Regular Languages

Chanathip Namprempre

Computer Science
Reed College

Outline

Computational Theory

- limitations of computers
- complexity

In this class, it is helpful to view **problems** as specification of input-output pairs.

Problem: Parity of integer

Input: an integer
Output: "yes" if the integer is odd. "no" otherwise.

We can certainly write programs to solve this problem.

Limitations of computers: Decidability

Other problems: Halting problem, Group isomorphism problem

We will learn that these two problems are **undecidable**.
[i.e., whatever programs you write to solve these problems, they will be wrong on some input.]

To be able to prove these claims, we need precise mathematical models of computation.

In this class, we learn about automata, grammar, and Turing machines.

Complexity

Some problems are solvable but take a long time, e.g. factoring.

We study the complexity of these problems, i.e., the inherent hardness of the problems.

This course is useful!

- Learn about limitations of computers
- Learn practical models: automata (calculators, doors, coke machines, microwaves, cruise control), grammars (compilers)
- Learn when to stop looking for better solutions to problems
- Learn abstract thinking skills

In theory of computation, we are interested in **modelling** computers, also known as machines.

We first start with the simplest kind of machine, finite automata.

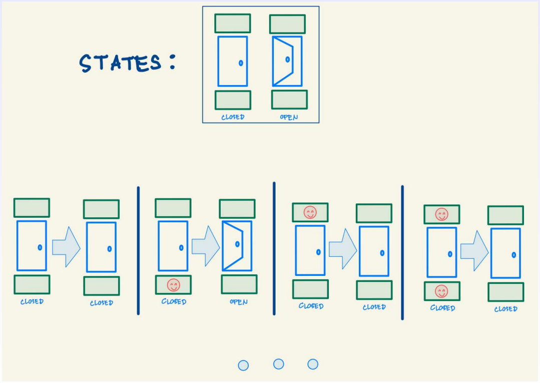
Their key feature is that they have a finite number of states. (So they are also called "finite state machines.")

Outline

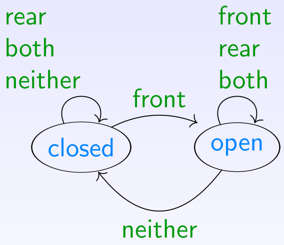
Finite Automata

Finite automata models computers with very small memory.

Examples: digital microwave, automatic doors, digital watches, fans, etc.



Old supermarket door DFA



state \ input	neither	front	rear	both
closed	closed	open	closed	closed
open	closed	open	open	open

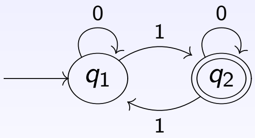
Mathematical objects required

- set
- ordered list
- cartesian product of sets
- sequence
- function
- symbol
- alphabet: set of symbols
- string: sequence of symbols
- language: set of strings

DFA example 2

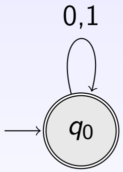
Problem: Parity of ones

Input: a string of zeros and ones
Output: “Yes” if there are odd number of ones. “No” otherwise.



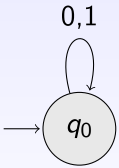
DFA example 3

Input: a string of zeros and ones
Output: "Yes" if ?? "No" if ??



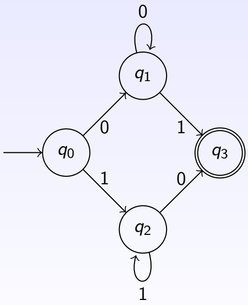
DFA example 4

Input: a string of zeros and ones
Output: "Yes" if ?? "No" if ??



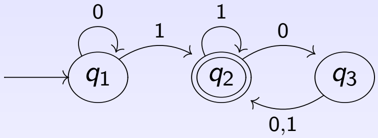
DFA example 5

Input: a string of zeros and ones
Output: "Yes" if ?? "No" if ??

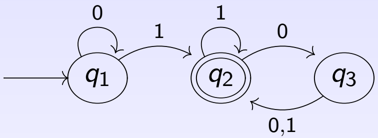


Parts of a finite automaton

Call this machine M_1



This is called a state diagram of M_1.
The states of M_1 are q_1, q_2, q_3.
The start state is q_1.
The accept state is q_2.
The transitions are indicated by arrows going from state to state.
The outputs of M_1 are either accept or reject.


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_3 \arrow[l, bend left, "{0,1}"]
\end{tikzcd}$$

Try using M_1 to process the strings 1101, 1, 01, 11, 0101010101, 100, 0100, 0101000000, 0, 10, 101000.

