 2.1: Let us formally specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 somewhere in the string. We can write this language L as:

$$\{w \mid w \text{ is of the form } x01y \text{ for some strings } x \text{ and } y \text{ consisting of 0's and 1's only}\}$$

Another equivalent description, using parameters x and y to the left of the vertical bar, is:

$$\{x01y \mid x \text{ and } y \text{ are any strings of 0's and 1's}\}$$

Examples of strings in the language include 01, 11010, and 100011. Examples of strings *not* in the language include ϵ, 0, and 111000.

What do we know about an automaton that can accept this language L? First, its input alphabet is $\Sigma = \{0, 1\}$. It has some set of states, Q, of which one, say q_0, is the start state. This automaton has to remember the important facts about what inputs it has seen so far. To decide whether 01 is a substring of the input, A needs to remember:

1. Has it already seen 01? If so, then it accepts every sequence of further inputs; i.e., it will only be in accepting states from now on.

2. Has it never seen 01, but its most recent input was 0, so if it now sees a 1, it will have seen 01 and can accept everything it sees from here on?

3. Has it never seen 01, but its last input was either nonexistent (it just started) or it last saw a 1? In this case, A cannot accept until it first sees a 0 and then sees a 1 immediately after.

These three conditions can each be represented by a state. Condition (3) is represented by the start state, q_0. Surely, when just starting, we need to see a 0 and then a 1. But if in state q_0 we next see a 1, then we are no closer to seeing 01, and so we must stay in state q_0. That is, $\delta(q_0, 1) = q_0$.

However, if we are in state q_0 and we next see a 0, we are in condition (2). That is, we have never seen 01, but we have our 0. Thus, let us use q_2 to represent condition (2). Our transition from q_0 on input 0 is $\delta(q_0, 0) = q_2$.

Now, let us consider the transitions from state q_2. If we see a 0, we are no better off than we were, but no worse either. We have not seen 01, but 0 was the last symbol, so we are still waiting for a 1. State q_2 describes this situation perfectly, so we want $\delta(q_2, 0) = q_2$. If we are in state q_2 and we see a 1 input, we now know there is a 0 followed by a 1. We can go to an accepting state, which we shall call q_1, and which corresponds to condition (1) above. That is, $\delta(q_2, 1) = q_1$.

Finally, we must design the transitions for state q_1. In this state, we have already seen a 01 sequence, so regardless of what happens, we shall still be in a situation where we've seen 01. That is, $\delta(q_1, 0) = \delta(q_1, 1) = q_1$.

Thus, $Q = \{q_0, q_1, q_2\}$. As we said, q_0 is the start state, and the only accepting state is q_1; that is, $F = \{q_1\}$. The complete specification of the automaton A that accepts the language L of strings that have a 01 substring, is

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

where δ is the transition function described above. $\square$

2.2.4 Extending the Transition Function to Strings

We have explained informally that the DFA defines a language: the set of all strings that result in a sequence of state transitions from the start state to an accepting state. In terms of the transition diagram, the language of a DFA is the set of labels along all the paths that lead from the start state to any accepting state.

Now, we need to make the notion of the language of a DFA precise. To do so, we define an *extended transition function* that describes what happens when we start in any state and follow any sequence of inputs. If δ is our transition function, then the extended transition function constructed from δ will be called $\hat{\delta}$. The extended transition function is a function that takes a state q and a string w and returns a state p — the state that the automaton reaches when starting in state q and processing the sequence of inputs w. We define $\hat{\delta}$ by induction on the length of the input string, as follows:

BASIS: $\hat{\delta}(q, \epsilon) = q$. That is, if we are in state q and read no inputs, then we are still in state q.

INDUCTION: Suppose w is a string of the form xa; that is, a is the last symbol of w, and x is the string consisting of all but the last symbol.[3] For example, $w = 1101$ is broken into $x = 110$ and $a = 1$. Then

$$\hat{\delta}(q, w) = \delta\big(\hat{\delta}(q, x), a\big) \tag{2.1}$$

Now (2.1) may seem like a lot to take in, but the idea is simple. To compute $\hat{\delta}(q, w)$, first compute $\hat{\delta}(q, x)$, the state that the automaton is in after processing all but the last symbol of w. Suppose this state is p; that is, $\hat{\delta}(q, x) = p$. Then $\hat{\delta}(q, w)$ is what we get by making a transition from state p on input a, the last symbol of w. That is, $\hat{\delta}(q, w) = \delta(p, a)$.

Example 2.4: Let us design a DFA to accept the language

$$L = \{w \mid w \text{ has both an even number of 0's and an even number of 1's}\}$$

It should not be surprising that the job of the states of this DFA is to count both the number of 0's and the number of 1's, but count them modulo 2. That is, the state is used to remember whether the number of 0's seen so far is even or odd, and also to remember whether the number of 1's seen so far is even or odd. There are thus four states, which can be given the following interpretations:

q_0: Both the number of 0's seen so far and the number of 1's seen so far are even.

q_1: The number of 0's seen so far is even, but the number of 1's seen so far is odd.

q_2: The number of 1's seen so far is even, but the number of 0's seen so far is odd.

q_3: Both the number of 0's seen so far and the number of 1's seen so far are odd.

State q_0 is both the start state and the lone accepting state. It is the start state, because before reading any inputs, the numbers of 0's and 1's seen so far are both zero, and zero is even. It is the only accepting state, because it describes exactly the condition for a sequence of 0's and 1's to be in language L.

We now know almost how to specify the DFA for language L. It is

$$A = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

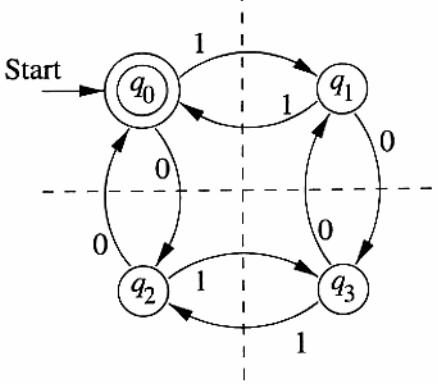


Figure 2.6: Transition diagram for the DFA of Example 2.4

where the transition function δ is described by the transition diagram of Fig. 2.6. Notice how each input 0 causes the state to cross the horizontal, dashed line. Thus, after seeing an even number of 0's we are always above the line, in state q_0 or q_1 while after seeing an odd number of 0's we are always below the line, in state q_2 or q_3. Likewise, every 1 causes the state to cross the vertical, dashed line. Thus, after seeing an even number of 1's, we are always to the left, in state q_0 or q_2, while after seeing an odd number of 1's we are to the right, in state q_1 or q_3. These observations are an informal proof that the four states have the interpretations attributed to them. However, one could prove the correctness of our claims about the states formally, by a mutual induction in the spirit of Example 1.23.

