بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۱)

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

ایمیل: h.falsafain@iut.ac.ir یا h.falsafain@gmail.com
آدرس دفتر: اتاق ۳۲۳ دانشکده
شمارهٔ تلفن دفتر: ۳۳۹۱۹۰۶۸ـ۰۳۱
آدرس وبسایت شخصی: https://falsafain.iut.ac.ir/

تمامي امور مربوط به درس (اعم از امور مربوط به تکالیف و کوئیزها) از طریق سامانهٔ یکتا صورت می‌پذیرند: https://yekta.iut.ac.ir/

لطفاً تا جای ممکن از طریق ایمیل درخواست‌ها، پرسش‌ها، و نظرات خود را مطرح کنید. لطفاً حتی‌الامکان روی اسکایپ، یکتا، تلگرام، و غیره پیامی نفرستید.

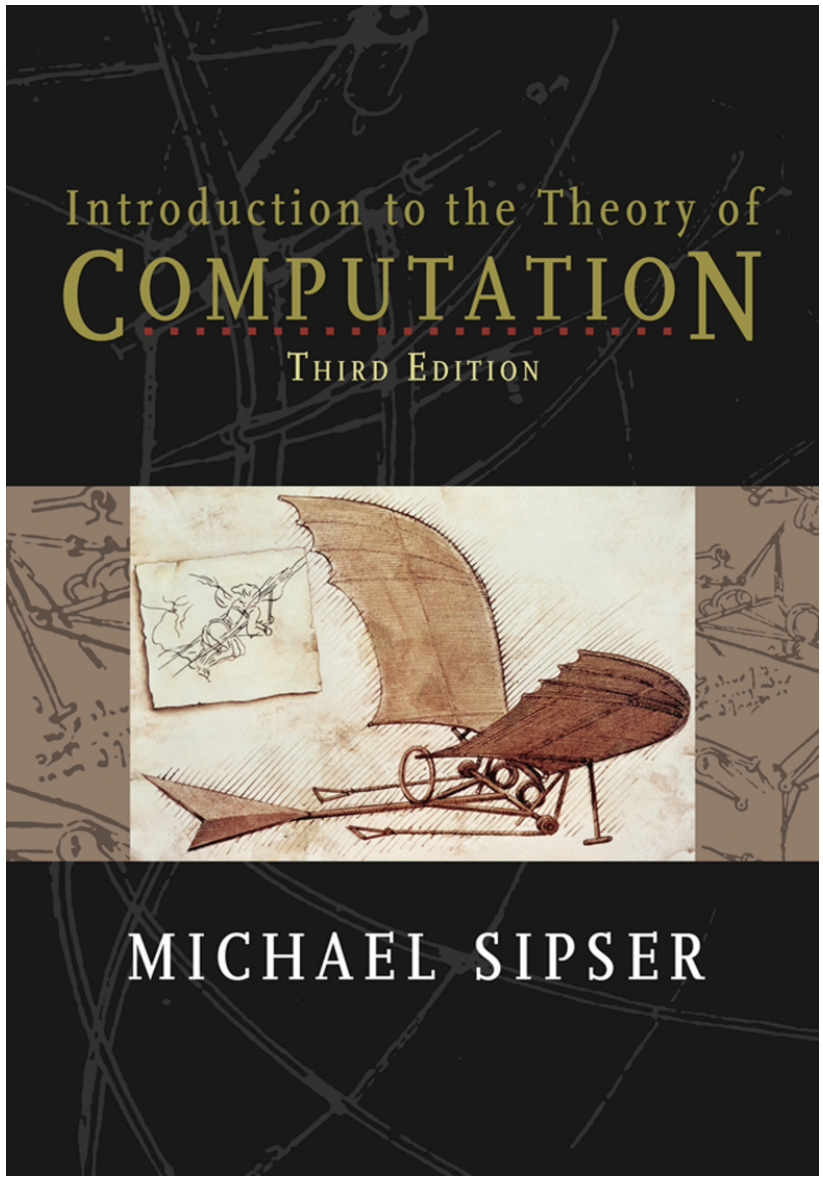
h.falsafain@iut.ac.ir & h.falsafain@gmail.com