[Basically, it accepts anything ending with 1 and anything having even number of 0s following the last 1. It rejects other strings.]

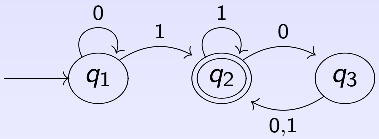
Formal definition of a finite automaton

Definition

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

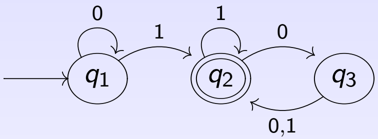
1. Q is a finite set called the set of states,
2. Σ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state,
5. $F \subseteq Q$ is the set of accept states.

How would M_1 be formally described?



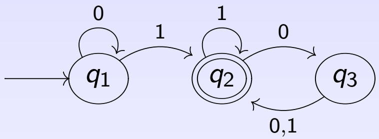
The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q =$
2. $\Sigma =$
3. δ is this function:
4. The start state is
5. $F =$


$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & \boxed{q_2} \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_3 \arrow[l, bend left, "{0,1}"] \end{tikzcd}$$

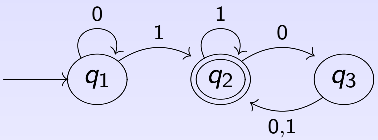
The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma =$
3. δ is this function:
4. The start state is
5. $F =$



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

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is this function:
4. The start state is
5. $F =$

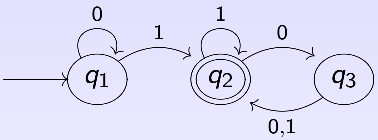


The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is this function:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. The start state is
5. $F =$

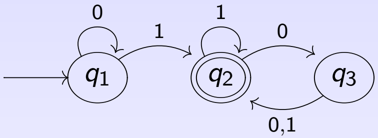


The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is this function:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. The start state is q_1
5. $F =$



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

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is this function:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. The start state is q_1
5. $F = \{q_2\}$

What it means for a DFA to accept strings and to recognize a language

Definition

A DFA M accepts a string w if, after executing on w as described, it ends up in an accept state.

Definition

A DFA M recognizes a language L if

1. For every $w \in L$, M accepts w
2. For every $w \notin L$, M rejects w.

More terminology

Let M be a machine, and let A be the set of all strings that a machine M accepts. All these phrases mean the same thing.

1. The language of machine $M = A$
2. M recognizes A
3. $L(M) = A$
4. M accepts A

Observations

- A machine may accept many strings but can recognize only one language.
- A machine accepting no strings is considered recognizing one language, i.e., the empty language $\emptyset$.
- If L is a language, and M is a DFA. It does not make sense to say these things:
  - the language 011
  - Run L on input 011.
  - L accepts 100.
  - The language accepted by M is 011.

Formal definition of computation

We know how to compute with a finite automaton, i.e., given a string w, we know how M_1 processes it. Now we formalizes this process.

Let $n \in \mathrm{Z}^+$, $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, and $w = w_1 w_2 \ldots w_n$ be a string over the alphabet Σ.

Definition

M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ exists in Q with the following three conditions:

1. [M starts in the start state.] $r_0 = q_0$,
2. [M moves according to δ.] $r_{i+1} = \delta(r_i, w_{i+1})$ for $i = 0, \ldots, n-1$,
3. [M ends up in an accept state.] $r_n \in F$.

Regular languages

Definition

A language is called a regular language if some finite automaton recognizes it.

Designing finite automata

Given some description of the (regular) language, we are interested in constructing a finite automaton that recognizes it.

Examples

1. The language of all strings with an odd number of 1s.
2. The language of all strings that contain the string 001 as a substring.
3. $\{w \mid w$ begins with a 1 and ends with a 0 $\}$
4. $\{w \mid w$ contains at least three 1s $\}$
5. $\{\varepsilon, 0\}$
6. $\emptyset$
7. all strings except the empty string

More examples of regular languages

Alphabet Σ	$\{0\}$
Definition of L_1	$\{0, 00, 000, 0000, \ldots\}$
Sample strings in L_1	$0, 00, 000, \ldots$
Alphabet Σ	$\{0\}$
Definition of L_2	$\{0^n$ for $n = 1, 2, 3, \ldots\}$
Sample strings in L_2	$0, 00, 000, \ldots$
Alphabet Σ	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Definition of L_3	{ any finite string of letters that does not start with the letter 0 }
Sample strings in L_3	$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, \ldots$

Alphabet Σ	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Definition of L_4	{ any finite string of letters that, if it starts with the letter 0, has no more letters after the first }
Sample strings in L_4	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, \ldots$

Alphabet Σ	$\{a, b, c, d, e, \ldots, z\}$
Definition of L_5	{ all the words in a dictionary }
Sample strings in L_5	discussion, list, body, trick, ...

