Theory of Computing 2023: Regular Languages

(Based on [Sipser 2006, 2013])

Yih-Kuen Tsay

March 6, 2023

1 Finite Automata

Finite Automata

- *What is a computer?*
- Real computers are complicated.
- To set up a manageable mathematical theory of computers, we use an idealized computer called a *computational model*.
- The *finite automaton* (finite-state machine) is the simplest of such models.
- It represents a computer with an extremely limited amount of memory.

Finite Automata (cont.)

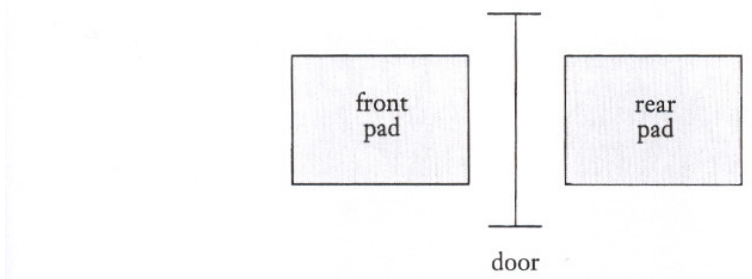


FIGURE 1.1
Top view of an automatic door

Source: [Sipser 2006]

Finite Automata (cont.)

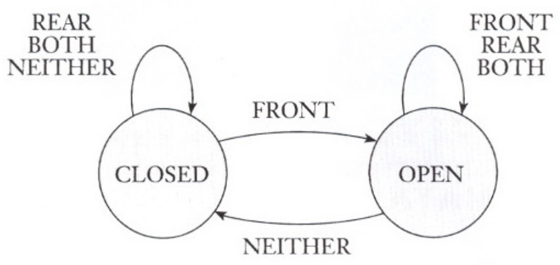
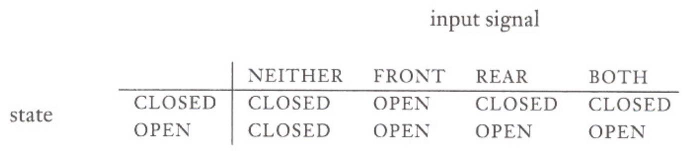


FIGURE 1.2
State diagram for automatic door controller

Source: [Sipser 2006]

Finite Automata (cont.)



		input signal			
		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

FIGURE 1.3
State transition table for automatic door controller

Source: [Sipser 2006]

Finite Automata (cont.)

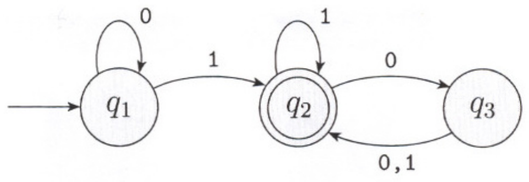


FIGURE 1.4
A finite automaton called M_1 that has three states

Source: [Sipser 2006]

Formal Definition

- Though state diagrams are easier to grasp intuitively, we need the formal definition, too.
- A formal definition is precise so as to resolve any uncertainties about what is allowed in a finite automaton.
- It also provides notation for concise and clear expression.

Definition 1 (1.5)**.** A *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 set of symbols (the *alphabet*),
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start* state, and
5. $F \subseteq Q$ is the set of *accept* states.

Formal Definition (cont.)

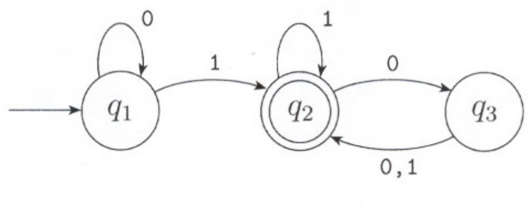


FIGURE 1.6
The finite automaton M_1

Source: [Sipser 2006]

A machine *accepts* a string if the machine stops at an accept state after processing/reading the string symbol by symbol. For instance, M_1 accepts 011 and 010100.

Definition of M_1

Formally, $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

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

Language Recognizers

- Let A be the set of all strings that a machine M accepts.
- We say that A is the *language of machine* M and write $L(M) = A$.
- We also say that *M recognizes A* (or that M accepts A).
- A machine is said to accept the empty language $\emptyset$ if it accepts no strings.
- Regarding the example automaton M_1,
  $L(M_1) = \{w \mid w$ contains at least one 1 and an even number of 0s follow the last 1$\}$.

Language Recognizers (cont.)

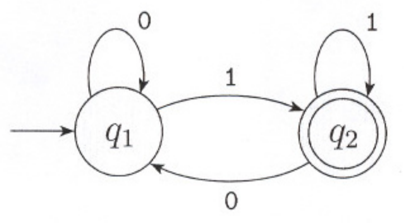


FIGURE 1.8
State diagram of the two-state finite automaton M_2

Source: [Sipser 2006]

Note: $L(M_2) = \{w \mid w \text{ ends in a } 1\}$

Language Recognizers (cont.)

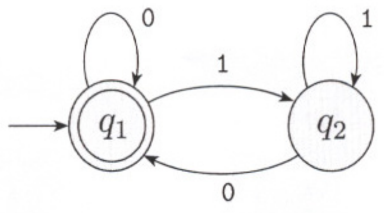


FIGURE 1.10
State diagram of the two-state finite automaton M_3

Source: [Sipser 2006]

Note: $L(M_3) = \{w \mid w \text{ is the empty string or ends in a } 0\}$

Language Recognizers (cont.)

