Introduction to the Theory of Computation

Jingde Cheng

Saitama University

Introduction to the Theory of Computation

- Enumerability and Diagonalization
- Finite Automata and Regular Languages
- Context-Free Languages
- Computation: Turing Machines
- Computation: Turing-Computability (Turing-Decidability)
- Computation: Reducibility (Turing-Reducibility)
- Computation: Recursive Functions
- Computation: Recursive Sets and Relations
- Equivalent Definitions of Computability
- Advanced Topics in Computability Theory
- Computational Complexity
- Time Complexity
- Space Complexity
- Intractability
- Advanced Topics in Complexity Theory

Automata Theory: What Is It and Why Study It? [HMU-ToC-07]

♣ Automata Theory: What is it?

- Automata theory is the study of abstract computing devices, or "machines".
- In the 1940's and 1950's, a number of researchers studied some kinds of "machines", called "*finite automata*" today, that are simpler than Turing's Turing machines.

♣ Automata Theory: Why study it?

- Finite automata are a useful model for many important kinds of hardware and software.
- There are many systems that may be viewed as being at all times in one of a finite number of "states"; these systems can be represented and analyzed by finite automata.

(Deterministic) Finite Automata (DFAs): An Example [S-ToC-13]

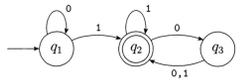


FIGURE **1.4**
A finite automaton called M_1 that has three states

Figure 1.4 is called the ***state diagram*** of M_1. It has three ***states***, labeled q_1, q_2, and q_3. The ***start state***, q_1, is indicated by the arrow pointing at it from nowhere. The ***accept state***, q_2, is the one with a double circle. The arrows going from one state to another are called ***transitions***.

For example, when we feed the input string 1101 to the machine M_1 in Figure 1.4, the processing proceeds as follows.

1. Start in state q_1.
2. Read 1, follow transition from q_1 to q_2.
3. Read 1, follow transition from q_2 to q_2.
4. Read 0, follow transition from q_2 to q_3.
5. Read 1, follow transition from q_3 to q_2.
6. *Accept* because M_1 is in an accept state q_2 at the end of the input.

(Deterministic) Finite Automata (DFAs) as Machines [LP-ToC-98]

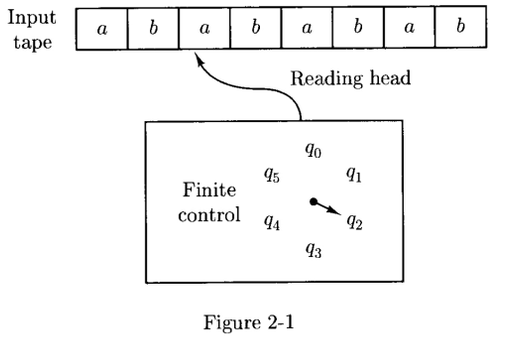


Figure 2-1

(Deterministic) Finite Automata (DFAs) as Machines [LP-ToC-98]

Let us now describe the operation of a finite automaton in more detail. Strings are fed into the device by means of an **input tape**, which is divided into squares, with one symbol inscribed in each tape square (see Figure 2-1). The main part of the machine itself is a "black box" with innards that can be, at any specified moment, in one of a finite number of distinct internal **states**. This black box —called the **finite control**— can sense what symbol is written at any position on the input tape by means of a movable **reading head**. Initially, the reading head is placed at the leftmost square of the tape and the finite control is set in a designated **initial state**. At regular intervals the automaton reads one symbol from the input tape and then enters a new state *that depends only on the current state and the symbol just read* —this is why we shall call this device a *deterministic* finite automaton, to be contrasted to the *nondeterministic* version introduced in the next section. After reading an input symbol, the reading head moves one square to the right on the input tape so that on the next move it will read the symbol in the next tape square. This process is repeated again and again; a symbol is read, the reading head moves to the right, and the state of the finite control changes. Eventually the reading head reaches the end of the input string. The automaton then indicates its approval or disapproval of what it has read by the state it is in at the end: if it winds up in one of a set of **final states** the input string is considered to be **accepted**. The **language accepted** by the machine is the set of strings it accepts.

确定有穷状态自动机

(Deterministic) Finite Automata (DFAs): Formal Definition

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

状态转移函数

- 1 *Transition function* δ is a function from the Cartesian (direct) product ($Q \times \Sigma$) of Q and Σ to Q.
- 2 *Accept states* are also called *final states*.

DFAs: Transition Diagrams and Tables [HMU-ToC-07]

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.

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 $\delta(q, a)$.

DFAs: An Example [S-ToC-13]

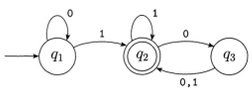


FIGURE 1.6
The finite automaton M_1

We can describe M_1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and
5. $F = \{q_2\}$.