ارزشیابی

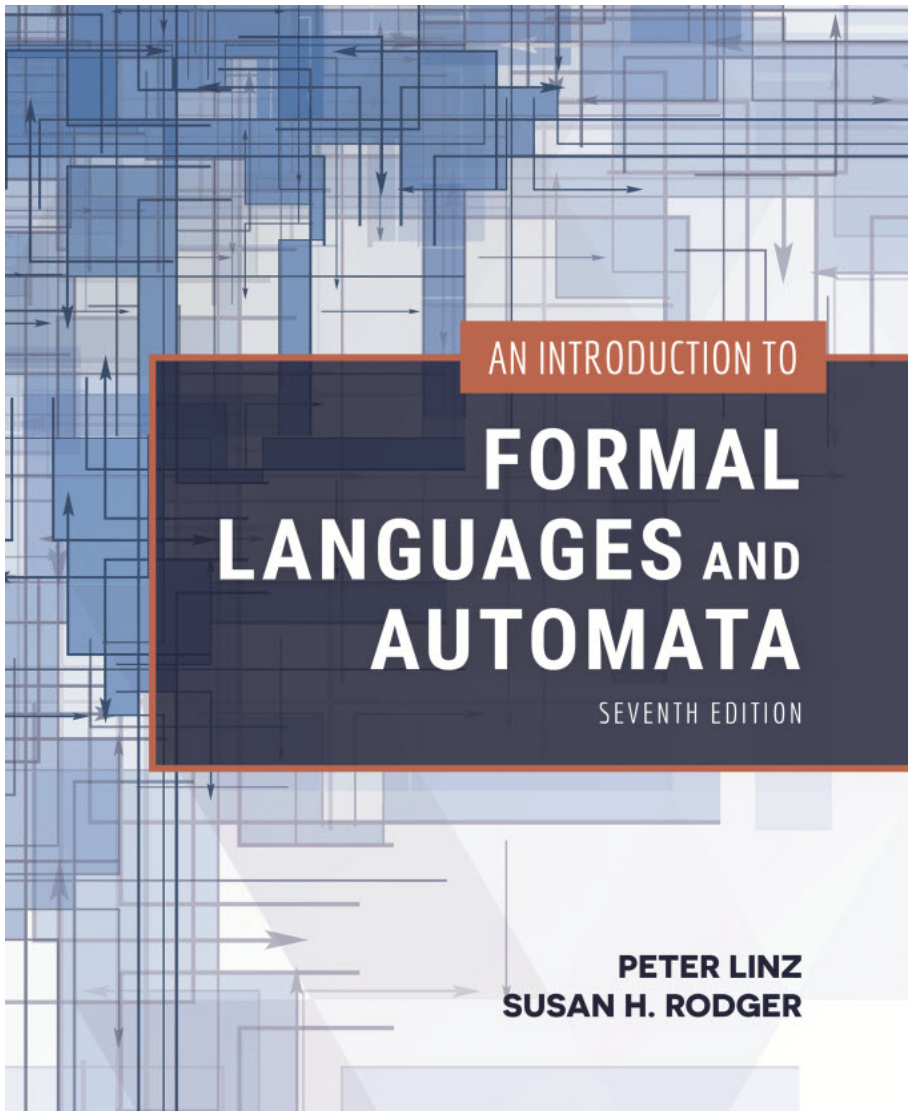
* کوییزها، تکالیف، و پرسش‌های کلاسی: تقریباً ۵ نمره
* آزمون میانترم: تقریباً ۷.۵ نمره
* آزمون پایانترم: تقریباً ۷.۵ نمره
* حضور در کلاس و فعالیت کلاسی: حداقل ۱ نمره

زمان تشکیل کلاس حل تمرین، بر اساس نظرسنجی معیّن خواهد شد.