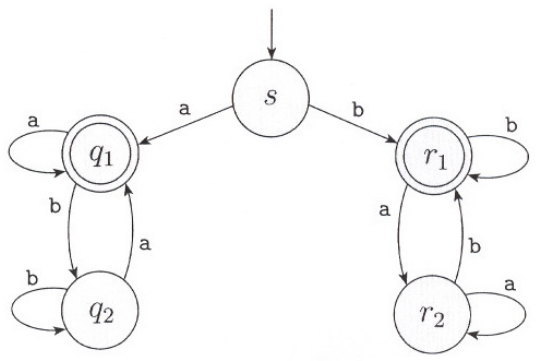


FIGURE 1.12
Finite automaton M_4

Source: [Sipser 2006]

Note: M_4 accepts strings that start and end with the same symbol.

Language Recognizers (cont.)

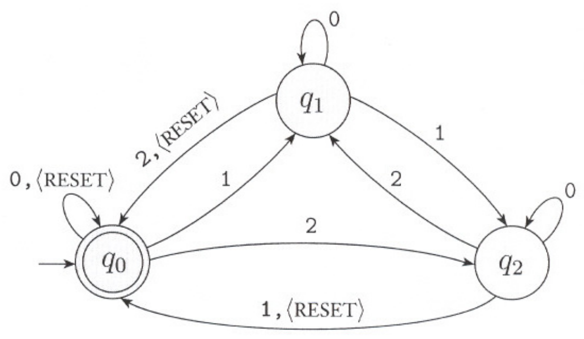


FIGURE 1.14
Finite automaton M_5

Source: [Sipser 2006]

Formal Definition of Computation

We already have an informal idea of how a machine computes, i.e., how a machine accepts or rejects a string. Below is a formalization.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 \dots w_n$ be a string over Σ.
- We say that M *accepts* $w =$ if a sequence of states $r_0, r_1, \dots, r_n$ exists such that
  1. $r_0 = q_0$,
  2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, 1, \dots, n-1$, and
  3. $r_n \in F$.

Regular Languages

Definition 2 (1.16)**.** A language is called a *regular language* if some finite automaton recognizes it.

- There are a few alternatives for defining regular languages.
- We will see some of them and show that they are all equivalent.

Designing Finite Automata

The "reader as automaton" method:

1. Determine the necessary information needed to be remembered about the string as it is being read.
2. Represent the information as a finite list of possibilities and assign a state to each of the possibilities.
3. Assign the transitions by seeing how to go from one possibility to another upon reading a symbol.
4. Set the start state to be the state corresponding to the possibility associated with having seen 0 symbols so far.
5. Set the accept states to be those corresponding to possibilities where you want to accept the input read so far.

Designing Finite Automata (cont.)

Consider constructing an automaton that recognizes binary strings with an odd number of 1's.

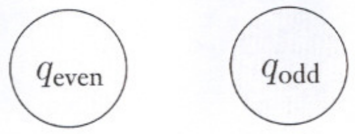


FIGURE 1.18
The two states q_{even} and q_{odd}

Source: [Sipser 2006]

Designing Finite Automata (cont.)

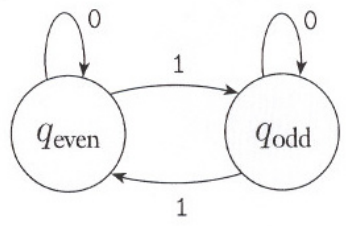


FIGURE 1.19
Transitions telling how the possibilities rearrange

Source: [Sipser 2006]

Designing Finite Automata (cont.)

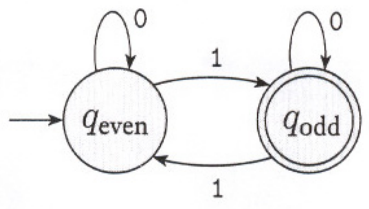


FIGURE 1.20
Adding the start and accept states

Source: [Sipser 2006]

Designing Finite Automata (cont.)

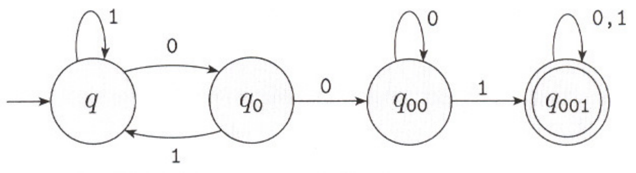

$$\begin{tikzcd}
\arrow[r] & q \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_0 \arrow[l, bend left, "1"] \arrow[r, "0"] & q_{00} \arrow[loop above, "0"] \arrow[r, "1"] & q_{001} \arrow[loop above, "{0,1}"]
\end{tikzcd}$$

FIGURE **1.22**
Accepts strings containing 001

Source: [Sipser 2006]

2 The Regular Operations

The Regular Operations

- In arithmetic, the basic objects are numbers and the tools for manipulating them are operations such as $+$ and $\times$.
- In the theory of computation the objects are languages and the tools include operations specifically designed for manipulating them. We consider three operations called regular operations.