Alphabets, Strings, and Languages

♣ **Alphabet** 字符集

- [D] An *alphabet* is a non-empty finite set of symbols. 非空有穷
- We generally use capital Greek letters Σ and Γ to designate alphabets.

♣ **Examples of alphabet**

- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{a, b, c, \ldots, z\}$
- $\Gamma = \{0, 1, x, y, z\}$

Alphabets, Strings, and Languages

♣ **String** 字符串

- [D] A *string* over an alphabet is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas.
- For a given alphabet Σ, Σ^* denotes all strings over Σ (The reflexive transitive closure of symbol connection relation).

♣ **The length of a string** 字符串长度

- [D] If w is string, the length of w, written $|w|$, is the number of its symbols.
- Note: A symbol may appear in a strings many times.

♣ **The empty string** 空串

- [D] The string of length zero is called *the empty string* and is written ε.

Alphabets, Strings, and Languages

♣ **The reverse of string** 字符串的逆

- [D] For string $w = w_1w_2w_3\ldots w_n$, *the reverse* of w, written w^R, is the string obtained by writing w in the opposite order, i.e., $w^R = w_n\ldots w_3w_2w_1$.

♣ **The concatenation of strings** 字符串的连接

- [D] For string x of length m and string y of length n, *the concatenation* of x and y, written xy, is the string obtained by appending y to the end of x, as in $x_1x_2x_3\ldots x_my_1y_2y_3\ldots y_n$.
- [D] To concatenate a string with itself many time, say k times, we use the superscript notation x^k.

x与自己k次连接

Alphabets, Strings, and Languages

- **Prefix of string** 字符串的前缀
  - **[D] For string y, we say that string x is *a prefix* of y if a string z exists where $xz = y$, and that x is *a proper prefix* of y if in addition $x \neq y$.**
  - **Note: Any string is a prefix of itself.**
- **Language** 语言
  - **[D] For a given alphabet Σ, *a language over Σ*, denoted by L_Σ, is a set of strings over Σ.**
  - **For any Σ, $L_\Sigma \subseteq \Sigma^*$, $L_\Sigma \cup L_\Sigma{}^C = \Sigma^*$, $L_\Sigma \cap L_\Sigma{}^C = \varnothing$.**
  - **[D] A language is *prefix-free* if no member is a proper prefix of another member.**

无前缀语言：没有成员是另一个成员的真前缀

DFAs: Regular Languages

- **Languages of finite automata** 有穷自动机的语言
  - **[D] If A is the set of all strings that finite automaton M accepts, we say that A is *the language of M* and write *$L(M)$* = A, also say that M *recognizes* A or that M *accepts* A.**
- **Notes**
  - **A finite automaton may accept several strings, but it always recognizes only one language.**
  - **If the finite automaton accepts no strings, it still recognizes one language, namely, *the empty language* ∅.**
- **Regular language** 正则语言
  - **[D] A language is called a *regular language* if some finite automaton recognizes it.**

有穷自动机只识别一种语言

存在有穷自动机能识别该语言.

Deterministic Finite Accepter (DFAs): Formal Definition [L-ToC-17]

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**,
$\delta : Q \times \Sigma \to Q$ 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**.

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,

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

DEFINITION 2.3

A language L is called **regular** if and only if there exists some deterministic finite accepter M such that

$$L = L(M).$$

DFA Examples [S-ToC-13]

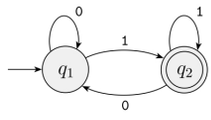


FIGURE 1.8
State diagram of the two-state finite automaton M_2

In the formal description, M_2 is $(\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_2\})$. The transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2.

DFA Examples [S-ToC-13]

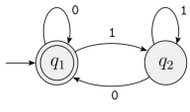


FIGURE 1.10
State diagram of the two-state finite automaton M_3

Machine M_3 is similar to M_2 except for the location of the accept state. As usual, the machine accepts all strings that leave it in an accept state when it has finished reading. Note that because the start state is also an accept state, M_3 accepts the empty string ε. As soon as a machine begins reading the empty string, it is at the end; so if the start state is an accept state, ε is accepted. In addition to the empty string, this machine accepts any string ending with a 0. Here,

$$L(M_3) = \{w | \ w \text{ is the empty string } \varepsilon \text{ or ends in a } 0\}.$$

若输入为空串 / 由0结尾的串，都能停止

DFA Examples [S-ToC-13]