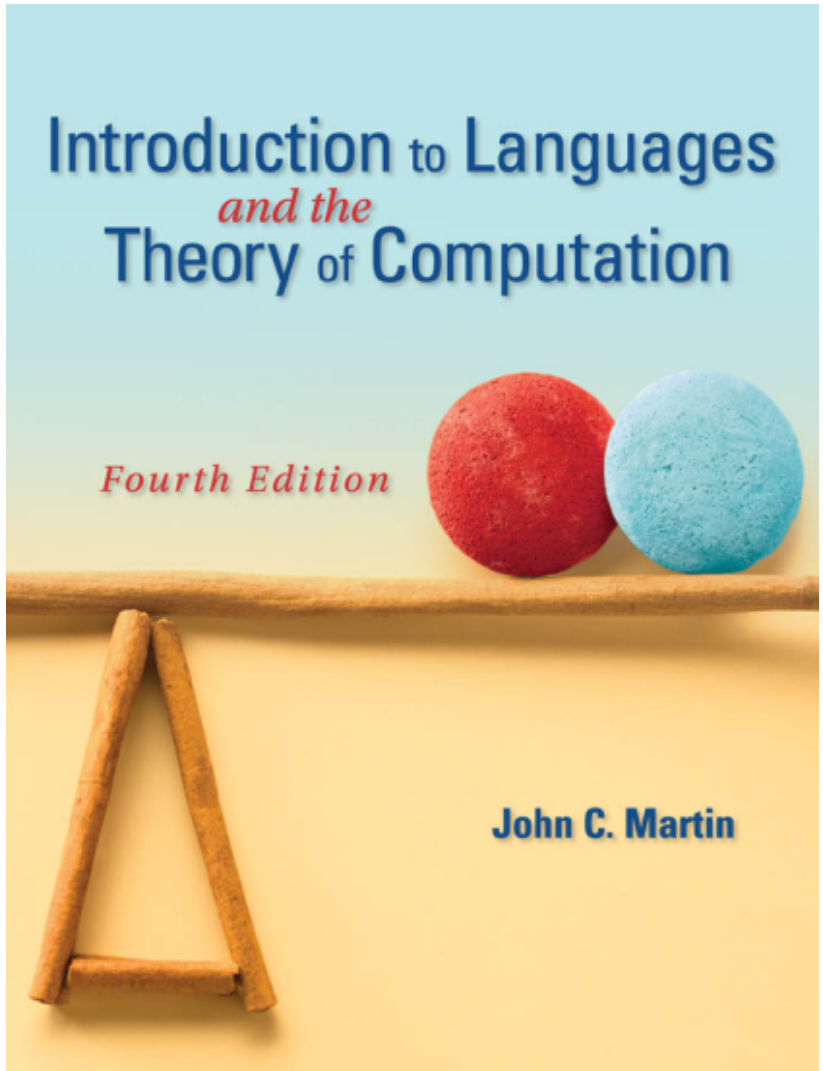
کتاب‌های مرجع



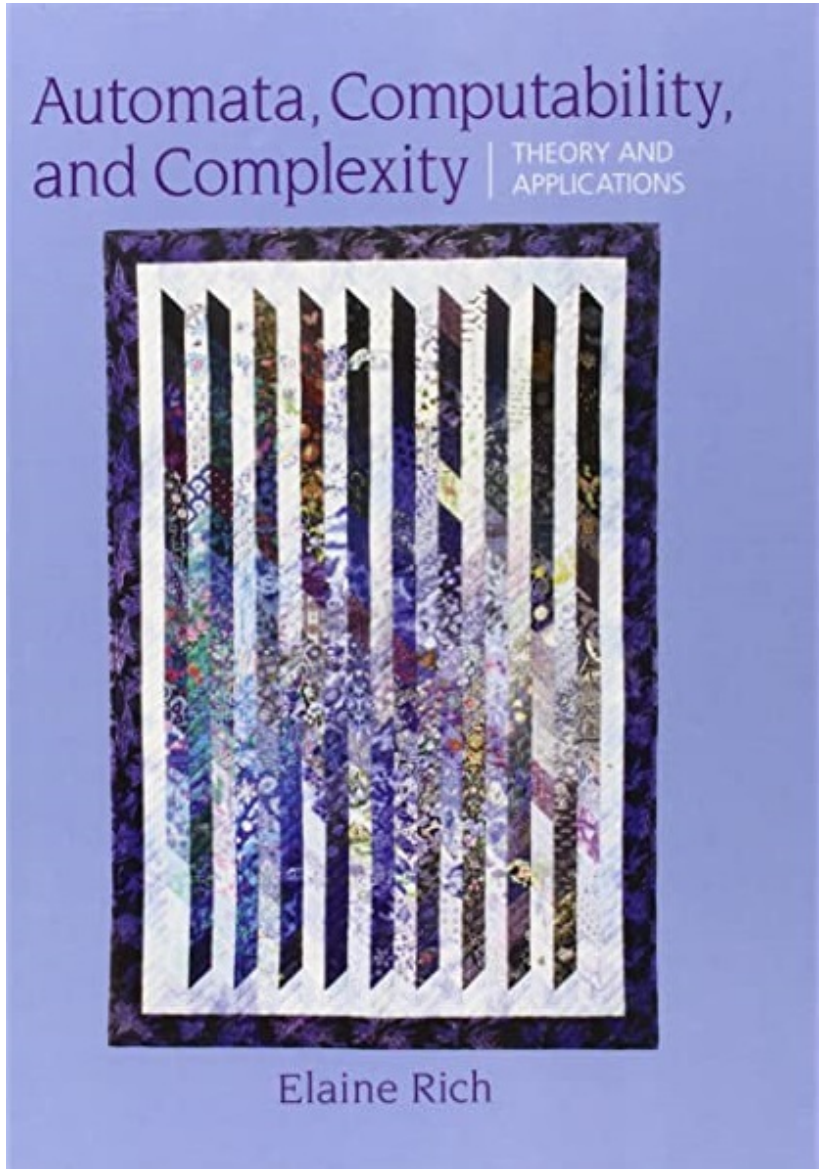
کتاب‌های مرجع



کتاب‌های مرجع



کتاب‌های مرجع



Automata, Computability, and Complexity

This course focuses on three traditionally central areas of the theory of computation: automata, computability, and complexity. They are linked by the question: What are the fundamental capabilities and limitations of computers? In each of the three areas—automata, computability, and complexity—this question is interpreted differently, and the answers vary according to the interpretation.

Automata Theory

Automata theory deals with the definitions and properties of mathematical models of computation. These models play a role in several applied areas of computer science. One model, called the finite automaton, is used in text processing, compilers, and hardware design. Another model, called the context-free grammar, is used in programming languages and artificial intelligence. Automata theory is an excellent place to begin the study of the theory of computation. The theories of computability and complexity require a precise definition of a computer. Automata theory allows practice with formal definitions of computation as it introduces concepts relevant to other non-theoretical areas of computer science.

Computability Theory

During the first half of the twentieth century, mathematicians such as Kurt Gödel, Alan Turing, and Alonzo Church discovered that certain basic problems cannot be solved by computers. The theories of computability and complexity are closely related. In complexity theory, the objective is to classify problems as easy ones and hard ones; whereas in computability theory, the classification of problems is by those that are solvable and those that are not. Computability theory introduces several of the concepts used in complexity theory.

Complexity Theory

Computer problems come in different varieties; some are easy, and some are hard. What makes some problems computationally hard and others easy? This is the central question of complexity theory. Remarkably, we don't know the answer to it, though it has been intensively researched for over 40 years. Later, we explore this fascinating question and some of its ramifications. In one important achievement of complexity theory thus far, researchers have discovered an elegant scheme for classifying problems according to their computational difficulty. It is analogous to the periodic table for classifying elements according to their chemical properties. Using this scheme, we can demonstrate a method for giving evidence that certain problems are computationally hard, even if we are unable to prove that they are.

One applied area that has been affected directly by complexity theory is the ancient field of cryptography. In most fields, an easy computational problem is preferable to a hard one because easy ones are cheaper to solve. Cryptography is unusual because it specifically requires computational problems that are hard, rather than easy. Secret codes should be hard to break without the secret key or password. Complexity theory has pointed cryptographers in the direction of computationally hard problems around which they have designed revolutionary new codes.