  Definition 3 (1.23)**.** Let A and B be languages. The three *regular operations* are defined as follows:

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

- We will use these operations to study the properties of finite automata.

Closedness

- A collection of objects is *closed* under some operation if applying the operation to members of the collection returns an object still in the collection.
- We will show that the collection of regular languages is closed under all three regular operations.

  /* The collection of regular languages is also closed under the three fundamental set operations: union, intersection, and complementation, which is relatively easy to prove. */

Closedness under Union

Theorem 4 (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 $A_1 \cup A_2$.*

- The proof is by construction. To prove that $A_1 \cup A_2$ is regular, we construct a finite automaton M that recognizes $A_1 \cup A_2$.
- Suppose that a finite automaton M_1 recognizes A_1 and another M_2 recognizes A_2.
- Machine M works by *simulating* both M_1 and M_2 and accepting if either simulation accepts.
- As the input symbols arrive one by one, M remembers the state that each machine would be in if it had read up to this point.

Closedness under Union (cont.)

Theorem 5 (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 $A_1 \cup A_2$.*

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1 and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2.
- Construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:
    1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
    2. Σ is the same. (Generalization is possible.)
    3. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
    4. $q_0 = (q_1, q_2)$.
    5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

/* This proof can be easily adapted to prove that the collection of regular languages is also closed under intersection. Do you see how? */

Closedness under Concatenation

Theorem 6 (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, so is $A_1 \circ A_2$.*

- Proof by construction along the lines of the proof for closedness under union does not work in this case.
- Suppose A_1 is the set of binary strings containing 001, while A_2 is the set of binary strings with an odd number of 1's.
    - The binary string 0010011 is in $A_1 \circ A_2$.
    - How can a machine, simulating M_1 and then M_2, knows that it should not stop M_1 and move to M_2 after seeing the first occurrence of 001?
- We resort to a new technique called *nondeterminism*.

3 Nondeterminism

Nondeterminism

- In a *nondeterministic* machine, several choices may exist for the next state after reading the next input symbol in a given state.
- The difference between a deterministic finite automaton (DFA) and a nondeterministic finite automaton (NFA):

	# of next states (per symbol)	input symbols
DFA	1	from Σ
NFA	0, 1, or more	from $\Sigma \cup \{\varepsilon\}$

Nondeterminism (cont.)

- Nondeterminism is a useful concept that has had great impact on computation theory.
- As we will show, *every NFA can be converted into an equivalent DFA*.
- However, constructing NFAs is sometimes easier than directly constructing DFAs. An NFA may be much smaller than its deterministic counterpart, or its functioning may be easier to understand.

Nondeterminism (cont.)

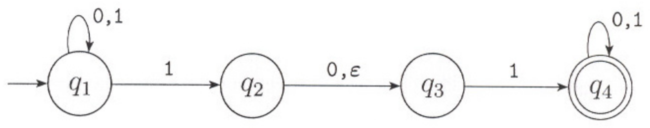


FIGURE 1.27
The nondeterministic finite automaton N_1

Source: [Sipser 2006]

Note: N_1 accepts all strings that contain either 101 or 11 as a substring.

How Does an NFA Compute?

1. If there are multiple choices for the next state, given the next input symbol, the machine splits into multiple copies, all moving to their respective next states in parallel.
2. Additional copies are also created if there are exiting arrows labeled with ε, one copy for each of such arrows. All copies move to their respective next states in parallel, but without consuming any input.
3. If *any* copy is in an accept state at the end of the input, the machine accepts the input string.
4. If there are input symbols remaining, the preceding steps are repeated.

Deterministic vs. Nondeterministic Comp.

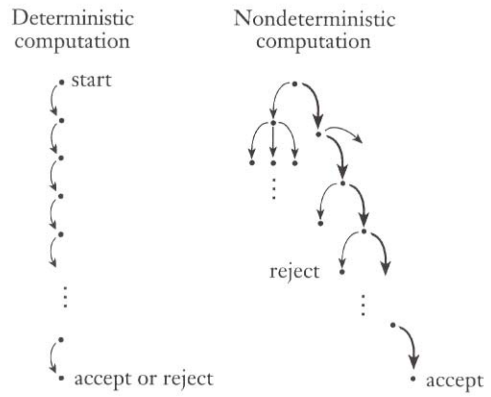


FIGURE 1.28
Deterministic and nondeterministic computations with an accepting branch

Source: [Sipser 2006]

A Computation of N_1

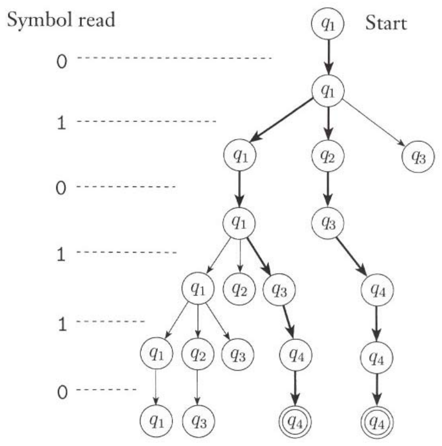


FIGURE 1.29
The computation of N_1 on input 010110

Source: [Sipser 2006]

Example NFA

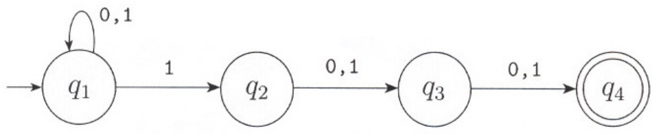


FIGURE 1.31
The NFA N_2 recognizing A

Source: [Sipser 2006]

Note: A is the set of all strings over $\{0, 1\}$ containing a 1 in the last third position.