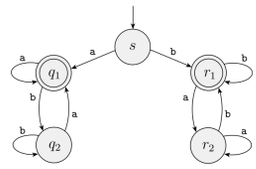


FIGURE 1.12
Finite automaton M_4

Machine M_4 has two accept states, q_1 and r_1, and operates over the alphabet $\Sigma = \{a, b\}$. Some experimentation shows that it accepts strings a, b, aa, bb, and bab, but not strings ab, ba, or bbba. This machine begins in state s, and after it reads the first symbol in the input, it goes either left into the q states or right into the r states. In both cases, it can never return to the start state (in contrast to the previous examples), as it has no way to get from any other state back to s. If the first symbol in the input string is a, then it goes left and accepts when the string ends with an a. Similarly, if the first symbol is a b, the machine goes right and accepts when the string ends in b. So M_4 accepts all strings that start and end with a or that start and end with b. In other words, M_4 accepts strings that start and end with the same symbol.

最后q_0接受的一定是3的倍数

DFA Examples [S-ToC-13]

Figure 1.14 shows the three-state machine M_5, which has a four-symbol input alphabet, $\Sigma = \{\langle \text{RESET} \rangle, 0, 1, 2\}$. We treat $\langle \text{RESET} \rangle$ as a single symbol.

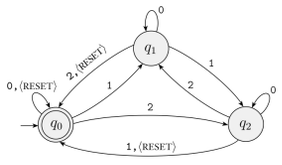


FIGURE 1.14
Finite automaton M_5

Machine M_5 keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives the ⟨RESET⟩ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3.

DFA Examples [L-ToC-17]

The graph in Figure 2.1 represents the dfa

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

where δ is given by

$$\begin{aligned} \delta(q_0, 0) &= q_0, & \delta(q_0, 1) &= q_1, \\ \delta(q_1, 0) &= q_0, & \delta(q_1, 1) &= q_2, \\ \delta(q_2, 0) &= q_2, & \delta(q_2, 1) &= q_1. \end{aligned}$$

This dfa accepts the string 01. Starting in state q_0, the symbol 0 is read first. Looking at the edges of the graph, we see that the automaton remains in state q_0. Next, the 1 is read and the automaton goes into state q_1. We are now at the end of the string and, at the same time, in a final state q_1. Therefore, the string 01 is accepted. The dfa does not accept the string 00, since after reading two consecutive 0's, it will be in state q_0. By similar reasoning, we see that the automaton will accept the strings 101, 0111, and 11001, but not 100 or 1100.

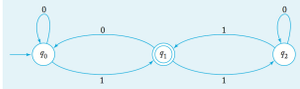


FIGURE 2.1

DFA Examples [L-ToC-17]

Consider the dfa in Figure 2.2.

In drawing Figure 2.2 we allowed the use of two labels on a single edge. Such multiply labeled edges are shorthand for two or more distinct transitions: The transition is taken whenever the input symbol matches any of the edge labels.

The automaton in Figure 2.2 remains in its initial state q_0 until the first b is encountered. If this is also the last symbol of the input, then the string is accepted. If not, the dfa goes into state q_2, from which it can never escape. The state q_2 is a **trap state**. We see clearly from the graph that the automaton accepts all strings consisting of an arbitrary number of a's, followed by a single b. All other input strings are rejected. In set notation, the language accepted by the automaton is

$$L = \{a^n b : n \geq 0\}.$$

FIGURE 2.2

	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

FIGURE 2.3

DFA Examples [L-ToC-17]

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

The only issue here is the first two symbols in the string; after they have been read, no further decisions are needed. Still, the automaton has to process the whole string before its decision is made. We can therefore solve the problem with an automaton that has four states: an initial state, two states for recognizing ab ending in a final trap state, and one nonfinal trap state. If the first symbol is an a and the second is a b, the automaton goes to the final trap state, where it will stay since the rest of the input does not matter. On the other hand, if the first symbol is not an a or the second one is not a b, the automaton enters the nonfinal trap state. The simple solution is shown in Figure 2.4.

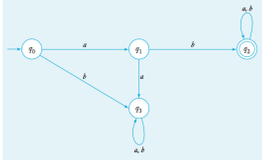


FIGURE 2.4

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

In deciding whether the substring 001 has occurred, we need to know not only the current input symbol, but we also need to remember whether or not it has been preceded by one or two 0's. We can keep track of this by putting the automaton into specific states and labeling them accordingly. Like variable names in a programming language, state names are arbitrary and can be chosen for mnemonic reasons. For example, the state in which two 0's were the immediately preceding symbols can be labeled simply 00.

If the string starts with 001, then it must be rejected. This implies that there must be a path labeled 001 from the initial state to a nonfinal state. For convenience, this nonfinal state is labeled 001. This state must be a trap state, because later symbols do not matter. All other states are accepting states.