We can also represent this DFA by a transition table. Figure 2.7 shows this table. However, we are not just concerned with the design of this DFA; we want to use it to illustrate the construction of $\hat{\delta}$ from its transition function δ. Suppose the input is 110101. Since this string has even numbers of 0's and 1's both, we expect it is in the language. Thus, we expect that $\hat{\delta}(q_0, 110101) = q_0$, since q_0 is the only accepting state. Let us now verify that claim.

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Figure 2.7: Transition table for the DFA of Example 2.4

The check involves computing $\hat{\delta}(q_0, w)$ for each prefix w of 110101, starting at ϵ and going in increasing size. The summary of this calculation is:

- $\hat{\delta}(q_0, \epsilon) = q_0$.
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$.
- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$.
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$.
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$.
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$.
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$.

□

2.2.5 The Language of a DFA

Now, we can define the *language* of a DFA $A = (Q, \Sigma, \delta, q_0, F)$. This language is denoted $L(A)$, and is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$

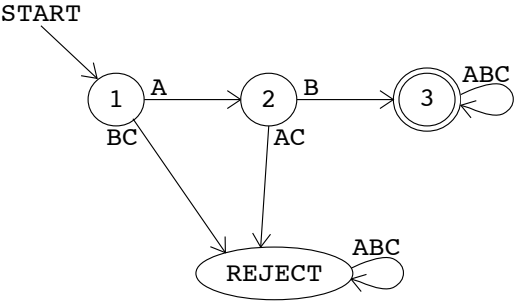
That is, the language of A is the set of strings w that take the start state q_0 to one of the accepting states. If L is $L(A)$ for some DFA A, then we say L is a *regular language.*

Dead States and DFA's Missing Some Transitions

We have formally defined a DFA to have a transition from any state, on any input symbol, to exactly one state. However, sometimes, it is more convenient to design the DFA to "die" in situations where we know it is impossible for any extension of the input sequence to be accepted.

Technically, this automaton is not a DFA, because it lacks transitions on most symbols from each of its states.


START
1
A
2
B
3
ABC


START
1
A
2
B
3
ABC
BC
AC
ABC
REJECT

In general, we can add a dead state to any automaton that has *no more* than one transition for any state and input symbol. Then, add a transition to the dead state from each other state q, on all input symbols for which q has no other transition. The result will be a DFA in the strict sense. Thus, we shall sometimes refer to an automaton as a DFA if it has *at most* one transition out of any state on any symbol, rather than if it has *exactly one* transition.

2.3 Nondeterministic Finite Automata

A "nondeterministic" finite automaton (*NFA*) has the power to be in several states at once. This ability is often expressed as an ability to "guess" something about its input. For instance, when the automaton is used to search for certain sequences of characters (e.g., keywords) in a long text string, it is helpful to "guess" that we are at the beginning of one of those strings and use a sequence of states to do nothing but check that the string appears, character by character.

2.3.2 Definition of Nondeterministic Finite Automata

Now, let us introduce the formal notions associated with nondeterministic finite automata. The differences between DFA's and NFA's will be pointed out as we do. An NFA is represented essentially like a DFA:

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

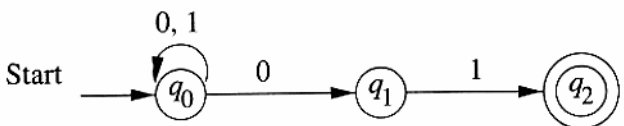
where:

1. Q is a finite set of *states*.
2. Σ is a finite set of *input symbols*.
3. q_0, a member of Q, is the *start state*.
4. F, a subset of Q, is the set of *final* (or *accepting*) states.
5. δ, the *transition function* is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q. Notice that the only difference between an NFA and a DFA is in the type of value that δ returns: a set of states in the case of an NFA and a single state in the case of a DFA.

Example 2.7: The NFA of Fig. 2.9 can be specified formally as

$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

Example 2.6: Figure 2.9 shows a nondeterministic finite automaton, whose job is to accept all and only the strings of 0's and 1's that end in 01. State q_0 is the start state, and we can think of the automaton as being in state q_0 (perhaps among other states) whenever it has not yet "guessed" that the final 01 has begun. It is always possible that the next symbol does not begin the final 01, even if that symbol is 0. Thus, state q_0 may transition to itself on both 0 and 1.



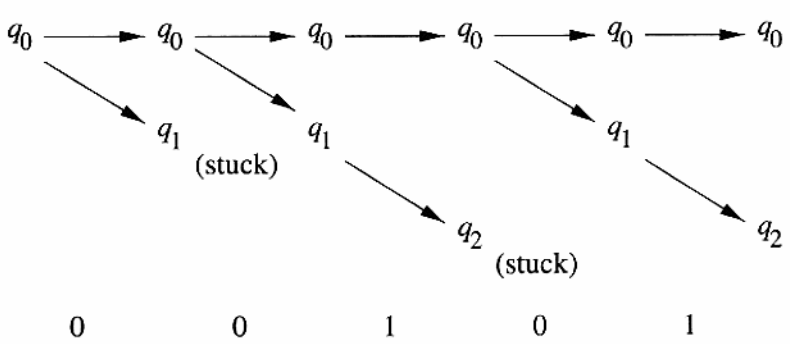


Figure 2.10: The states an NFA is in during the processing of input sequence 00101

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

Figure 2.11: Transition table for an NFA that accepts all strings ending in 01

Notice that transition tables can be used to specify the transition function for an NFA as well as for a DFA. The only difference is that each entry in the table for the NFA is a set, even if the set is a *singleton* (has one member). Also notice that when there is no transition at all from a given state on a given input symbol, the proper entry is $\emptyset$, the empty set.

2.3.3 The Extended Transition Function

As for DFA's, we need to extend the transition function δ of an NFA to a function $\hat{\delta}$ that takes a state q and a string of input symbols w, and returns the set of states that the NFA is in if it starts in state q and processes the string w. The idea was suggested by Fig. 2.10; in essence $\hat{\delta}(q, w)$ is the column of states found after reading w, if q is the lone state in the first column. For instance, Fig. 2.10 suggests that $\hat{\delta}(q_0, 001) = \{q_0, q_2\}$. Formally, we define $\hat{\delta}$ for an NFA's transition function δ by:

BASIS: $\hat{\delta}(q, \epsilon) = \{q\}$. That is, without reading any input symbols, we are only in the state we began in.

INDUCTION: Suppose w is of the form $w = xa$, where a is the final symbol of w and x is the rest of w. Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$. Let

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$$

Then $\hat{\delta}(q, w) = \{r_1, r_2, \ldots, r_m\}$. Less formally, we compute $\hat{\delta}(q, w)$ by first computing $\hat{\delta}(q, x)$, and by then following any transition from any of these states that is labeled a.