Example NFA (cont.)

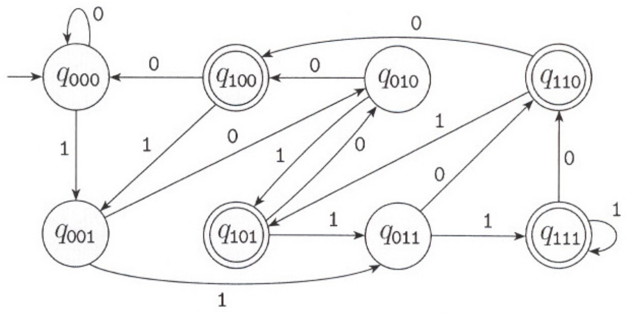


FIGURE 1.32
A DFA recognizing A

Source: [Sipser 2006]

Example NFA (cont.)

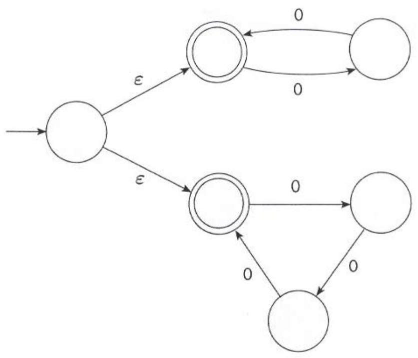


FIGURE 1.34
The NFA N_3

Source: [Sipser 2006]

Note: N_3 accepts all strings of the form 0^k where k is a multiple of 2 or 3.

Example NFA (cont.)

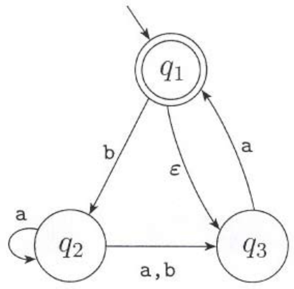


FIGURE 1.36
The NFA N_4

Source: [Sipser 2006]

Does N_4 accept ε? How about `babaa`?

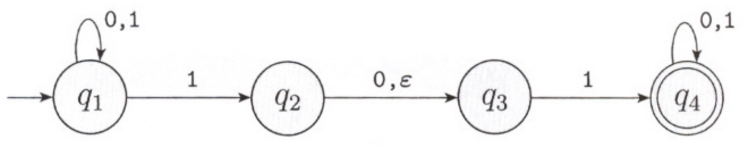
Definition of an NFA

- The transition function of an NFA takes a state and an input symbol *or the empty string* and produces *a set of possible next states*.
- Let $\mathcal{P}(Q)$ be the power set of Q and let Σ_ε denote $\Sigma \cup \{\varepsilon\}$.

Definition 7 (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.

Definition of an NFA (cont.)



Source: [Sipser 2006]

Definition of N_1

Formally, $N_1 = (Q, \Sigma, \delta, q_1, F)$, 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. $F = \{q_4\}$.

Formal Def. of Nondeterministic Comp.

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w be a string over Σ.
- We say that N *accepts* w if we can write $w = y_1 y_2 \ldots y_m$, where $y_i \in \Sigma_\varepsilon$, and a sequence of states $r_0, r_1, \ldots, r_m$ exists such that
  1. $r_0 = q_0$,
  2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, 1, \ldots, m-1$, and
  3. $r_m \in F$.

Equivalence of NFA and DFA

Two machines are *equivalent* if they recognize the same language.

Theorem 8 (1.39)**.** *Every nondeterministic finite automaton has an equivalent deterministic finite automaton.*

Corollary 9 (1.40)**.** *A language is regular if and only if some nondeterministic finite automaton recognizes it.*

Equivalence of NFA and DFA (cont.)

Theorem 10 (1.39)**.** *Every NFA has an equivalent DFA.*

- The idea is to convert a given NFA into an equivalent DFA that *simulates* the NFA.
- An NFA can be in one of several possible states, as it reads the input.
- If k is the number of states of the NFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the simulating DFA must remember.

Equivalence of NFA and DFA (cont.)

Theorem 11 (1.39)**.** *Every NFA has an equivalent DFA.*

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing some language A.
- Construct $M = (Q', \Sigma, \delta', q_0', F')$ to recognize A as follows:

Equivalence of NFA and DFA (cont.)

1. $Q' = \mathcal{P}(Q)$.
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$.
3. $q_0' = \{q_0\}$.
4. $F' = \{R \in Q' \mid R \text{ contains some element of } F\}$.

- To allow ε arrows, define for $R \subseteq Q$,

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ by } \varepsilon \text{ arrows}\}.$$

- Replace $\delta(r, a)$ with $E(\delta(r, a))$ and set q_0' to be $E(\{q_0\})$ in the construction of N.