This gives us the basic structure of the solution, but we still must add provisions for the substring 001 occurring in the middle of the input. We must define Q and δ so that whatever we need to make the correct decision is remembered by the automaton. In this case, when a symbol is read, we need to know some part of the string to the left, for example, whether or not the two previous symbols were 00. If we label the states with the relevant symbols, it is very easy to see what the transitions must be. For example,

$$\delta(00, 0) = 00$$

because this situation arises only if there are three consecutive 0's. We are only interested in the last two, a fact we remember by keeping the dfa in the state 00. A complete solution is shown in Figure 2.5. We see from this example how useful mnemonic labels on the states are for keeping track of things. Trace a few strings, such as 100100 and 1010100, to see that the solution is indeed correct.

FIGURE 2.5

DFA Examples [L-ToC-17]

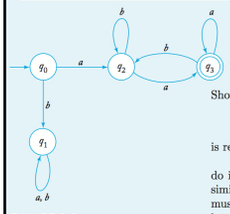


FIGURE 2.6

Show that the language

$$L = \{awa : w \in \{a, b\}^*\}$$

is regular.

To show that this or any other language is regular, all we have to do is find a dfa for it. The construction of a dfa for this language is similar to Example 2.3, but a little more complicated. What this dfa must do is check whether a string begins and ends with an a; what is between is immaterial. The solution is complicated by the fact that there is no explicit way of testing the end of the string. This difficulty is overcome by simply putting the dfa into a final state whenever the second a is encountered. If this is not the end of the string, and another b is found, it will take the dfa out of the final state. Scanning continues in this way, each a taking the automaton back to its final state. The complete solution is shown in Figure 2.6. Again, trace a few examples to see why this works. After one or two tests, it will be obvious that the dfa accepts a string if and only if it begins and ends with an a. Since we have constructed a dfa for the language, we can claim that, by definition, the language is regular.

DFA Examples [L-ToC-17]

Let L be the language in Example 2.5. Show that L^2 is regular. Again we show that the language is regular by constructing a dfa for it. We can write an explicit expression for L^2, namely,

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

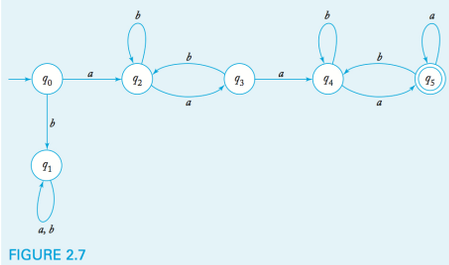


FIGURE 2.7

DFAs: Formal Definition of Computation

由有穷自动机的计算

♣ **Formal definition of computation by a finite automata**

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

开始为初始态，中间符合状态转移，结束为接受态。

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w \mid M \text{ accepts } w\}$.

♣ **Regular language**

- **[D] A language is called a *regular language* if some finite automaton recognizes it.**

DFA Examples [S-ToC-13]

Figure 1.14 shows the three-state machine M_5, which has a four-symbol input alphabet, $\Sigma = \{\langle\text{RESET}\rangle, 0, 1, 2\}$. We treat $\langle\text{RESET}\rangle$ as a single symbol.

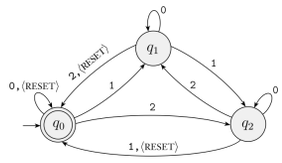


FIGURE **1.14**
Finite automaton M_5

Machine M_5 keeps a running count of the sum of the numerical input symbols it reads, modulo 3. Every time it receives the $\langle\text{RESET}\rangle$ symbol, it resets the count to 0. It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3.

DFAs: Formal Definition of Computation

♣ **Example: Machine M_5**

- **Let *w* be the string 10⟨RESET⟩22⟨RESET⟩012.**
- **Then M_5 accepts *w* according to the formal definition of computation because the sequence of states it enters when computing on *w* is**
  $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$,
  which satisfies the three conditions.
- **The language of M_5 is**
  $L(M_5) = \{w \mid$ the sum of the symbols in *w* is 0 modulo 3, except that ⟨RESET⟩ resets the count to 0$\}$.
- **As M_5 recognizes this language, it is a regular language.**

Designing DFAs: An Example [S-ToC-13]

- **Suppose that the alphabet is {0, 1} and that the language consists of all strings with an odd number of 1s.**
- **Let us construct a finite automaton E_1 to recognize this language.**
- **We start getting an input string of 0s and 1s symbol by symbol.**
- **We do not need to remember the entire string seen so far in order to determine whether the number of 1s is odd.**
- **We just need to remember whether the number (answer) of 1s seen so far is even or odd and keep track of this information as we read new symbols.**
- **If we read a 1, flip the answer (even to odd, or odd to even); but if we read a 0, leave the answer as is.**

Designing DFAs: An Example [S-ToC-13]

- **Once we have determined the necessary information to remember about the string as it is being read, we represent this information as a finite list of possibilities.**
- **In this instance, the possibilities would be:**
  1. even so far, and
  2. odd so far.

