Encodings and Problems

1 Alphabets and Strings

In this section, we go over the formal definitions of strings and some common string operations.

Definition (Alphabet, symbol/character)**.** An *alphabet* is a non-empty, finite set, and is usually denoted by Σ. The elements of Σ are called *symbols* or *characters*.

Example (Unary alphabet)**.** A unary alphabet consists of one symbol. A common choice for that symbol is `1`. So an example of a unary alphabet is $\Sigma = \{\texttt{1}\}$.

Example (Binary alphabet)**.** A binary alphabet consists of two symbols. Often we represent those symbols using `0` and `1`. So an example of a binary alphabet is $\Sigma = \{\texttt{0}, \texttt{1}\}$. Another example of a binary alphabet is $\Sigma = \{\texttt{a}, \texttt{b}\}$ where `a` and `b` are the symbols.

Example (Ternary alphabet)**.** A ternary alphabet consists of three symbols. So $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}\}$ and $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ are examples of ternary alphabets.

Definition (String/word, empty string)**.** Given an alphabet Σ, a *string* (or *word*) over Σ is a (possibly infinite) sequence of symbols, written as $a_1a_2a_3\ldots$, where each $a_i \in \Sigma$. The string with no symbols is called the *empty string* and is denoted by ϵ.

Example (Strings over the unary alphabet)**.** For $\Sigma = \{\texttt{1}\}$, the following is a list of 6 strings over Σ:

$$\epsilon,\ \texttt{1},\ \texttt{11},\ \texttt{111},\ \texttt{1111},\ \texttt{11111}.$$

Furthermore, the infinite sequence $\texttt{111111}\ldots$ is also a string over Σ.

Example (Strings over the binary alphabet)**.** For $\Sigma = \{\texttt{0}, \texttt{1}\}$, the following is a list of 8 strings over Σ:

$$\epsilon,\ \texttt{0},\ \texttt{1},\ \texttt{00},\ \texttt{01},\ \texttt{10},\ \texttt{11},\ \texttt{000}.$$

The infinite strings $\texttt{000000}\ldots$, $\texttt{111111}\ldots$ and $\texttt{010101}\ldots$ are also examples of strings over Σ.

Remark (Strings and quotation marks)**.** In our notation of a string, we do not use quotation marks. For instance, we write `1010` rather than "`1010`", even though the latter notation using the quotation marks is the standard one in many programming languages. Occasionally, however, we may use quotation marks to distinguish a string like "`1010`" from another type of object with the representation 1010 (e.g. the binary *number* 1010).

Definition (Length of a string)**.** The *length of a string* w, denoted $|w|$, is the number of symbols in w. If w has an infinite number of symbols, then the length is undefined.

Example (Lengths of `01001` and ϵ)**.** Let $\Sigma = \{\texttt{0}, \texttt{1}\}$. The length of the word `01001`, denoted by $|\texttt{01001}|$, is equal to 5. The length of ϵ is 0.

Definition (Star operation on alphabets)**.** Let Σ be an alphabet. We denote by Σ^* the set of *all* strings over Σ consisting of finitely many symbols. Equivalently, using set notation,

$$\Sigma^* = \{a_1 a_2 \ldots a_n : \ n \in \mathbb{N}, \text{and } a_i \in \Sigma \text{ for all } i\}.$$

Example ($\{\texttt{a}\}^*$)**.** For $\Sigma = \{\texttt{a}\}$, Σ^* denotes the set of all finite-length words consisting of `a`'s. So

$$\{\texttt{a}\}^* = \{\epsilon, \texttt{a}, \texttt{aa}, \texttt{aaa}, \texttt{aaaa}, \texttt{aaaaa}, \ldots\}.$$

Example ($\{\texttt{0}, \texttt{1}\}^*$)**.** For $\Sigma = \{\texttt{0}, \texttt{1}\}$, Σ^* denotes the set of all finite-length words consisting of `0`'s and `1`'s. So

$$\{\texttt{0}, \texttt{1}\}^* = \{\epsilon, \texttt{0}, \texttt{1}, \texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}, \texttt{000}, \texttt{001}, \texttt{010}, \texttt{011}, \texttt{100}, \texttt{101}, \texttt{110}, \texttt{111}, \ldots\}.$$

Note (Finite vs infinite strings)**.** We will use the words "string" and "word" to refer to a finite-length string/word. When we want to talk about infinite-length strings, we will explicitly use the word "infinite".

Note (Size of Σ^*)**.** By Definition (Alphabet, symbol/character), an alphabet Σ cannot be the empty set. This implies that Σ^* is an infinite set since there are infinitely many strings of finite length over a non-empty Σ. We will later see that Σ^* is always *countably* infinite.