Equivalence of NFA and DFA (cont.)

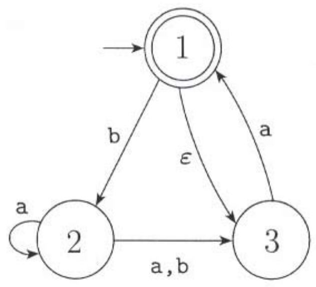


FIGURE **1.42**
The NFA N_4

Source: [Sipser 2006]

Equivalence of NFA and DFA (cont.)

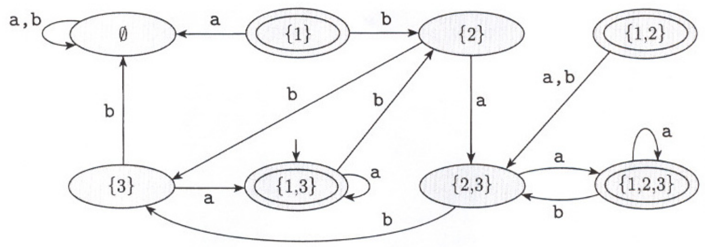


FIGURE **1.43**
A DFA D that is equivalent to the NFA N_4

Source: [Sipser 2006]

Equivalence of NFA and DFA (cont.)

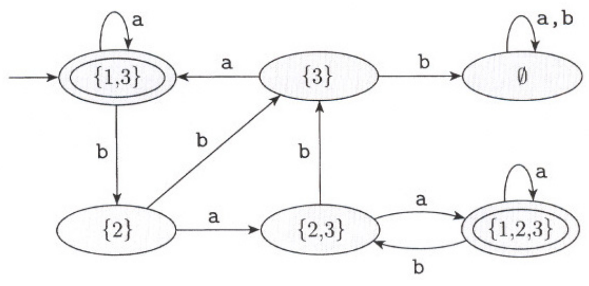


FIGURE 1.44
DFA D after removing unnecessary states

Source: [Sipser 2006]

Closedness under Union

Theorem 12 (1.45)**.** *The class of regular languages is closed under the union operation.*

- Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizing A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizing A_2.
- Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$ as follows:

Closedness under Union (cont.)

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. q_0 $(\notin Q_1 \cup Q_2)$ is the start state.
3. For $q \in Q$ and $a \in \Sigma_\varepsilon$, $\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$
4. $F = F_1 \cup F_2$.

Closedness under Union (cont.)

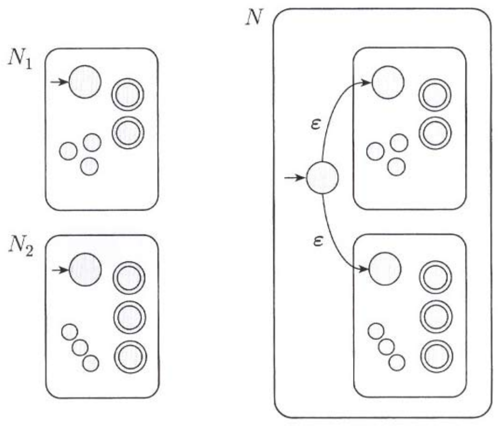


FIGURE 1.46
Construction of an NFA N to recognize $A_1 \cup A_2$

Source: [Sipser 2006]

Closedness under Concatenation

Theorem 13 (1.47)**.** *The class of regular languages is closed under the concatenation operation.*

- Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizing A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizing A_2.
- Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$ as follows:
  1. $Q = Q_1 \cup Q_2$.
  2. For $q \in Q$ and $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ but } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \ . \end{cases}$$

Closedness under Concatenation (cont.)

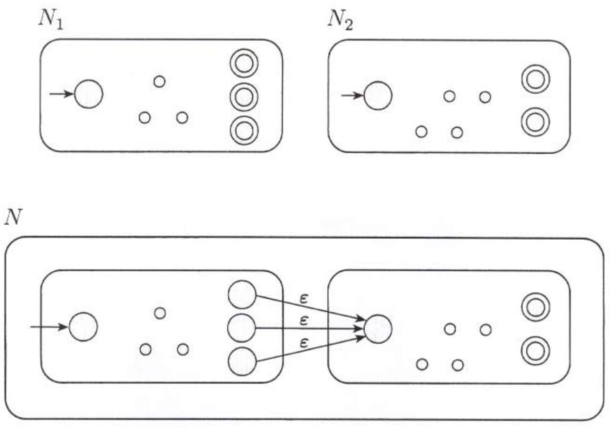


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

Source: [Sipser 2006]

Closedness under Star

Theorem 14 (1.49)**.** *The class of regular languages is closed under the star operation.*

- Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizing A.
- Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A^* as follows:
  1. $Q = \{q_0\} \cup Q_1$.
  2. For $q \in Q$ and $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ but } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

  3. $F = \{q_0\} \cup F_1$.

Closedness under Star (cont.)