FIGURE **1.18**
The two states q_{even} and q_{odd}

- **Next, we assign the transitions by seeing how to go from one possibility to another upon reading a symbol.**
- **So, if state q_{even} represents the even possibility and state q_{odd} represents the odd possibility, we would set the transitions to flip state on a 1 and stay put on a 0.**

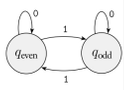
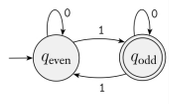


FIGURE **1.19**
Transitions telling how the possibilities rearrange

Designing DFAs: An Example [S-ToC-13]

- **Next, we set the start state to be the state corresponding to the possibility associated with having seen 0 symbols so far (the empty string ε).**
- **In this case, the start state corresponds to state q_{even} because 0 is an even number.**
- **Last, set the accept states to be those corresponding to possibilities where we want to accept the input string.**
- **Set q_{odd} to be an accept state because we want to accept when we have seen an odd number of 1s.**



- **These additions are shown in the figure.**

FIGURE **1.20**
Adding the start and accept states

Designing DFAs: An Example [S-ToC-13]

EXAMPLE **1.21**

This example shows how to design a finite automaton E_2 to recognize the regular language of all strings that contain the string 001 as a substring. For example, 0010, 1001, 001, and 11111110011111 are all in the language, but 11 and 0000 are not. How would you recognize this language if you were pretending to be E_2? As symbols come in, you would initially skip over all 1s. If you come to a 0, then you note that you may have just seen the first of the three symbols in the pattern 001 you are seeking. If at this point you see a 1, there were too few 0s, so you go back to skipping over 1s. But if you see a 0 at that point, you should remember that you have just seen two symbols of the pattern. Now you simply need to continue scanning until you see a 1. If you find it, remember that you succeeded in finding the pattern and continue reading the input string until you get to the end.

So there are four possibilities: You

1. haven't just seen any symbols of the pattern,
2. have just seen a 0,
3. have just seen 00, or
4. have seen the entire pattern 001.

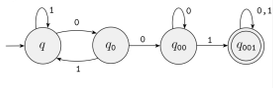


FIGURE **1.22**
Accepts strings containing 001

Assign the states q, q_0, q_{00}, and q_{001} to these possibilities. You can assign the transitions by observing that from q reading a 1 you stay in q, but reading a 0 you move to q_0. In q_0 reading a 1 you return to q, but reading a 0 you move to q_{00}. In q_{00} reading a 1 you move to q_{001}, but reading a 0 leaves you in q_{00}. Finally, in q_{001} reading a 0 or a 1 leaves you in q_{001}. The start state is q, and the only accept state is q_{001}, as shown in Figure 1.22.

The Regular Operations: Formal Definition

♣ **Regular Operations** 正则运算

DEFINITION **1.23**

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

- **Union**: $A \cup B = \{x | \ x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | \ x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k | \ k \geq 0 \text{ and each } x_i \in A\}$.

♣ **Notes**

- **The concatenation operation attaches a string from A in front of a string from B in all possible ways to get the strings in the new language.**
- **The star operation works by attaching any number of strings in A together to get a string in the new language. The *empty string* ε is always a member of A^*, no matter what A is.**

空串永远是A^*的成员

The Regular Operations: Theorems

♣ **Closed under operation** 运算下的封闭

- **[D] Generally speaking, a collection of objects is *closed* under some operation if applying that operation to members of the collection returns an object still in the collection.**

♣ **Three theorems**

正则语言在上面所说的三个运算下是封闭的

THEOREM **1.25**

The class of regular languages is closed under the union operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

THEOREM **1.26**

The class of regular languages is closed under the concatenation operation.

In other words, if A_1 and A_2 are regular languages then so is $A_1 \circ A_2$.

THEOREM **1.49**

The class of regular languages is closed under the star operation.

Non-deterministic Finite Automata (NFAs): An Example [S-ToC-13]

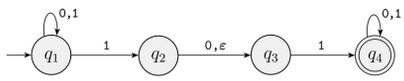


FIGURE **1.27**
The nondeterministic finite automaton N_1

The difference between a deterministic finite automaton, abbreviated DFA, and a nondeterministic finite automaton, abbreviated NFA, is immediately apparent. First, every state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet. The NFA shown in Figure 1.27 violates that rule. State q_1 has one exiting arrow for 0, but it has two for 1; q_2 has one arrow for 0, but it has none for 1. In an NFA, a state may have zero, one, or many exiting arrows for each alphabet symbol.