Example 2.8: Let us use $\hat{\delta}$ to describe the processing of input 00101 by the NFA of Fig. 2.9. A summary of the steps is:

1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$.
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$.

3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.

4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.

6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

Line (1) is the basis rule. We obtain line (2) by applying δ to the lone state, q_0, that is in the previous set, and get $\{q_0, q_1\}$ as a result. Line (3) is obtained by taking the union over the two states in the previous set of what we get when we apply δ to them with input 0. That is, $\delta(q_0, 0) = \{q_0, q_1\}$, while $\delta(q_1, 0) = \emptyset$. For line (4), we take the union of $\delta(q_0, 1) = \{q_0\}$ and $\delta(q_1, 1) = \{q_2\}$. Lines (5) and (6) are similar to lines (3) and (4). □

2.3.4 The Language of an NFA

As we have suggested, an NFA accepts a string w if it is possible to make any sequence of choices of next state, while reading the characters of w, and go from the start state to any accepting state. The fact that other choices using the input symbols of w lead to a nonaccepting state, or do not lead to any state at all (i.e., the sequence of states "dies"), does not prevent w from being accepted by the NFA as a whole. Formally, if $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA, then

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

That is, $L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

2.5 Finite Automata With Epsilon-Transitions

We shall now introduce another extension of the finite automaton. The new "feature" is that we allow a transition on ϵ, the empty string. In effect, an NFA is allowed to make a transition spontaneously, without receiving an input

2.5.1 Uses of ϵ-Transitions

We shall begin with an informal treatment of ϵ-NFA's, using transition diagrams with ϵ allowed as a label. In the examples to follow, think of the automaton as accepting those sequences of labels along paths from the start state to an accepting state. However, each ϵ along a path is "invisible"; i.e., it contributes nothing to the string along the path.

Example 2.16: In Fig. 2.18 is an ϵ-NFA that accepts decimal numbers consisting of:

1. An optional + or − sign,
2. A string of digits,
3. A decimal point, and
4. Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.

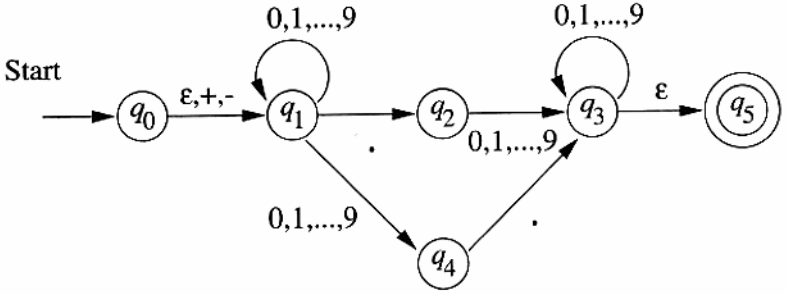


Figure 2.18: An ϵ-NFA accepting decimal numbers

Of particular interest is the transition from q_0 to q_1 on any of ϵ, +, or −. Thus, state q_1 represents the situation in which we have seen the sign if there is one, and perhaps some digits, but not the decimal point. State q_2 represents the situation where we have just seen the decimal point, and may or may not have seen prior digits. In q_4 we have definitely seen at least one digit, but not the decimal point. Thus, the interpretation of q_3 is that we have seen a decimal point and at least one digit, either before or after the decimal point. We may stay in q_3 reading whatever digits there are, and also have the option of "guessing" the string of digits is complete and going spontaneously to q_5, the accepting state. □

2.5.2 The Formal Notation for an ϵ-NFA

We may represent an ϵ-NFA exactly as we do an NFA, with one exception: the transition function must include information about transitions on ϵ. Formally, we represent an ϵ-NFA A by $A = (Q, \Sigma, \delta, q_0, F)$, where all components have their same interpretation as for an NFA, except that δ is now a function that takes as arguments:

1. A state in Q, and
2. A member of $\Sigma \cup \{\epsilon\}$, that is, either an input symbol, or the symbol ϵ. We require that ϵ, the symbol for the empty string, cannot be a member of the alphabet Σ, so no confusion results.

Example 2.18: The ϵ-NFA of Fig. 2.18 is represented formally as

$$E = (\{q_0, q_1, \ldots, q_5\}, \{., +, -, 0, 1, \ldots, 9\}, \delta, q_0, \{q_5\})$$

where δ is defined by the transition table in Fig. 2.20. □

	ϵ	$+, -$	.	$0, 1, \ldots, 9$
q_0	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

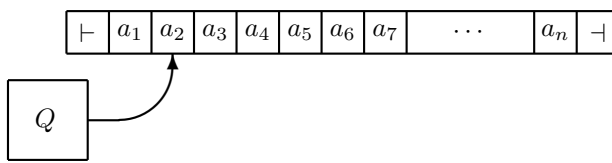
Figure 2.20: Transition table for Fig. 2.18

Two-Way Finite Automata

Two-way finite automata are similar to the machines we have been studying, except that they can read the input string in either direction. We think of them as having a *read head*, which can move left or right over the input string. Like ordinary finite automata, they have a finite set Q of *states* and can be either deterministic (2DFA) or nondeterministic (2NFA).

Although these automata appear much more powerful than one-way finite automata, in reality they are equivalent in the sense that they only accept regular sets. We will prove this result using the Myhill–Nerode theorem.

We think of the symbols of the input string as occupying cells of a finite tape, one symbol per cell. The input string is enclosed in left and right endmarkers $\vdash$ and $\dashv$, which are not elements of the input alphabet Σ. The read head may not move outside of the endmarkers.



Informally, the machine starts in its start state s with its read head pointing to the left endmarker. At any point in time, the machine is in some state q with its read head scanning some tape cell containing an input symbol a_i or one of the endmarkers. Based on its current state and the symbol occupying

the tape cell it is currently scanning, it moves its read head either left or right one cell and enters a new state. It *accepts* by entering a special accept state t and *rejects* by entering a special reject state r. The machine's action on a particular state and symbol is determined by a transition function δ that is part of the specification of the machine.

Example 17.1 Here is an informal description of a 2DFA accepting the set

$$A \quad = \quad \{x \in \{a,b\}^* \mid \#a(x) \text{ is a multiple of 3 and } \#b(x) \text{ is even}\}.$$

The machine starts in its start state scanning the left endmarker. It scans left to right over the input, counting the number of a's mod 3 and ignoring the b's. When it reaches the right endmarker $\dashv$, if the number of a's it has seen is not a multiple of 3, it enters its reject state, thereby rejecting the input—the input string x is not in the set A, since the first condition is not satisfied. Otherwise it scans right to left over the input, counting the number of b's mod 2 and ignoring the a's. When it reaches the left endmarker $\vdash$ again, if the number of b's it has seen is odd, it enters its reject state; otherwise, it enters its accept state. □

Unlike ordinary finite automata, a 2DFA needs only a single accept state and a single reject state. We can think of it as halting immediately when it enters one of these two states, although formally it keeps running but remains in the accept or reject state. The machine need not read the entire input before accepting or rejecting. Indeed, it need not ever accept or reject at all, but may loop infinitely without ever entering its accept or reject state.