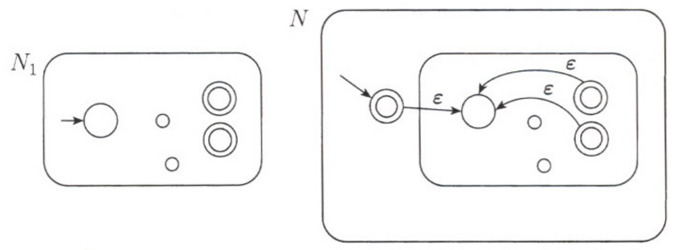


FIGURE 1.50
Construction of N to recognize A^*

Source: [Sipser 2006]

4 Regular Expressions

Regular Expressions

- We can use the regular operations (union, concatenation, star) to build up expressions, called *regular expressions*, to describe languages.
- The *value* of a regular expression is a *language*.
- For example, the value of $(0 \cup 1)0^*$ is the language consisting of all strings starting with a 0 or 1 followed by any number of 0s. (The symbols 0 and 1 are shorthands for the sets $\{0\}$ and $\{1\}$.)
- Regular expressions have an important role in computer science applications involving text.

Formal Definition of a Regular Expression

Definition 15 (1.52)**.** We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

- A definition of this type is called an *inductive definition*.
- We write $L(R)$ to denote the language of R.

Example Regular Expressions

Let Σ be $\{0, 1\}$.

- $0^*10^* = \{w \mid w \text{ has exactly a single } 1\}$.
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
- $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$.
- $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
- $\emptyset^* = \{\varepsilon\}$.

$R \cup \emptyset = R$, $R \circ \varepsilon = R$, $R \circ \emptyset = \emptyset$, but $R \cup \varepsilon$ may not equal R.

Regular Expressions vs. Finite Automata

Theorem 16 (1.54)**.** *A language is regular if and only if some regular expression describes it.*

- This theorem has two directions:
- If a language is described by a regular expression, then it is regular.
- If a language is regular, then it is described by a regular expression.
- We prove them separately.

Regular Expressions vs. Finite Automata (cont.)

Lemma 17 (1.55). *If a language is described by a regular expression, then it is regular.*

1. $R = a$ for some $a \in \Sigma$.
   $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$, $\delta(r, b) = \emptyset$ for $r \neq q_1$ or $b \neq a$.
2. $R = \varepsilon$.
   $N = (\{q\}, \Sigma, \delta, q, \{q\})$, where $\delta(r, b) = \emptyset$ for any r and b.
3. $R = \emptyset$.
   $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, where $\delta(r, b) = \emptyset$ for any r and b.
4. $R = R_1 \cup R_2$. Closed under union.
5. $R = R_1 \circ R_2$. Closed under concatenation.
6. $R = R_1^*$. Closed under star.

Regular Expressions vs. Finite Automata (cont.)

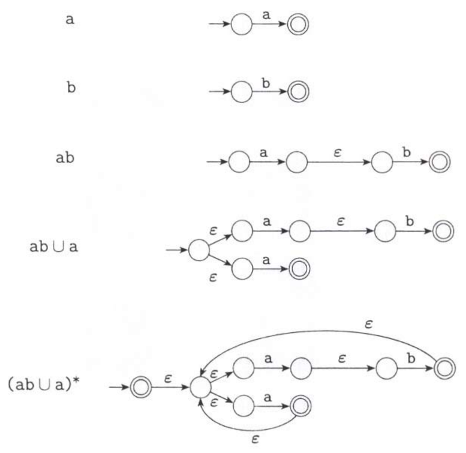


FIGURE 1.57
Building an NFA from the regular expression (ab ∪ a)*

Source: [Sipser 2006]

Regular Expressions vs. Finite Automata (cont.)

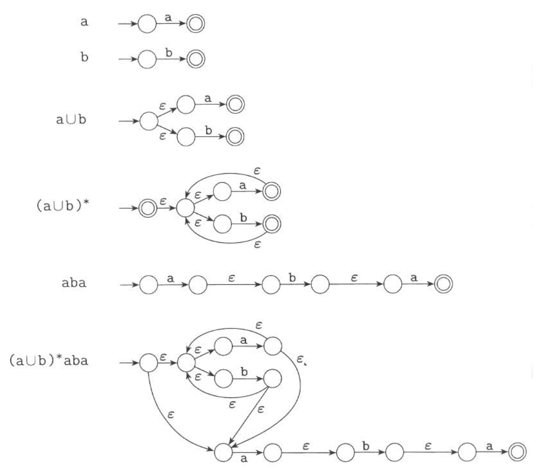


FIGURE 1.59
Building an NFA from the regular expression $(a \cup b)^*aba$

Source: [Sipser 2006]

Regular Expressions vs. Finite Automata (cont.)

Lemma 18 (1.60)**.** *If a language is regular, then it is described by a regular expression.*

- Every regular language is recognized by some DFA.
- We describe a procedure for converting DFAs into equivalent regular expressions.
- For this purpose, we introduce a new type of finite automaton called a *generalized nondeterministic finite automaton* (GNFA).
- We show how to convert DFAs into GNFAs and then GNFAs into regular expressions.

Regular Expressions vs. Finite Automata (cont.)