The regular operations

Definition

Let A and B be languages. We define the regular operations union, concatenation, and star 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: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Union	take all the strings in both A and B and lump them together into one language.
Concatenation	attach a string from A in front of a string from B in all possible ways to get strings in the new language.
Star	attach any number of strings in A together to get a string in the new language.

Examples

Let the alphabet be $\{a, b, c, \ldots\}$ for L_1, L_2 and $\{0, 1\}$ for L_3. Try regular operations on these languages:

Example

$L_1 = \{$ good, bad $\}$; $L_2 = \{$ dog, cat $\}$; $L_3 = \{$ 1, 10, 11 $\}$

- $L_1 \cup L_1 =?$
- $L_1 \cup L_2 =?$
- $L_2 \cup L_1 =?$
- $L_1 \circ L_2 =?$
- $L_2 \circ L_1 =?$
- $L_1 \circ L_1 =?$
- $L_1 \circ \{\varepsilon\} =?$
- $L_1 \circ \emptyset =?$
- $L_2^* =?$
- $L_3^* =?$
- Does L_3^* contain 101?, 11010?, 1001?

Introducing closure

Consider the example languages we have seen:

Example

$L_1 = \{ \text{ good, bad } \}$; $L_2 = \{ \text{ dog, cat } \}$; $L_3 = \{ 1, 10, 11 \}$

- They are all regular languages. (Can you write DFAs for them?)
- Is $L_1 \circ L_2$ regular? What about $L_1 \cup L_2$? What about L_3^*?

It turns out that, if you take any regular languages and apply a regular operation on them, you always get back a regular language.

Closure

Definition

A set is closed under an operation *op* if applying *op* to members of the set returns an object still in the set.

Theorem

The class of regular languages is closed under the union, concatenation, and star operations.

If A, B are regular languages, this means that $A \cup B$, $A \circ B$, and A^* are all regular languages.

Proof?

Theorem

The class of regular languages is closed under union.

How to prove this theorem?

Outline

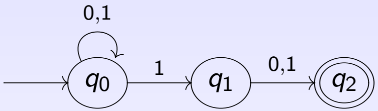
$$L = \{ w \mid w \text{ has a 1 in the 2nd position from the right end } \}$$

We can write a DFA for this, but it is a bit of a pain. It is hard because we cannot rewind the input.

We can use nondeterminism to help us.

Suppose we can guess where the 2nd position from the right end is. All we have to do once we make a guess is to verify if our guess is correct.

$$L = \{ w \mid w \text{ has a 1 in the 2nd position from the right end } \}$$



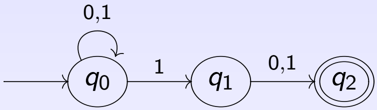
Try it on 01011: None of these accept, but it's our fault for making wrong guesses.

- $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0$
- $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$
- $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_1$

But there's a way to get the machine to accept!

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$$

$$L = \{ w \mid w \text{ has a 1 in the 2nd position from the right end } \}$$



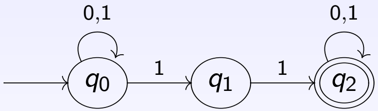
Try it on 1001: no way to get the machine to accept.
e.g. $q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1$

Note that we can represent all the possible ways to process a string with the machine using a **tree**.

The nodes at each level in the tree can be thought of as your fingers keeping track of all the possible states you can be at after having processed the input symbols upto that level.

Another example

$$L = \{ w \mid w \text{ contains 11 as a substring } \}$$



$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[loop above, "0{,}1"] \arrow[r, "1"] & q_1 \arrow[r, "1"] & \boxed{q_2} \arrow[loop above, "0{,}1"]
\end{tikzcd}$$

Ways to think about non-determinism

According to Michael Sipser (https://youtu.be/oNsscmUwjMU)

Computational:	Fork new parallel thread and accept if any thread leads to an accept state.
Mathematical:	Tree with branches. Accept if any branch leads to an accept state.
Magical:	Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to acceptance, if possible.

Definition

We say that an NFA M accept an input string w if there is *some path* which can be followed on input w and leads to an accept state.

It rejects w if *all* paths that can be followed on input w lead to rejection.

Definition

We say that M accepts a language L if

- for every input $w \in L$ we have $M(w)$ accepts, and
- for every input $w \notin L$, we have $M(w)$ rejects.