Definition (Reversal of a string)**.** The *reversal of a string* $w = a_1 a_2 \ldots a_n$, denoted w^R, is the string $w^R = a_n a_{n-1} \ldots a_1$.

Example (Reversal of `01001`)**.** The reversal of `01001` is `10010`.

Example (Reversal of `1`)**.** The reversal of `1` is `1`.

Example (Reversal of ϵ)**.** The reversal of ϵ is ϵ.

Definition (Concatenation of strings)**.** The *concatenation of strings* u and v in Σ^*, denoted by uv or $u \cdot v$, is the string obtained by joining together u and v.

Example (Concatenation of `101` and `001`)**.** If $u = \texttt{101}$ and $v = \texttt{001}$, then $uv = \texttt{101001}$.

Example (Concatenation of `101` and ϵ)**.** If $u = \texttt{101}$ and $v = \epsilon$, then $uv = \texttt{101}$.

Example (Concatenation of ϵ and ϵ)**.** If $u = \epsilon$ and $v = \epsilon$, then $uv = \epsilon$.

Definition (Powers of a string)**.** For $n \in \mathbb{N}$, the n'th *power of a string* u, denoted by u^n, is the word obtained by concatenating u with itself n times.

Example (Third power of `101`)**.** If $u =$ `101` then $u^3 =$ `101101101`.

Example (Zeroth power of a string)**.** For any string u, $u^0 = \epsilon$.

Definition (Substring)**.** We say that a string u is a *substring* of string w if $w = xuy$ for some strings x and y.

Example (Various substring examples)**.** The string `101` is a substring of `11011` and also a substring of `0101`. On the other hand, it is not a substring of `1001`.

Every string is a substring of itself. And the empty string is a substring of any other string.

Definition (Prefix, suffix)**.** For strings x and y, we say that y is a *prefix* of x if there exists a string z such that $x = yz$. If $z \neq \epsilon$, then y is a *proper prefix* of x.

We say that y is a *suffix* of x if there exists a string z such that $x = zy$. If $z \neq \epsilon$, then y is a *proper suffix* of x.

Example (Various prefix and suffix examples)**.** Let $\Sigma = \{$`a`, `b`$\}$. Then if $x =$ `abbbab` and $y =$ `abb`, y is a proper prefix of x. Every string is a prefix of itself.

If $x =$ `abbbab` and $y =$ `bab`, y is a proper suffix of x. Every string is a suffix of itself.

The empty string is a prefix and suffix of every other string.

2 Encodings

Encodings allow us to go from an arbitrary object to a string representation of the object. We give the formal definition as well as various examples below.

Definition (Encoding elements of a set)**.** Let A be a set and let Σ be an alphabet. An *encoding* of the elements of A, using Σ, is an injective function $\text{Enc} : A \to \Sigma^*$. We denote the encoding of $a \in A$ by $\langle a \rangle$.

If $w \in \Sigma^*$ is such that there is some $a \in A$ with $w = \langle a \rangle$, then we say w is a *valid encoding* of an element in A.

A set that can be encoded is called *encodable*.[1]

Example (Decimal encoding of naturals)**.** When we (humans) communicate numbers among ourselves, we usually use the base-10 representation, which corresponds to an encoding of $\mathbb{N}$ using the alphabet $\Sigma = \{$`0`, `1`, `2`, $\dots$, `9`$\}$. For example, we encode the number four as `4` and the number twelve as `12`.

Example (Binary encoding of naturals)**.** As you know, every number has a base-2 representation (which is also known as the binary representation). This representation corresponds to an encoding of $\mathbb{N}$ using the alphabet $\Sigma = \{$`0`, `1`$\}$. For example, four is encoded as `100` and twelve is encoded as `1100`.

[1]Not every set is encodable. Can you figure out exactly which sets are encodable?

Example (Binary encoding of integers)**.** An integer is a natural number together with a sign, which is either negative or positive. Let $\text{Enc} : \mathbb{N} \to \{\texttt{0}, \texttt{1}\}^*$ be any binary encoding of $\mathbb{N}$. Then we can extend this encoding to an encoding of $\mathbb{Z}$, by defining $\text{Enc}' : \mathbb{Z} \to \{\texttt{0}, \texttt{1}\}^*$ as follows:

$$\text{Enc}'(x) = \begin{cases} \texttt{0}\text{Enc}(x) & \text{if } x \geq 0, \\ \texttt{1}\text{Enc}(|x|) & \text{if } x < 0. \end{cases}$$

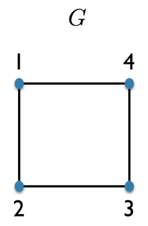
Effectively, this encoding of integers takes the encoding of natural numbers and precedes it with a bit indicating the integer's sign.