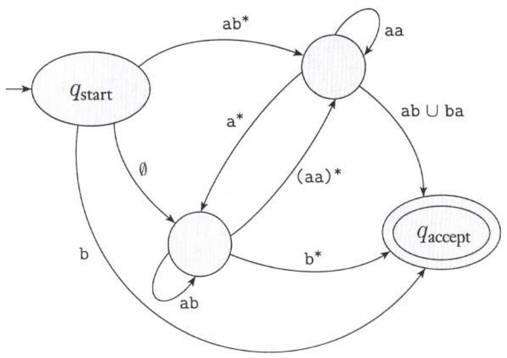


FIGURE 1.61
A generalized nondeterministic finite automaton

Source: [Sipser 2006]

Regular Expressions vs. Finite Automata (cont.)

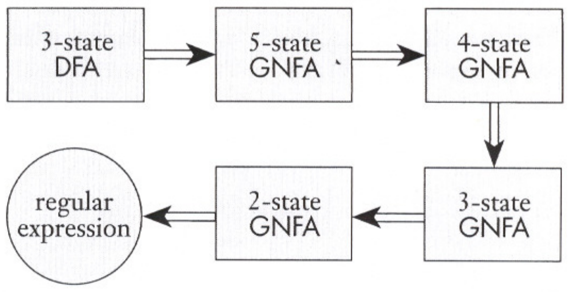


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

Source: [Sipser 2006]

Regular Expressions vs. Finite Automata (cont.)

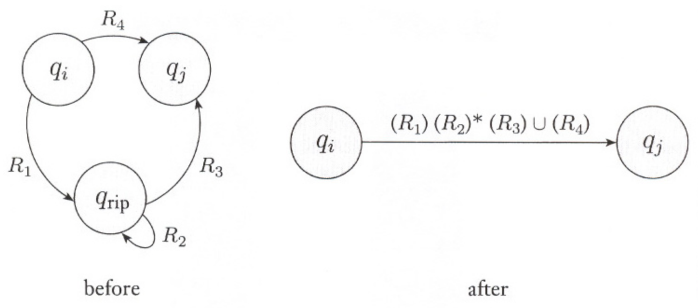


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

Source: [Sipser 2006]

Definition of a GNFA

Definition 19 (1.52)**.** A *generalized nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ is the transition function (where $\mathcal{R}$ is the collection of all regular expressions over Σ),
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

Computation of a GNFA (cont.)

A GNFA accepts a string w in Σ^* if $w = w_1 w_2 \dots w_k$, where each w_i is in Σ^*, and a sequence of states $q_0, q_1, \dots, q_k$ exists such that

1. $q_0 = q_{\text{start}}$,
2. $q_k = q_{\text{accept}}$, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

Converting a GNFA

1. Let k be the number of states of the input G.
2. If $k = 2$, return the label R of the only transition.
3. If $k > 2$, select $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept}.
   Let G' be $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where
   $$Q' = Q - \{q_{\text{rip}}\}$$
   and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$,
   $$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$
   where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Repeat with G'.

Converting a GNFA (cont.)

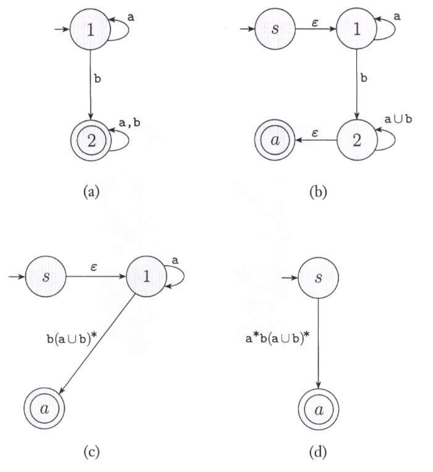


FIGURE 1.67
Converting a two-state DFA to an equivalent regular expression

Source: [Sipser 2006]

Converting a GNFA (cont.)

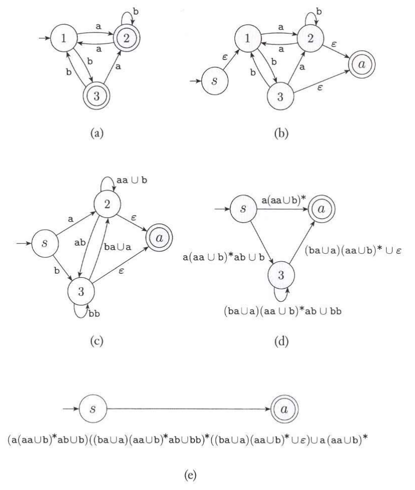


FIGURE 1.69
Converting a three-state DFA to an equivalent regular expression

Source: [Sipser 2006]

5 Nonregular Languages: The Pumping Lemma

Nonregular Languages

- To understand the power of finite automata we must also understand their limitations.
- Consider the language $B = \{0^n 1^n \mid n \geq 0\}$.
- To recognize B, a machine will have to remember how many 0s have been read so far. This cannot be done with any finite number of states, since the number of 0s is not limited.
- $C = \{w \mid w$ has an equal number of 0s and 1s$\}$ is not regular, either.

  But, $D = \{w \mid w$ has equal occurrences of 01 and 10 as substrings$\}$ is regular.
- To prove that a language is not regular, we will need a technique based on the *pumping lemma*.