Second, in a DFA, labels on the transition arrows are symbols from the alphabet. This NFA has an arrow with the label ε. In general, an NFA may have arrows labeled with members of the alphabet or ε. Zero, one, or many arrows may exit from each state with the label ε.

Non-deterministic Finite Automata (NFAs): An Example [S-ToC-13]

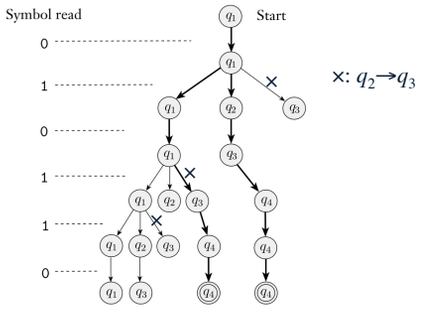


FIGURE **1.29**
The computation of N_1 on input 010110

非确定有穷自动机

Non-deterministic Finite Automata (NFAs): Formal Definition

DEFINITION **1.37**

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

下一个状态是一个集合

- **For any alphabet Σ we write Σ_ε to be $\Sigma \cup \{\varepsilon\}$.**
- **$P(Q)$ (2^Q) is the power set of Q.**
- **Important fact: Every NFA can be converted into an equivalent DFA.**

每个NFA都有等价的DFA

Non-deterministic Finite Accepters (NFAs): Formal Definition [L-ToC-17]

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, F are defined as for deterministic finite accepters, but

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

- **λ is the empty string, 2^Q is the power set of Q.**

DEFINITION 2.5

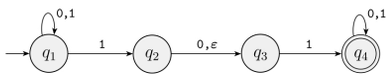
For an nfa, the extended transition function is defined so that $\delta^*(q_i, w)$ contains q_j if and only if there is a walk in the transition graph from q_i to q_j labeled w. This holds for all $q_i, q_j \in Q$, and $w \in \Sigma^*$.

NFA vs. DFA: Differences between Their Definitions

- **Multiple next states** 多个下一状态
  - **In an NFA, the range of transition function δ is the power set $P(Q)$ (2^Q), so that its value is not a single element of Q, but a subset of it. This subset defines the set of all possible states that can be reached by a transition.**
- **Allowing the empty string as the second argument of δ** 允许空串作为转移函数的第二个参数
  - **The empty string ε is allowed as the second argument of transition function δ. This means that the NFA can make a situation without consuming an input symbol.**
- **No next state** 可以没有下一状态
  - **In an NFA, the set $\delta(q_i, a)$ may be empty, meaning that there is no transition defined for this specific situation.**

NFAs: An Example [S-ToC-13]

Recall the NFA N_1:



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

NFA Examples [S-ToC-13]

Let A be the language consisting of all strings over {0,1} containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA N_2 recognizes A.

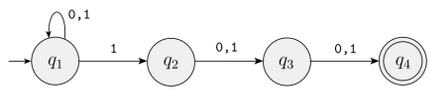


FIGURE 1.31
The NFA N_2 recognizing A

One good way to view the computation of this NFA is to say that it stays in the start state q_1 until it "guesses" that it is three places from the end. At that point, if the input symbol is a 1, it branches to state q_2 and uses q_3 and q_4 to "check" on whether its guess was correct.

NFA Examples [S-ToC-13]

As mentioned, every NFA can be converted into an equivalent DFA; but sometimes that DFA may have many more states. The smallest DFA for A contains eight states. Furthermore, understanding the functioning of the NFA is much easier, as you may see by examining the following figure for the DFA.

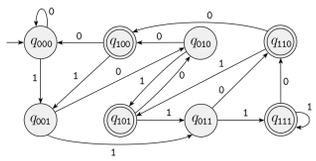


FIGURE 1.32
A DFA recognizing A

Suppose that we added ε to the labels on the arrows going from q_2 to q_3 and from q_3 to q_4 in machine N_2 in Figure 1.31. So both arrows would then have the label 0, 1, ε instead of just 0, 1. What language would N_2 recognize with this modification? Try modifying the DFA in Figure 1.32 to recognize that language.

NFA Examples [S-ToC-13]

The following NFA N_3 has an input alphabet {0} consisting of a single symbol. An alphabet containing only one symbol is called a ***unary alphabet***.