Strings & Languages

The alphabet over which the strings are defined may vary with the application. For our purposes, we define an alphabet to be any nonempty finite set. The members of the alphabet are the symbols (or letters) of the alphabet. We generally use capital Greek letters Σ and Γ to designate alphabets and a typewriter font for symbols from an alphabet. The following are a few examples of alphabets.

$$\Sigma_1 = \{0, 1\}$$

$$\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$$

$$\Gamma = \{\mathtt{0}, \mathtt{1}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$$

☞ *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. If* $\Sigma_1 = \{0, 1\}$*, then* 01001 *is a string over* Σ_1*. If* $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$*, then* $\mathtt{abracadabra}$ *is a string over* Σ_2*.*

☞ *If* w *is a string over* Σ*, the length of* w*, written* $|w|$*, is the number of symbols that it contains.*

☞ *The string of length zero is called the empty string and is written* ε*. The empty string plays the role of* 0 *in a number system.*

☞ *If* w *has length* n*, we can write* $w = w_1 w_2 \cdots w_n$ *where each* $w_i \in \Sigma$*.*

☞ *The reverse of* w*, written* w^R*, is the string obtained by writing* w *in the opposite order (i.e.,* $w_n w_{n-1} \cdots w_1$*).*

☞ *String* z *is a substring of* w *if* z *appears consecutively within* w*. For example,* $\mathtt{cad}$ *is a substring of* $\mathtt{abracadabra}$*.*

☞ *If we have string* x *of length* m *and string* y *of length* n*, the concatenation of* x *and* y*, written* xy*, is the string obtained by append-*

ing y to the end of x, as in $x_1 \cdots x_m y_1 \cdots y_n$. To concatenate a string with itself many times, we use the superscript notation x^k to mean

$$\overbrace{xx \cdots x}^{k}.$$

☞ *The lexicographic order of strings is the same as the familiar dictionary order. We'll occasionally use a modified lexicographic order, called shortlex order or simply string order, that is identical to lexicographic order, except that shorter strings precede longer strings. Thus the string ordering of all strings over the alphabet $\{0, 1\}$ is*

$$(\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots).$$

☞ *Say that string x is a prefix of string y if a string z exists where $xz = y$, and that x is a proper prefix of y if in addition $x \neq y$.*

☞ *A language is a set of strings. A language is prefix-free if no member is a proper prefix of another member.*

What is a computer?

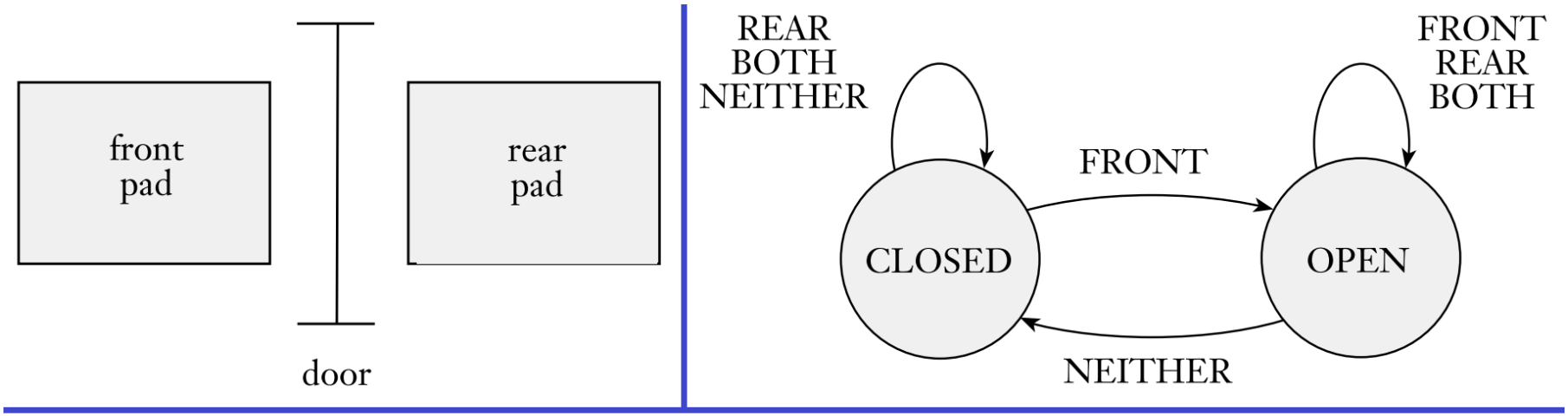
The theory of computation begins with a question: What is a computer? It is perhaps a silly question, as everyone knows that this thing I type on is a computer. But these real computers are quite complicated—too much so to allow us to set up a manageable mathematical theory of them directly. Instead, we use an idealized computer called a computational model. As with any model in science, a computational model may be accurate in some ways but perhaps not in others. Thus we will use several different computational models, depending on the features we want to focus on. We begin with the simplest model, called the finite state machine or finite automaton.