Formal Definition of 2DFA

Formally, a 2DFA is an octuple

$$M \quad = \quad (Q,\ \Sigma,\ \vdash,\ \dashv,\ \delta,\ s,\ t,\ r),$$

where

- Q is a finite set (the *states*),
- Σ is a finite set (the *input alphabet*),
- $\vdash$ is the *left endmarker*, $\vdash \notin \Sigma$,
- $\dashv$ is the *right endmarker*, $\dashv \notin \Sigma$,
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to (Q \times \{L, R\})$ is the *transition function* (L, R stand for left and right, respectively),
- $s \in Q$ is the *start state*,
- $t \in Q$ is the *accept state*, and

- $r \in Q$ is the *reject state*, $r \neq t$,

such that for all states q,

$$\begin{array}{lcll} \delta(q, \vdash) & = & (u, R) & \text{for some } u \in Q, \\ \delta(q, \dashv) & = & (v, L) & \text{for some } v \in Q, \end{array} \tag{17.1}$$

and for all symbols $b \in \Sigma \cup \{\vdash\}$,

$$\begin{array}{ll} \delta(t, b) = (t, R), & \delta(r, b) = (r, R), \\ \delta(t, \dashv) = (t, L), & \delta(r, \dashv) = (r, L). \end{array} \tag{17.2}$$

Intuitively, the function δ takes a state and a symbol as arguments and returns a new state and a direction to move the head. If $\delta(p, b) = (q, d)$, then whenever the machine is in state p and scanning a tape cell containing symbol b, it moves its head one cell in the direction d and enters state q. The restrictions (17.1) prevent the machine from ever moving outside the input area. The restrictions (17.2) say that once the machine enters its accept or reject state, it stays in that state and moves its head all the way to the right of the tape. The octuple is not a legal 2DFA if its transition function δ does not satisfy these conditions.

Example 17.2 Here is a formal description of the 2DFA described informally in Example 17.1 above.

$$\begin{array}{lcl} Q & = & \{q_0, q_1, q_2, p_0, p_1, t, r\}, \\ \Sigma & = & \{a, b\}. \end{array}$$

The start, accept, and reject states are q_0, t, and r, respectively. The transition function δ is given by the following table:

	$\vdash$	a	b	$\dashv$
q_0	(q_0, R)	(q_1, R)	(q_0, R)	(p_0, L)
q_1	$-$	(q_2, R)	(q_1, R)	(r, L)
q_2	$-$	(q_0, R)	(q_2, R)	(r, L)
p_0	(t, R)	(p_0, L)	(p_1, L)	$-$
p_1	(r, R)	(p_1, L)	(p_0, L)	$-$
t	(t, R)	(t, R)	(t, R)	(t, L)
r	(r, R)	(r, R)	(r, R)	(r, L)

The entries marked $-$ will never occur in any computation, so it doesn't matter what we put here. The machine is in states q_0, q_1, or q_2 on the first pass over the input from left to right; it is in state q_i if the number of a's it has seen so far is i mod 3. The machine is in state p_0 or p_1 on the second pass over the input from right to left, the index indicating the parity of the number of b's it has seen so far. □

Configurations and Acceptance

Fix an input $x \in \Sigma^*$, say $x = a_1 a_2 \cdots a_n$. Let $a_0 = \vdash$ and $a_{n+1} = \dashv$. Then

$$a_0 a_1 a_2 \cdots a_n a_{n+1} \quad = \quad \vdash x \dashv .$$

A *configuration* of the machine on input x is a pair (q, i) such that $q \in Q$ and $0 \leq i \leq n+1$. Informally, the pair (q, i) gives a current state and current position of the read head. The *start configuration* is $(s, 0)$, meaning that the machine is in its start state s and scanning the left endmarker.

A binary relation $\xrightarrow[x]{1}$, the *next configuration relation*, is defined on configurations as follows:

$$\delta(p, a_i) = (q, L) \quad \Rightarrow \quad (p, i) \xrightarrow[x]{1} (q, i-1),$$
$$\delta(p, a_i) = (q, R) \quad \Rightarrow \quad (p, i) \xrightarrow[x]{1} (q, i+1).$$

The relation $\xrightarrow[x]{1}$ describes one step of the machine on input x. We define the relations $\xrightarrow[x]{n}$ inductively, $n \geq 0$:

- $(p, i) \xrightarrow[x]{0} (p, i)$; and
- if $(p, i) \xrightarrow[x]{n} (q, j)$ and $(q, j) \xrightarrow[x]{1} (u, k)$, then $(p, i) \xrightarrow[x]{n+1} (u, k)$.

The relation $\xrightarrow[x]{n}$ is just the n-fold composition of $\xrightarrow[x]{1}$. The relations $\xrightarrow[x]{n}$ are functions; that is, for any configuration (p, i), there is exactly one configuration (q, j) such that $(p, i) \xrightarrow[x]{n} (q, j)$. Now define

$$(p, i) \xrightarrow[x]{*} (q, j) \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \exists n \geq 0 \ (p, i) \xrightarrow[x]{n} (q, j).$$

Note that the definitions of these relations depend on the input x. The machine is said to *accept* the input x if

$$(s, 0) \xrightarrow[x]{*} (t, i) \quad \text{for some } i.$$

In other words, the machine enters its accept state at some point. The machine is said to *reject* the input x if

$$(s, 0) \xrightarrow[x]{*} (r, i) \quad \text{for some } i.$$

In other words, the machine enters its reject state at some point. It cannot both accept and reject input x by our assumption that $t \neq r$ and by properties (17.2). The machine is said to *halt* on input x if it either accepts x or rejects x. Note that this is a purely mathematical definition—the machine doesn't really grind to a halt! It is possible that the machine neither accepts nor rejects x, in which case it is said to *loop* on x. The set $L(M)$ is defined to be the set of strings accepted by M.

2.3.5 Equivalence of Deterministic and Nondeterministic Finite Automata

Subset Construction

Although there are many languages for which an NFA is easier to construct than a DFA, such as the language (Example 2.6) of strings that end in 01, it is a surprising fact that every language that can be described by some NFA can also be described by some DFA. Moreover, the DFA in practice has about as many states as the NFA, although it often has more transitions. In the worst case, however, the smallest DFA can have 2^n states while the smallest NFA for the same language has only n states.