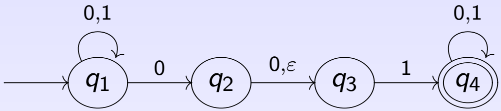
We write, $L(M)$ for the language L accepted by M.

Formal definition of a nondeterministic finite automaton

Definition

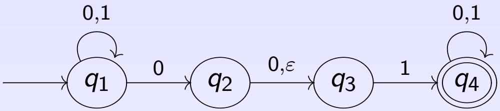
A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

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



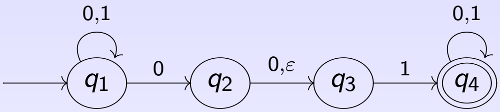
The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma =$
3. δ is this function:
4. The start state is
5. $F =$



The formal description of this machine 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 this function:
4. The start state is
5. $F =$

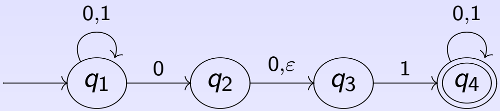


The formal description of this machine is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. δ is this function:

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

4. The start state is
5. $F =$

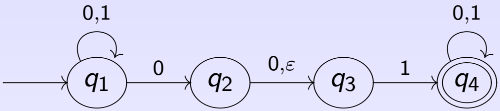

$$\begin{tikzcd}
{} \arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "0"] & q_2 \arrow[r, "0{,}\varepsilon"] & q_3 \arrow[r, "1"] & \boxed{q_4} \arrow[loop above, "0,1"]
\end{tikzcd}$$

The formal description of this machine 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 this function:

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

4. The start state is q_1
5. $F =$


$$\begin{tikzcd}
{} \arrow[r] & q_1 \arrow[loop above, "{0,1}"] \arrow[r, "0"] & q_2 \arrow[r, "{0,\varepsilon}"] & q_3 \arrow[r, "1"] & \boxed{q_4} \arrow[loop above, "{0,1}"]
\end{tikzcd}$$

The formal description of this machine 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 this function:

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

4. The start state is q_1
5. $F = \{q_4\}$

Formal definition of computation for NFA

Let $n \in \mathrm{Z}^+$, $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, and $w = w_1 w_2 \dots w_n$ be a string over the alphabet Σ.

Definition

M accepts w if a sequence of states $r_0, r_1, \dots, r_n$ exists in Q with the following three conditions:

1. [M starts in the start state.] $r_0 = q_0$,
2. [r_{i+1} is one of the allowable next states from r_i with input w_{i+1}.] $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \dots, n-1$,
3. [M ends up in an accept state.] $r_n \in F$.

Equivalence of NFAs and DFAs

Theorem

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

The proof of this theorem is called the subset construction.
Key observation: keep track of states with fingers on them.

Corollary

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

Remember closure?

Theorem

The class of regular languages is closed under the union, concatenation, and star operations.

We can prove closure of regular languages under union, concatenation, and star very easily using NFAs.

Closure of class of regular languages under union

Theorem

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

Proof.

Let L_1 be a regular language recognized by an NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
Let L_2 be a regular language recognized by an NFA $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
Construct $N = (Q, \Sigma, \delta, q_0, F)$ for $L_1 \cup L_2$ as follows:

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$
2. q_0 is the start state of N.
3. $F = F_1 \cup F_2$
4. For any $q \in Q$ and $a \in \Sigma \cup \{\varepsilon\}$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \varepsilon \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

□

Closure of class of regular languages under star

Theorem

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

Proof.

Let L be a regular language recognized by an NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
Construct $N = (Q, \Sigma, \delta, q_0, F)$ for L^* as follows. Note there is a bug in the proof in Sipser's book and video.

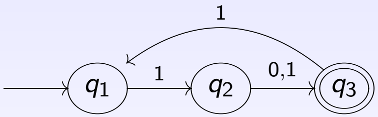
1. $Q = \{q_0\} \cup Q_1$
2. q_0 is the new start state of N.
3. $F = \{q_0\} \cup F_1$
4. For any $q \in Q$ and $a \in \Sigma \cup \{\varepsilon\}$,

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

□

Bug in Sipser's proof for closure under star

To see why the construction provided in the book is problematic, consider the following NFA N_1 recognizing the language $L = 1(0 \cup 1)(11(0 \cup 1))^*$.


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[r, "1"] & q_2 \arrow[r, "{0,1}"] & q_3 \arrow[ll, bend right, "1"']
\end{tikzcd}$$

Imagine the new machine N constructed to recognize L^* according to the construction in the book.

- Does N accept 1010?
- Should N accept 1010?