只有一个字符的字符集称为一元集

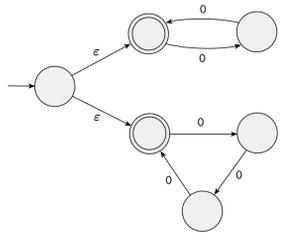


FIGURE 1.34
The NFA N_3

NFA Examples [S-ToC-13]

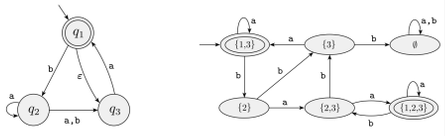


FIGURE 1.36
The NFA N_4

FIGURE 1.44
DFA D after removing unnecessary states

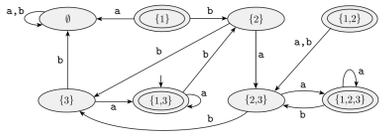


FIGURE 1.43
A DFA D that is equivalent to the NFA N_4

NFA Examples [L-ToC-17]

Consider the transition graph in Figure 2.8. It describes a nondeterministic accepter since there are two transitions labeled a out of q_0.

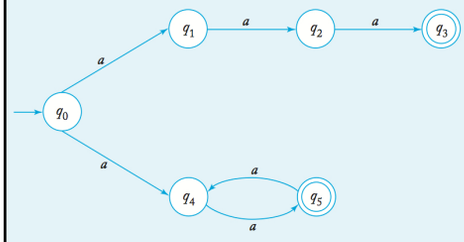


FIGURE 2.8

NFA Examples [L-ToC-17]

A nondeterministic automaton is shown in Figure 2.9. It is nondeterministic not only because several edges with the same label originate from one vertex, but also because it has a λ-transition. Some transitions, such as $\delta(q_2, 0)$, are unspecified in the graph. This is to be interpreted as a transition to the empty set, that is, $\delta(q_2, 0) = \varnothing$. The automaton accepts strings λ, 1010, and 101010, but not 110 and 10100. Note that for 10 there are two alternative walks, one leading to q_0, the other to q_2. Even though q_2 is not a final state, the string is accepted because one walk leads to a final state.

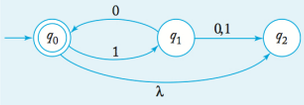


FIGURE 2.9

NFA Examples [L-ToC-17]

Figure 2.10 represents an nfa. It has several λ-transitions and some undefined transitions such as $\delta(q_2, a)$.

Suppose we want to find $\delta^*(q_1, a)$ and $\delta^*(q_2, \lambda)$. There is a walk labeled a involving two λ-transitions from q_1 to itself. By using some of the λ-edges twice, we see that there are also walks involving λ-transitions to q_0 and q_2. Thus,

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

Since there is a λ-edge between q_2 and q_0, we have immediately that $\delta^*(q_2, \lambda)$ contains q_0. Also, since any state can be reached from itself by making no move, and consequently using no input symbol, $\delta^*(q_2, \lambda)$ also contains q_2. Therefore,

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

Using as many λ-transitions as needed, you can also check that

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

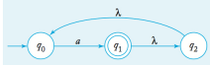


FIGURE 2.10

Non-deterministic Finite Accepters (NFAs): Formal Definition [L-ToC-17]

DEFINITION 2.6

The language L accepted by an nfa $M = (Q, \Sigma, \delta, q_0, F)$ is defined as the set of all strings accepted in the above sense. Formally,

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

In words, the language consists of all strings w for which there is a walk labeled w from the initial vertex of the transition graph to some final vertex.

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 both accept the same language.

THEOREM 2.2

Let L be the language accepted by a nondeterministic finite accepter $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Then there exists a deterministic finite accepter $M_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that

$$L = L(M_D).$$

等价的NFA与DFA

Equivalence of NFAs and DFAs

- ♣ **Equivalence**
  - ◆ **[D] We say that two FAs are *equivalent* if they recognize the same language.**
  - ◆ **Important fact: DFAs and NFAs recognize the same class of languages.**
- ♣ **Equivalence theorem**

THEOREM 1.39

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

COROLLARY 1.40

A language is regular if and only if some nondeterministic finite automaton recognizes it.

语言是正则的 ⇔ 存在NFA能识别该语言。

Equivalence of NFAs and DFAs [L-ToC-17]

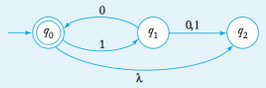


FIGURE 2.9

The dfa shown in Figure 2.11 is equivalent to the nfa in Figure 2.9 since they both accept the language $\{(10)^n : n \geq 0\}$.

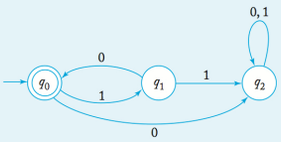


FIGURE 2.11

Equivalence of NFAs and DFAs [L-ToC-17]

Convert the nfa in Figure 2.12 to an equivalent dfa. The nfa starts in state q_0, so the initial state of the dfa will be labeled $\{q_0\}$. After reading an a, the nfa can be in state q_1 or, by making a λ-transition, in state q_2. Therefore, the corresponding dfa must have a state labeled $\{q_1, q_2\}$ and a transition

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

In state q_0, the nfa has no specified transition when the input is b; therefore,

$$\delta(\{q_0\}, b) = \varnothing.$$

A state labeled $\varnothing$ represents an impossible move for the nfa and, therefore, means nonacceptance of the string. Consequently, this state in the dfa must be a nonfinal trap state.