The proof that DFA's can do whatever NFA's can do involves an important "construction" called the *subset construction* because it involves constructing all subsets of the set of states of the NFA. In general, many proofs about automata involve constructing one automaton from another. It is important for us to observe the subset construction as an example of how one formally describes one automaton in terms of the states and transitions of another, without knowing the specifics of the latter automaton.

The subset construction starts from an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Its goal is the description of a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(D) = L(N)$. Notice that the input alphabets of the two automata are the same, and the start state of D is the set containing only the start state of N. The other components of D are constructed as follows.

- Q_D is the set of subsets of Q_N; i.e., Q_D is the *power set* of Q_N. Note that if Q_N has n states, then Q_D will have 2^n states. Often, not all these states are accessible from the start state of Q_D. Inaccessible states can be "thrown away," so effectively, the number of states of D may be much smaller than 2^n.

- F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. That is, F_D is all sets of N's states that include at least one accepting state of N.

- For each set $S \subseteq Q_N$ and for each input symbol a in Σ,

$$\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$$

  That is, to compute $\delta_D(S, a)$ we look at all the states p in S, see what states N goes to from p on input a, and take the union of all those states.

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Figure 2.12: The complete subset construction from Fig. 2.9

Example 2.10: Let N be the automaton of Fig. 2.9 that accepts all strings that end in 01. Since N's set of states is $\{q_0, q_1, q_2\}$, the subset construction

produces a DFA with $2^3 = 8$ states, corresponding to all the subsets of these three states. Figure 2.12 shows the transition table for these eight states; we shall show shortly the details of how some of these entries are computed.

Notice that this transition table belongs to a deterministic finite automaton. Even though the entries in the table are sets, the states of the constructed DFA *are* sets. To make the point clearer, we can invent new names for these states, e.g., A for $\emptyset$, B for $\{q_0\}$, and so on. The DFA transition table of Fig 2.13 defines exactly the same automaton as Fig. 2.12, but makes clear the point that the entries in the table are single states of the DFA.

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$*D$	A	A
E	E	F
$*F$	E	B
$*G$	A	D
$*H$	E	F

Figure 2.13: Renaming the states of Fig. 2.12

Of the eight states in Fig. 2.13, starting in the start state B, we can only reach states B, E, and F. The other five states are inaccessible from the start state and may as well not be there. We often can avoid the exponential-time step of constructing transition-table entries for every subset of states if we perform "lazy evaluation" on the subsets, as follows.

BASIS: We know for certain that the singleton set consisting only of N's start state is accessible.

INDUCTION: Suppose we have determined that set S of states is accessible. Then for each input symbol a, compute the set of states $\delta_D(S, a)$; we know that these sets of states will also be accessible.

For the example at hand, we know that $\{q_0\}$ is a state of the DFA D. We find that $\delta_D(\{q_0\}, 0) = \{q_0, q_1\}$ and $\delta_D(\{q_0\}, 1) = \{q_0\}$. Both these facts are established by looking at the transition diagram of Fig. 2.9 and observing that on 0 there are arcs out of q_0 to both q_0 and q_1, while on 1 there is an arc only to q_0. We thus have one row of the transition table for the DFA: the second row in Fig. 2.12.

One of the two sets we computed is "old"; $\{q_0\}$ has already been considered. However, the other — $\{q_0, q_1\}$ — is new and its transitions must be computed. We find $\delta_D(\{q_0, q_1\}, 0) = \{q_0, q_1\}$ and $\delta_D(\{q_0, q_1\}, 1) = \{q_0, q_2\}$. For instance, to see the latter calculation, we know that

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

We now have the fifth row of Fig. 2.12, and we have discovered one new state of D, which is $\{q_0, q_2\}$. A similar calculation tells us

$$\delta_D(\{q_0, q_2\}, 0) = \delta_N(q_0, 0) \cup \delta_N(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$$
$$\delta_D(\{q_0, q_2\}, 1) = \delta_N(q_0, 1) \cup \delta_N(q_2, 1) = \{q_0\} \cup \emptyset = \{q_0\}$$

These calculations give us the sixth row of Fig. 2.12, but it gives us only sets of states that we have already seen.

Thus, the subset construction has converged; we know all the accessible states and their transitions. The entire DFA is shown in Fig. 2.14. Notice that it has only three states, which is, by coincidence, exactly the same number of states as the NFA of Fig. 2.9, from which it was constructed. However, the DFA of Fig. 2.14 has six transitions, compared with the four transitions in Fig. 2.9. □

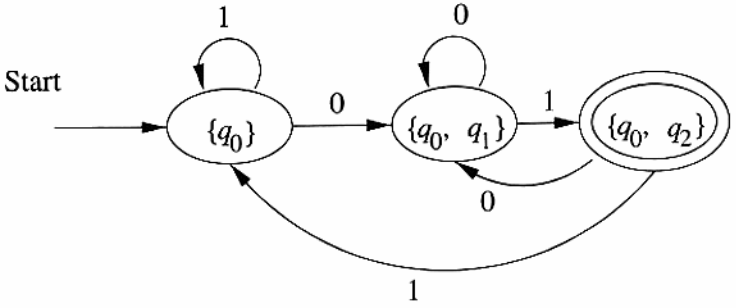


Figure 2.14: The DFA constructed from the NFA of Fig 2.9

We need to show formally that the subset construction works, although the intuition was suggested by the examples. After reading sequence of input symbols w, the constructed DFA is in one state that is the set of NFA states that the NFA would be in after reading w. Since the accepting states of the DFA are those sets that include at least one accepting state of the NFA, and the NFA also accepts if it gets into at least one of its accepting states, we may then conclude that the DFA and NFA accept exactly the same strings, and therefore accept the same language.

Theorem 2.11: If $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ is the DFA constructed from NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ by the subset construction, then $L(D) = L(N)$.

PROOF: What we actually prove first, by induction on $|w|$, is that

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$$

Notice that each of the $\hat{\delta}$ functions returns a set of states from Q_N, but $\hat{\delta}_D$ interprets this set as one of the states of Q_D (which is the power set of Q_N), while $\hat{\delta}_N$ interprets this set as a subset of Q_N.

BASIS: Let $|w| = 0$; that is, $w = \epsilon$. By the basis definitions of $\hat{\delta}$ for DFA's and NFA's, both $\hat{\delta}_D(\{q_0\}, \epsilon)$ and $\hat{\delta}_N(q_0, \epsilon)$ are $\{q_0\}$.

INDUCTION: Let w be of length $n+1$, and assume the statement for length n. Break w up as $w = xa$, where a is the final symbol of w. By the inductive hypothesis, $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$. Let both these sets of N's states be $\{p_1, p_2, \ldots, p_k\}$.