Finite automata

Finite automata are good models for computers with an extremely limited amount of memory. What can a computer do with such a small memory? Many useful things! In fact, we interact with such computers all the time, as they lie at the heart of various electromechanical devices.

We will now take a closer look at finite automata from a mathematical perspective. We will develop a precise definition of a finite automaton, terminology for describing and manipulating finite automata, and theoretical results that describe their power and limitations. Besides giving you a clearer understanding of what finite automata are and what they can and cannot do, this theoretical development will allow you to practice and become more comfortable with mathematical definitions, theorems, and proofs in a relatively simple setting.

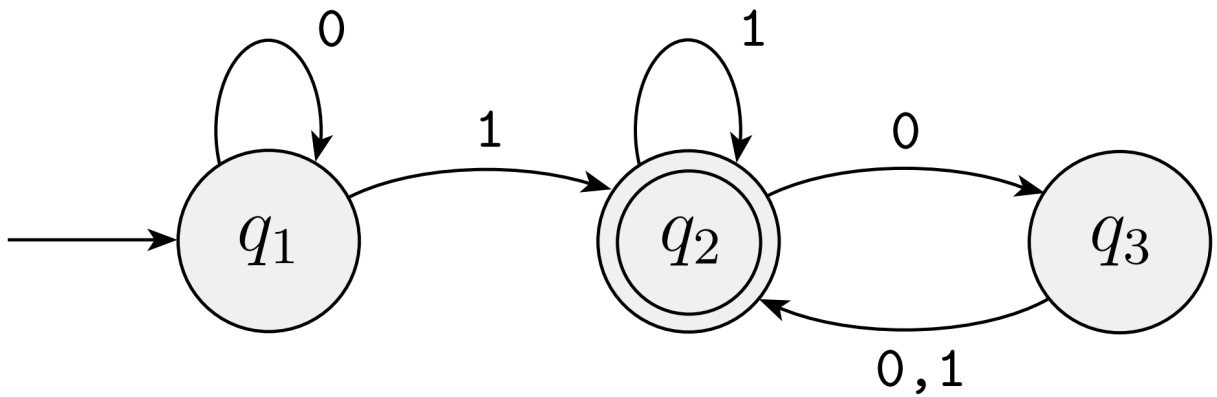
The controller for an automatic door is one example of such a device:



input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

The following figure depicts a finite automaton called M_1.



The figure 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.

When this automaton receives an input string such as `1101`*, it processes that string and produces an output. The output is either accept or reject. We will consider only this yes/no type of output for now to keep things simple. The processing begins in* M_1*'s start state. The automaton receives the symbols from the input string one by one from left to right. After reading each symbol,* M_1 *moves from one state to another along the transition that has that symbol as its label. When it reads the last symbol,* M_1 *produces its output. The output is accept if* M_1 *is now in an accept state and reject if it is not.*

For example, when we feed the input string 1101 *into the machine* M_1*, 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.*

Experimenting with this machine on a variety of input strings reveals that it accepts the strings 1, 01, 11, ***and*** 0101010101. ***In fact,*** M_1 ***accepts any string that ends with a*** 1, ***as it goes to its accept state*** q_2 ***whenever it reads the symbol*** 1. ***In addition, it accepts strings*** 100, 0100, 110000, ***and*** 0101000000, ***and any string that ends with an even number of*** 0***s following the last*** 1. ***It rejects other strings, such as*** 0, 10, 101000.