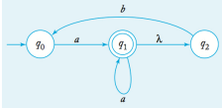


FIGURE 2.12

Equivalence of NFAs and DFAs [L-ToC-17]

We have now introduced into the dfa the state $\{q_1, q_2\}$, so we need to find the transitions out of this state. Remember that this state of the dfa corresponds to two possible states of the nfa, so we must refer back to the nfa. If the nfa is in state q_1 and reads an a, it can go to q_1. Furthermore, from q_1 the nfa can make a λ-transition to q_2. If, for the same input, the nfa is in state q_2, then there is no specified transition. Therefore,

$$\delta(\{q_1, q_2\}, a) = \{q_1, q_2\}.$$

Similarly,

$$\delta(\{q_1, q_2\}, b) = \{q_0\}.$$

At this point, every state has all transitions defined. The result, shown in Figure 2.13, is a dfa, equivalent to the nfa with which we started. The nfa in Figure 2.12 accepts any string for which $\delta^*(q_0, w)$ contains q_1. For the corresponding dfa to accept every such w, any state whose label includes q_1 must be made a final state.

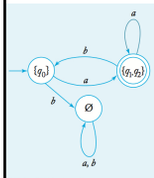


FIGURE 2.13

NFA Application Example: Text Search [HMU-ToC-07]

Example 2.14: Suppose we want to design an NFA to recognize occurrences of the words **web** and **ebay**. The transition diagram for the NFA designed using the rules above is in Fig. 2.16. State 1 is the start state, and we use Σ to stand for the set of all printable ASCII characters. States 2 through 4 have the job of recognizing **web**, while states 5 through 8 recognize **ebay**. □

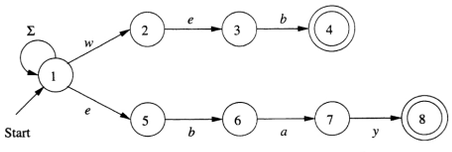


Figure 2.16: An NFA that searches for the words **web** and **ebay**

DFA Application Example: Text Search [HMU-ToC-07]

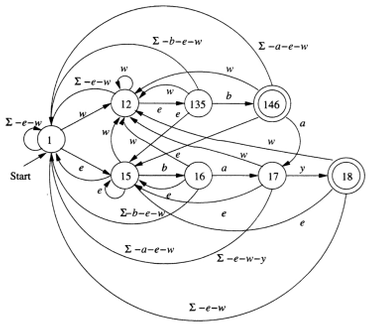


Figure 2.17: Conversion of the NFA from Fig. 2.16 to a DFA

NFA Application Example: Uses of ε-Transition [HMU-ToC-07]

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.

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

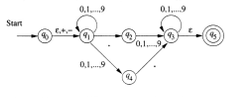


Figure 2.18: An ϵ-NFA accepting decimal numbers

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. □

DFA Application Example: Eliminating ε-Transition [HMU-ToC-07]

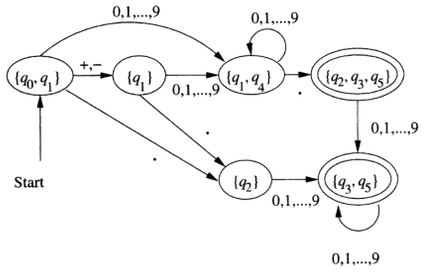


Figure 2.22: The DFA D that eliminates ϵ-transitions from Fig. 2.18

NFA Application Example: Uses of ε-Transition [HMU-ToC-07]

Example 2.17: The strategy we outlined in Example 2.14 for building an NFA that recognizes a set of keywords can be simplified further if we allow ϵ-transitions. For instance, the NFA recognizing the keywords `web` and `ebay`, which we saw in Fig. 2.16, can also be implemented with ϵ-transitions as in Fig. 2.19. In general, we construct a complete sequence of states for each keyword, as if it were the only word the automaton needed to recognize. Then, we add a new start state (state 9 in Fig. 2.19), with ϵ-transitions to the start-states of the automata for each of the keywords. □

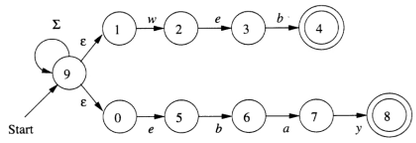


Figure 2.19: Using ϵ-transitions to help recognize keywords