The inductive part of the definition of $\hat{\delta}$ for NFA's tells us

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \tag{2.2}$$

The subset construction, on the other hand, tells us that

$$\delta_D(\{p_1, p_2, \ldots, p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \tag{2.3}$$

Now, let us use (2.3) and the fact that $\hat{\delta}_D(\{q_0\}, x) = \{p_1, p_2, \ldots, p_k\}$ in the inductive part of the definition of $\hat{\delta}$ for DFA's:

$$\hat{\delta}_D(\{q_0\}, w) = \delta_D\big(\hat{\delta}_D(\{q_0\}, x), a\big) = \delta_D(\{p_1, p_2, \ldots, p_k\}, a) = \bigcup_{i=1}^{k} \delta_N(p_i, a) \tag{2.4}$$

Thus, Equations (2.2) and (2.4) demonstrate that $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$. When we observe that D and N both accept w if and only if $\hat{\delta}_D(\{q_0\}, w)$ or $\hat{\delta}_N(q_0, w)$, respectively, contain a state in F_N, we have a complete proof that $L(D) = L(N)$. □

Theorem 2.12: A language L is accepted by some DFA if and only if L is accepted by some NFA.

PROOF: (If) The "if" part is the subset construction and Theorem 2.11.

(Only-if) This part is easy; we have only to convert a DFA into an identical NFA. Put intuitively, if we have the transition diagram for a DFA, we can also interpret it as the transition diagram of an NFA, which happens to have exactly one choice of transition in any situation. More formally, let $D = (Q, \Sigma, \delta_D, q_0, F)$ be a DFA. Define $N = (Q, \Sigma, \delta_N, q_0, F)$ to be the equivalent NFA, where δ_N is defined by the rule:

- If $\delta_D(q, a) = p$, then $\delta_N(q, a) = \{p\}$.

It is then easy to show by induction on $|w|$, that if $\hat{\delta}_D(q_0, w) = p$ then

$$\hat{\delta}_N(q_0, w) = \{p\}$$

We leave the proof to the reader. As a consequence, w is accepted by D if and only if it is accepted by N; i.e., $L(D) = L(N)$. □

2.5.5 Eliminating ϵ-Transitions

Given any ϵ-NFA E, we can find a DFA D that accepts the same language as E. The construction we use is very close to the subset construction, as the states of D are subsets of the states of E. The only difference is that we must incorporate ϵ-transitions of E, which we do through the mechanism of the ϵ-closure.

Let $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$. Then the equivalent DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

is defined as follows:

1. Q_D is the set of subsets of Q_E. More precisely, we shall find that all accessible states of D are *ϵ-closed* subsets of Q_E, that is, sets $S \subseteq Q_E$ such that $S = \text{ECLOSE}(S)$. Put another way, the ϵ-closed sets of states S are those such that any ϵ-transition out of one of the states in S leads to a state that is also in S. Note that $\emptyset$ is an ϵ-closed set.

2. $q_D = \text{ECLOSE}(q_0)$; that is, we get the start state of D by closing the set consisting of only the start state of E. Note that this rule differs from the original subset construction, where the start state of the constructed automaton was just the set containing the start state of the given NFA.

3. F_D is those sets of states that contain at least one accepting state of E. That is, $F_D = \{S \mid S \text{ is in } Q_D \text{ and } S \cap F_E \neq \emptyset\}$.

4. $\delta_D(S, a)$ is computed, for all a in Σ and sets S in Q_D by:

(a) Let $S = \{p_1, p_2, \ldots, p_k\}$.

(b) Compute $\bigcup_{i=1}^{k} \delta_E(p_i, a)$; let this set be $\{r_1, r_2, \ldots, r_m\}$.

(c) Then $\delta_D(S, a) = \bigcup_{j=1}^{m} \text{ECLOSE}(r_j)$.

We have now explained the arcs out of $\{q_0, q_1\}$ in Fig. 2.22. The other transitions are computed similarly, and we leave them for you to check. Since q_5 is the only accepting state of E, the accepting states of D are those accessible states that contain q_5. We see these two sets $\{q_3, q_5\}$ and $\{q_2, q_3, q_5\}$ indicated by double circles in Fig. 2.22. □

Theorem 2.22: A language L is accepted by some ϵ-NFA if and only if L is accepted by some DFA.

PROOF: (If) This direction is easy. Suppose $L = L(D)$ for some DFA. Turn D into an ϵ-DFA E by adding transitions $\delta(q, \epsilon) = \emptyset$ for all states q of D. Technically, we must also convert the transitions of D on input symbols, e.g., $\delta_D(q, a) = p$ into an NFA-transition to the set containing only p, that is $\delta_E(q, a) = \{p\}$. Thus, the transitions of E and D are the same, but E explicitly states that there are no transitions out of any state on ϵ.

(Only-if) Let $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ be an ϵ-NFA. Apply the modified subset construction described above to produce the DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

We need to show that $L(D) = L(E)$, and we do so by showing that the extended transition functions of E and D are the same. Formally, we show $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ by induction on the length of w.

BASIS: If $|w| = 0$, then $w = \epsilon$. We know $\hat{\delta}_E(q_0, \epsilon) = \text{ECLOSE}(q_0)$. We also know that $q_D = \text{ECLOSE}(q_0)$, because that is how the start state of D is defined. Finally, for a DFA, we know that $\hat{\delta}(p, \epsilon) = p$ for any state p, so in particular, $\hat{\delta}_D(q_D, \epsilon) = \text{ECLOSE}(q_0)$. We have thus proved that $\hat{\delta}_E(q_0, \epsilon) = \hat{\delta}_D(q_D, \epsilon)$.

INDUCTION: Suppose $w = xa$, where a is the final symbol of w, and assume that the statement holds for x. That is, $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x)$. Let both these sets of states be $\{p_1, p_2, \ldots, p_k\}$.

By the definition of $\hat{\delta}$ for ϵ-NFA's, we compute $\hat{\delta}_E(q_0, w)$ by:

1. Let $\{r_1, r_2, \ldots, r_m\}$ be $\bigcup_{i=1}^{k} \delta_E(p_i, a)$.

2. Then $\hat{\delta}_E(q_0, w) = \bigcup_{j=1}^{m} \text{ECLOSE}(r_j)$.

If we examine the construction of DFA D in the modified subset construction above, we see that $\delta_D(\{p_1, p_2, \ldots, p_k\}, a)$ is constructed by the same two steps (1) and (2) above. Thus, $\hat{\delta}_D(q_D, w)$, which is $\delta_D(\{p_1, p_2, \ldots, p_k\}, a)$ is the same set as $\hat{\delta}_E(q_0, w)$. We have now proved that $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ and completed the inductive part. □

2DFAs and Regular Sets

In this lecture we show that 2DFAs are no more powerful than ordinary DFAs. Here is the idea. Consider a long input string broken up in an arbitrary place into two substrings xz. How much information about x can the machine carry across the boundary from x into z? Since the machine is two-way, it can cross the boundary between x and z several times. Each time it crosses the boundary moving from right to left, that is, from z into x, it does so in some state q. When it crosses the boundary again moving from left to right (if ever), it comes out of x in some state, say p. Now if it ever goes into x in the future in state q again, it will emerge again in state p, because its future action is completely determined by its current configuration (state and head position). Moreover, the state p depends only on q and x. We will write $T_x(q) = p$ to denote this relationship. We can keep track of all such information by means of a finite table

$$T_x : (Q \cup \{\bullet\}) \quad \rightarrow \quad (Q \cup \{\perp\}),$$

where Q is the set of states of the 2DFA M, and $\bullet$ and $\perp$ are two other objects not in Q whose purpose is described below.