The Pumping Lemma

Theorem 20 (1.70)**.** *If A is a regular language, then there is a number p (the pumping length) such that, if s is any string in A and $|s| \geq p$, then s may be divided as $s = xyz$ satisfying:*

1. *for each $i \geq 0$, $xy^iz \in A$ (string s can be "pumped"),*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

- Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that recognizes A.
- We assign the pumping length p to be the number of states of M.
- We show that any string s in A of length at least p may be broken into xyz satisfying the three conditions.

The Pumping Lemma (cont.)

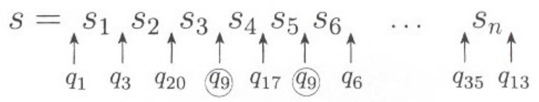


FIGURE 1.71
Example showing state q_9 repeating when M reads s

Source: [Sipser 2006]

The Pumping Lemma (cont.)

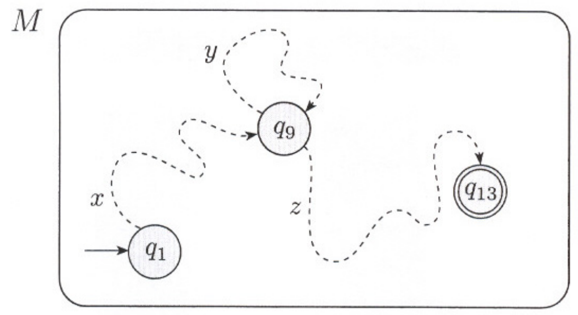


FIGURE 1.72
Example showing how the strings x, y, and z affect M

Source: [Sipser 2006]

Proving Nonregularity

Below are the steps in applying the pumping lemma to prove that a language B is not regular:

- Assume toward contradiction that B is regular.
- Then, from the pumping lemma, any string in B that is long enough (at least of the pumping length p) can be pumped.
- Find a particular string s that is long enough (whatever p is)
- Consider every possible division of s as xyz.
- The divisions may be grouped into a few patterns/cases; we may always require $|xy| \leq p$, according to the pumping lemma.
- Show that, in each of the division patterns, $xy^iz \notin B$ for some $i \geq 0$, a contradiction.

/* The statement of the pumping lemma is in the form of "if P, then Q" ($P \rightarrow Q$). The proof by contradiction assumes P and shows $\neg Q$, which together with Q implies *false* (a contradiction). Therefore, it must be the case that $\neg P$ holds. */

Example Nonregular Languages

$B = \{0^n1^n \mid n \geq 0\}$ is not regular.

- Let s be 0^p1^p, where p is the pumping length (for B).
- Three cases of dividing s as xyz (where $|y| > 0$):
  1. $\overbrace{0\cdots\underbrace{0\cdots0}_{y}\cdots0}^{p}\overbrace{1\cdots1}^{p}$: xy^2z will have more 0s than 1s and so is not in B.
  2. Similarly, for $0\cdots01\cdots\underbrace{1\cdots1}_{y}\cdots1$.
  3. $0\cdots0\underbrace{0\cdots01\cdots1}_{y}1\cdots1$: xy^2z will have some 0s after 1s and so is not in B.

Example Nonregular Languages (cont.)

$C = \{w \mid w$ has an equal number of 0s and 1s$\}$ is not regular.

- Let s be 0^p1^p, like in the proof for B.
- But, how do we deal with $0\cdots0\underbrace{0\cdots01\cdots1}_{y}1\cdots1$?
- We may assume $|xy| \leq p$:

  $\overbrace{\underbrace{0\cdots\cdots0}_{xy}0\cdots0}^{p}\overbrace{1\cdots1}^{p}$: xy^2z will have more 0s than 1s and so is not in C.
- Alternative proof:

  If C were regular, then $C \cap 0^*1^*$ would also be regular. But, we already know that $B(= C \cap 0^*1^*)$ is not regular, a contradiction.

Example Nonregular Languages (cont.)

$F = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

- Let s be 0^p10^p1.
- Again, we assume $|xy| \leq p$:

  $\overbrace{\underbrace{0\cdots\cdots0}_{xy}0\cdots0}^{p}1\overbrace{0\cdots0}^{p}1$: xy^2z will have more than p 0s before the first 1 and so is not in F.

Example Nonregular Languages (cont.)

$D = \{1^{n^2} \mid n \geq 0\}$ is not regular.

- Let s be 1^{p^2}.
- $\overbrace{\underbrace{1\cdots\cdots1}_{xy}1\cdots1}^{p^2}$: xy^2z will have $(p^2 + |y|)$ 1s.
- Again, we assume $|xy| \leq p$. Together with $|y| > 0$, we have $0 < |y| \leq p$.
- It follows that $p^2 < p^2 + |y| < p^2 + 2p + 1 = (p+1)^2$ and so xy^2z is not in D.

Example Nonregular Languages (cont.)

$E = \{0^i 1^j \mid i > j\}$ is not regular.

- Let s be $0^{p+1}1^p$.
- We assume $|y| > 0$ and $|xy| \le p$: $\overbrace{\underbrace{0\cdots\cdots0}_{xy}0\cdots0}^{p+1}\overbrace{1\cdots1}^{p}$.
- The strings xy^2z, xy^3z, etc. all have more 0s than 1s and are actually in E!
- But, by "pumping down," we get $xy^0z = xz$ which cannot have more 0s than 1s and so is not in E.