

Languages and Machines

Alphabets

An *alphabet* is a finite set of *symbols*. There is no definition of *symbol*. Alphabets used in this course include:

The alphabet of all ASCII symbols.

The Roman alphabet: upper case, lower case, or both.

The decimal alphabet: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

The binary alphabet: $\{0, 1\}$.

The unary alphabet: $\{1\}$.

Small subsets of the Roman alphabet, such as $\{a, b\}$.

Strings

A *string* is a finite sequence of symbols over some alphabet. For example, if $\Sigma = \{a, b, c\}$, then a , b , aba , $abccaa$, are strings of length 1, 3, or 6 over $\{a, b, c\}$. The empty string, denoted λ (or ϵ) has length zero and consists of no symbols.

We write Σ^* to mean the set of all strings over the alphabet Σ . Σ^* , which is countably infinite. For any string $w \in \Sigma^*$, we let $|w|$ be the length of w .

The binary alphabet is of particular importance in computer science. We use the term *binary string* to mean any string over the binary alphabet.

Languages

A *language* is defined to be a set of strings over a particular alphabet. If L is a language over Σ , then $L \subseteq \Sigma^*$.

There is no definition of symbol, and thus anything can be a symbol. The language of DNA strings is over the alphabet consisting of the four nucleotides: adenine, thymine, guanine, and cytosine, usually abbreviated as A, T, G, and C.

Example. A *programming language* is a set of *programs*, each of which is a string over the alphabet consisting of all symbols used in that language. including blank and end-of-line.

Numerals and Numbers

We distinguish between a number and a numeral. A number is an abstract object which has no physical existence. A numeral is something (usually a string) which denotes a number. If n is a number, we write $\langle n \rangle$ to mean a numeral which denotes n .

Problems and Languages

We are primarily interested in infinite problems, that is, problems which have infinitely many instances. For example, “What is $2+3$?” is an instance of the addition problem.

A 0/1 problem is any problem where the answer for each instance is either 0 (false) or 1 (true). For example, an instance of the *binary primality* problem is a binary numeral $\langle n \rangle$, and the answer is 1 (true) if n is prime, 0 (false) otherwise. We let L_{prime} be the language of all binary numerals for prime numbers. For example, the strings 1011 and 10010 are instances of the binary primality problem, but only one of those, namely 1011, is a member of the language L_{prime} .

A problems that have multiple possible answers frequently have a 0/1 version. For example, instead of asking for the prime factors of n , we could ask whether n has a prime factor smaller than a given other number a .

Languages and 0/1 problems are essentially the same thing. For any language L , there is a *membership problem*. If $L \subseteq \Sigma^*$, every string over Σ is an instance of the membership problem for L . For the instance $w \in \Sigma^*$, the answer is 1 if $w \in L$ and 0 if $w \notin L$. Many language classes, such as \mathcal{P} -TIME, are defined by the hardness of their membership problems. A language is said to be “hard” or “easy” if its membership problem is hard or easy.¹

Deterministic Finite Automata

A machine M is called a *finite automaton* (FA) if its **id** consists of one of a finite set of states together with its current unread input. A *deterministic finite automaton* (DFA) M has a finite set of *states* Q , one of which (usually called q_0) is the *start* state. There is a subset $F \subseteq Q$ of *final* states. An input for a DFA is a string $w \in \Sigma^*$, where Σ is called the input alphabet. M also has a *transition function* $\delta : Q \times \Sigma \rightarrow Q$. Formally, M is the quintuple $(Q, \Sigma, \delta, q_0, F)$. An **id** of M is an ordered pair (q, u) , where $q \in Q$ is the current state and $u \in \Sigma^*$ is the remaining (unread) input. The initial **id** of M is (q_0, w) , where w is the input string. We can generalize the transition function to $\delta : Q \times \Sigma^* \rightarrow Q$ by recursion:

$$\begin{aligned}\delta(q, \lambda) &= q, \text{ for } q \in Q. \\ \delta(q, wa) &= \delta(\delta(q, w), a), \text{ for } q \in Q, w \in \Sigma^*, a \in \Sigma.\end{aligned}$$

Steps of M . The number of steps a DFA M takes during a computation is equal to the length of the input string. During each step, M reads the first symbol of the remaining input, then changes its state. If $q \in Q$ is the current state and a is the next symbol of input, the state changes to $\delta(q, a)$. If the last state is final, w is accepted, otherwise w is rejected. If a DFA M accepts a language L , it is also true that M decides L , since it always halts. A language is defined to be *regular* if it is accepted by some DFA

¹“Hard” and “easy” are relative terms, like “large” and “small,” or “warm” and “cool.” A “small” planet is larger than a “large” animal.

Example

Let M be the DFA where $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, and δ is defined by the transition table given in Table 1, and illustrated as a state diagram in Figure 2

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_1

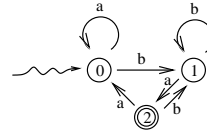


Figure 2: State Diagram of M

Table 1

Figure 3 shows a computation of M which accepts the string $abba$, while Figure 4 shows a computation of M which rejects the string $abab$.

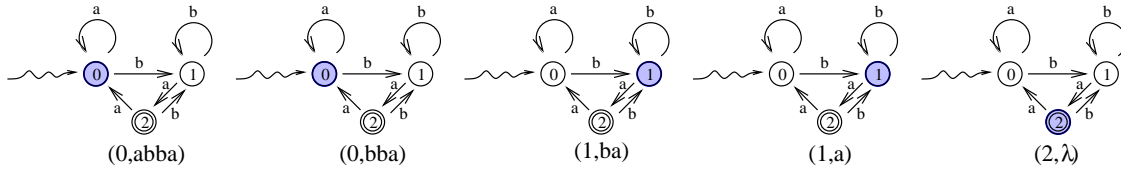


Figure 3: Computation of M accepting $abba$. For simplicity, the states are labeled 0, 1, 2 instead of q_0, q_1, q_2 . The final state is doubly circled. The figures show the sequence of **ids**. The current state is indicated in blue, and the current **id** is underneath the figure. Note that the last state is final.

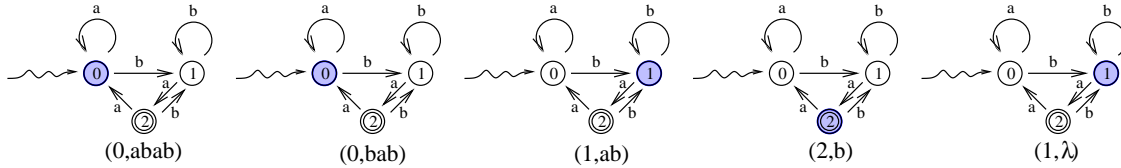


Figure 4: Computation of M rejecting $abab$. Note that the last state is not final.