On input xz, the machine M starts in its start state scanning the left endmarker. As it computes, it moves its read head. The head may eventually cross the boundary moving left to right from x into z. The first time it does so (if ever), it is in some state, which we will call $T_x(\bullet)$ (this is the purpose of $\bullet$). The machine may *never* emerge from x; in this case we

write $T_x(\bullet) = \perp$ (this is the purpose of $\perp$). The state $T_x(\bullet)$ gives some information about x, but only a finite amount of information, since there are only finitely many possibilities for $T_x(\bullet)$. Note also that $T_x(\bullet)$ depends only on x and not on z: if the input were xw instead of xz, the first time the machine passed the boundary from x into w, it would also be in state $T_x(\bullet)$, because its action up to that point is determined only by x; it hasn't seen anything to the right of the boundary yet.

If $T_x(\bullet) = \perp$, M must be in an infinite loop inside x and will never accept or reject, by our assumption about moving all the way to the right endmarker whenever it accepts or rejects.

Suppose that the machine does emerge from x into z. It may wander around in z for a while, then later may move back into x from right to left in state q. If this happens, then it will either

- eventually emerge from x again in some state p, in which case we define $T_x(q) = p$; or
- never emerge, in which case we define $T_x(q) = \perp$.

Again, note that $T_x(q)$ depends only on x and q and not on z. If the machine entered x from the right on input xw in state q, then it would emerge again in state $T_x(q)$ (or never emerge, if $T_x(q) = \perp$), because M is deterministic, and its behavior while inside x is completely determined by x and the state it entered x in.

If we write down $T_x(q)$ for every state q along with $T_x(\bullet)$, this gives all the information about x one could ever hope to carry across the boundary from x to z. One can imagine an agent sitting to the right of the boundary between x and z, trying to obtain information about x. All it is allowed to do is observe the state $T_x(\bullet)$ the first time the machine emerges from x (if ever) and later send probes into x in various states q to see what state $T_x(q)$ the machine comes out in (if at all). If y is another string such that $T_y = T_x$, then x and y will be indistinguishable from the agent's point of view.

Now note that there are only finitely many possible tables

$$T : (Q \cup \{\bullet\}) \quad \rightarrow \quad (Q \cup \{\perp\}),$$

namely $(k+1)^{k+1}$, where k is the size of Q. Thus there is only a finite amount of information about x that can be passed across the boundary to the right of x, and it is all encoded in the table T_x.

Note also that if $T_x = T_y$ and M accepts xz, then M accepts yz. This is because the sequence of states the machine is in as it passes the boundary between x and z (or between y and z) in either direction is completely determined by the table $T_x = T_y$ and z. To accept xz, the machine must at some point be scanning the right endmarker in its accept state t. Since

the sequence of states along the boundary is the same and the action when the machine is scanning z is the same, this also must happen on input yz.

Now we can use the Myhill–Nerode theorem to show that $L(M)$ is regular. We have just argued that

$$\begin{aligned} T_x = T_y \quad &\Rightarrow \quad \forall z \ (M \text{ accepts } xz \iff M \text{ accepts } yz) \\ &\iff \quad \forall z \ (xz \in L(M) \iff yz \in L(M)) \\ &\iff \quad x \equiv_{L(M)} y, \end{aligned}$$

where $\equiv_{L(M)}$ is the relation first defined in Eq. (16.1) of Lecture 16. Thus if two strings have the same table, then they are equivalent under $\equiv_{L(M)}$. Since there are only finitely many tables, the relation $\equiv_{L(M)}$ has only finitely many equivalence classes, at most one for each table; therefore, $\equiv_{L(M)}$ is of finite index. By the Myhill–Nerode theorem, $L(M)$ is a regular set.

Constructing a DFA

The argument above may be a bit unsatisfying, since it does not explicitly construct a DFA equivalent to a given 2DFA M. We can easily do so, however. Intuitively, we can build a DFA whose states correspond to the tables.

Formally, define

$$x \equiv y \quad \stackrel{\text{def}}{\iff} \quad T_x = T_y.$$

That is, call two strings in Σ^* equivalent if they have the same table. There are only finitely many equivalence classes, at most one for each table; thus $\equiv$ is of finite index. We can also show the following:

(i) The table T_{xa} is uniquely determined by T_x and a; that is, if $T_x = T_y$, then $T_{xa} = T_{ya}$. This says that $\equiv$ is a right congruence.

(ii) Whether or not x is accepted by M is completely determined by T_x; that is, if $T_x = T_y$, then either both x and y are accepted by M or neither is. This says that $\equiv$ refines $L(M)$.

These observations together say that $\equiv$ is a Myhill–Nerode relation for $L(M)$. Using the construction $\equiv \mapsto M_{\equiv}$ described in Lecture 15, we can obtain a DFA for $L(M)$ explicitly.

To show (i), we show how to construct T_{xa} from T_x and a.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_k \in Q$ such that $\delta(p_i, a) = (q_i, L)$ and $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $\delta(p_k, a) = (q_k, R)$, then $T_{xa}(p_0) = q_k$.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_{k-1} \in Q$ such that $\delta(p_i, a) = (q_i, L)$ and $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $p_k = p_i$, $i < k$, then $T_{xa}(p_0) = \perp$.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_k \in Q$ such that $\delta(p_i, a) = (q_i, L)$, $0 \le i \le k$, $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $T_x(q_k) = \perp$, then $T_{xa}(p_0) = \perp$.

- If $T_x(\bullet) = \perp$, then $T_{xa}(\bullet) = \perp$.

- If $T_x(\bullet) = p$, then $T_{xa}(\bullet) = T_{xa}(p)$.

For (ii), suppose $T_x = T_y$ and consider the sequence of states M is in as it crosses the boundary in either direction between the input string and the right endmarker $\dashv$. This sequence is the same on input x as it is on input y, since it is completely determined by the table. Both strings x and y are accepted iff this sequence contains the accept state t.

We have shown that the relation $\equiv$ is a Myhill–Nerode relation for $L(M)$, where M is an arbitrary 2DFA. The construction $\equiv \mapsto M_{\equiv}$ of Lecture 15 gives a DFA equivalent to M. Recall that in that construction, the states of the DFA correspond in a one-to-one fashion with the $\equiv$-classes; and here, each $\equiv$-class $[x]$ corresponds to a table $T_x : (Q \cup \{\bullet\}) \to (Q \cup \{\perp\})$.

If we wanted to, we could build a DFA M' directly from the tables:

$$\begin{aligned} Q' &\stackrel{\text{def}}{=} \{T : (Q \cup \{\bullet\}) \to (Q \cup \{\perp\})\}, \\ s' &\stackrel{\text{def}}{=} T_\epsilon, \\ \delta'(T_x, a) &\stackrel{\text{def}}{=} T_{xa}, \\ F' &\stackrel{\text{def}}{=} \{T_x \mid x \in L(M)\}. \end{aligned}$$

The transition function δ' is well defined because of property (i), and $T_x \in F'$ iff $x \in L(M)$ by property (ii). As usual, one can prove by induction on $|y|$ that

$$\widehat{\delta}'(T_x, y) = T_{xy};$$

then

$$\begin{aligned} x \in L(M') &\iff \widehat{\delta}'(s', x) \in F' \\ &\iff \widehat{\delta}'(T_\epsilon, x) \in F' \\ &\iff T_x \in F' \\ &\iff x \in L(M). \end{aligned}$$

Thus $L(M') = L(M)$.

More on Regular Sets

Here is another example of a regular set that is a little harder than the example given last time. Consider the set

$$\{x \in \{0,1\}^* \mid x \text{ represents a multiple of three in binary}\} \tag{4.1}$$

(leading zeros permitted, ϵ represents the number 0). For example, the following binary strings represent multiples of three and should be accepted:

Binary	*Decimal equivalent*
0	0
11	3
110	6
1001	9
1100	12
1111	15
10010	18
⋮	⋮

Strings not representing multiples of three should be rejected. Here is an automaton accepting the set (4.1):

		0	1
→	**0***F*	**0**	**1**
	1	**2**	**0**
	2	**1**	**2**

The states **0**, **1**, **2** are written in boldface to distinguish them from the input symbols $0, 1$.

In the diagram, the states are **0**, **1**, **2** from left to right. We prove that this automaton accepts exactly the set (4.1) by induction on the length of the input string. First we associate a meaning to each state:

if the number represented by the string scanned so far is[1]	*then the machine will be in state*
0 mod 3	**0**
1 mod 3	**1**
2 mod 3	**2**

Let $\#x$ denote the number represented by string x in binary. For example,

$$\begin{aligned} \#\epsilon &= 0, \\ \#0 &= 0, \\ \#11 &= 3, \\ \#100 &= 4, \end{aligned}$$

and so on. Formally, we want to show that for any string x in $\{0,1\}^*$,

$$\begin{aligned} \widehat{\delta}(\mathbf{0}, x) &= \mathbf{0} \text{ iff } \#x \equiv 0 \bmod 3, \\ \widehat{\delta}(\mathbf{0}, x) &= \mathbf{1} \text{ iff } \#x \equiv 1 \bmod 3, \\ \widehat{\delta}(\mathbf{0}, x) &= \mathbf{2} \text{ iff } \#x \equiv 2 \bmod 3, \end{aligned} \tag{4.2}$$

or in short,

$$\widehat{\delta}(\mathbf{0}, x) = \#x \bmod 3. \tag{4.3}$$

This will be our induction hypothesis. The final result we want, namely (4.2), is a weaker consequence of (4.3), but we need the more general statement (4.3) for the induction hypothesis.

We have by elementary number theory that

$$\#(x0) = 2(\#x) + 0,$$

$$\#(x1) = 2(\#x) + 1,$$

or in short,

$$\#(xc) = 2(\#x) + c \tag{4.4}$$

for $c \in \{0,1\}$. From the machine above, we see that for any state $q \in \{\mathbf{0},\mathbf{1},\mathbf{2}\}$ and input symbol $c \in \{0,1\}$,

$$\delta(q,c) = (2q + c) \bmod 3. \tag{4.5}$$

This can be verified by checking all six cases corresponding to possible choices of q and c. (In fact, (4.5) would have been a great way to *define* the transition function formally—then we wouldn't have had to prove it!) Now we use the inductive definition of $\widehat{\delta}$ to show (4.3) by induction on $|x|$.

Basis

For $x = \epsilon$,

$$\begin{aligned}
\widehat{\delta}(\mathbf{0},\epsilon) &= \mathbf{0} && \text{by definition of } \widehat{\delta} \\
&= \#\epsilon && \text{since } \#\epsilon = 0 \\
&= \#\epsilon \bmod 3.
\end{aligned}$$

Induction step

Assuming that (4.3) is true for $x \in \{0,1\}^*$, we show that it is true for xc, where $c \in \{0,1\}$.

$$\begin{aligned}
\widehat{\delta}(\mathbf{0},xc) &= \delta(\widehat{\delta}(\mathbf{0},x),c) && \text{definition of } \widehat{\delta} \\
&= \delta(\#x \bmod 3, c) && \text{induction hypothesis} \\
&= (2(\#x \bmod 3) + c) \bmod 3 && \text{by (4.5)} \\
&= (2(\#x) + c) \bmod 3 && \text{elementary number theory} \\
&= \#xc \bmod 3 && \text{by (4.4).}
\end{aligned}$$

Note that each step has its reason. We used the definition of δ, which is specific to this automaton; the definition of $\widehat{\delta}$ from δ, which is the same for all automata; and elementary properties of numbers and strings.

Exercise 2.2.4: Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

* a) The set of all strings ending in 00.

b) The set of all strings with three consecutive 0's (not necessarily at the end).

c) The set of strings with 011 as a substring.

Exercise 2.2.5: Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

a) The set of all strings such that each block of five consecutive symbols contains at least two 0's.

b) The set of all strings whose tenth symbol from the right end is a 1.

c) The set of strings that either begin or end (or both) with 01.

d) The set of strings such that the number of 0's is divisible by five, and the number of 1's is divisible by 3.

Exercise 2.2.6: Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

* a) The set of all strings beginning with a 1 that, when interpreted as a binary integer, is a multiple of 5. For example, strings 101, 1010, and 1111 are in the language; 0, 100, and 111 are not.

b) The set of all strings that, when interpreted *in reverse* as a binary integer, is divisible by 5. Examples of strings in the language are 0, 10011, 1001100, and 0101.

Exercise 2.4.1: Design NFA's to recognize the following sets of strings.

* a) `abc`, `abd`, and `aacd`. Assume the alphabet is $\{a, b, c, d\}$.

b) `0101`, `101`, and `011`.

c) `ab`, `bc`, and `ca`. Assume the alphabet is $\{a, b, c\}$. Es. Dizionario