Example (Unary encoding of naturals)**.** It is possible (and straightforward) to encode the natural numbers using the alphabet $\Sigma = \{\texttt{1}\}$ as follows. Let $\text{Enc}(n) = \texttt{1}^n$ for all $n \in \mathbb{N}$.

Example (Ternary encoding of pairs of naturals)**.** Suppose we want to encode the set $A = \mathbb{N} \times \mathbb{N}$ using the alphabet $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}\}$. One way to accomplish this is to make use of a binary encoding $\text{Enc}' : \mathbb{N} \to \{\texttt{0}, \texttt{1}\}^*$ of the natural numbers. With Enc' in hand, we can define $\text{Enc} : \mathbb{N} \times \mathbb{N} \to \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$ as follows. For $(x, y) \in \mathbb{N} \times \mathbb{N}$, $\text{Enc}(x, y) = \text{Enc}'(x)\texttt{2}\text{Enc}'(y)$. Here the symbol 2 acts as a separator between the two numbers. To make the separator symbol advertise itself as such, we usually pick a symbol like $ rather than 2. So the ternary alphabet is often chosen to be $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{\$}\}$.

Example (Binary encoding of pairs of naturals)**.** Having a ternary alphabet to encode pairs of naturals was convenient since we could use the third symbol as a separator. It is also relatively straightforward to take that ternary encoding and turn it into a binary encoding, as follows. Encode every element of the ternary alphabet in binary using two bits. For instance, if the ternary alphabet is $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{\$}\}$, then we could encode 0 as 00, 1 as 01 and $ as 11. This mapping allows us to convert any encoded string over the ternary alphabet into a binary encoding. For example, a string like $0$1 would have the binary representation 11001101.

Example (Ternary encoding of graphs)**.** Let A be the set of all undirected graphs.[2] Every graph $G = (V, E)$ can be represented by its $|V|$ by $|V|$ adjacency matrix. In this matrix, every row corresponds to a vertex of the graph, and similarly, every column corresponds to a vertex of the graph. The (i, j)'th entry contains a 1 if $\{i, j\}$ is an edge, and contains a 0 otherwise. Below is an example.



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 1 & 0 & 1 & 0 \end{array}$$

Such a graph can be encoded using a ternary alphabet as follows. Take the adjacency matrix of the graph, view each row as a binary string, and concatenate all the rows by putting a separator symbol between them. The encoding of the above example would be

$$\langle G \rangle = \texttt{0101\$1010\$0101\$1010}.$$

Example (Encoding of Python functions)**.** Let A be the set of all functions in the programming language Python. Whenever we type up a Python function in a code editor, we are creating a string representation/encoding of the function, where the alphabet is all the Unicode symbols.[3] For example, consider a Python function named `absValue`, which we can write as

[2]We will define graphs formally in a future chapter, however, we assume you are already familiar with the concept.

[3]https://en.wikipedia.org/wiki/Unicode

```
def absValue(N):
    if (N < 0): return -N
    else: return N
```

By writing out the function, we have already encoded it. More specifically, $\langle \text{abs\_value} \rangle$ is the following string.

```
def absValue(N):\n    if (N < 0): return -N\n    else: return N
```

Exercise (Unary encoding of integers)**.** Describe an encoding of $\mathbb{Z}$ using the alphabet $\Sigma = \{\texttt{1}\}$.

Hint. Think about a bijection between integers and naturals.

Solution. Let $\text{Enc} : \mathbb{Z} \to \{\texttt{1}\}^*$ be defined as follows:

$$\text{Enc}(x) = \begin{cases} \texttt{1}^{2x-1} & \text{if } x > 0, \\ \texttt{1}^{-2x} & \text{if } x \leq 0. \end{cases}$$

This solution is inspired by thinking of a bijection between integers and naturals. Indeed, the function $f : \mathbb{Z} \to \mathbb{N}$ defined by

$$f(x) = \begin{cases} 2x - 1 & \text{if } x > 0, \\ -2x & \text{if } x \leq 0, \end{cases}$$

is such a bijection. ■

3 Computational Problems

In general, a *computational problem* specifies a set of instances (possible inputs) as well as the conditions for a solution (output) to be a satisfactory solution for a given instance. In this section, we define two kinds of computational problems: function problems and decision problems.

Definition (Function problem)**.** Let Σ be an alphabet. Any function $f : \Sigma^* \to \Sigma^*$ is called a *function problem* over the alphabet Σ.

Example (Addition as a function problem)**.** Consider the function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined as $g(x, y) = x + y$. This is a function that expresses the addition problem in naturals. We can view g as a function problem over an alphabet Σ once we fix an encoding of the domain $\mathbb{N} \times \mathbb{N}$ using Σ and an encoding of the co-domain $\mathbb{N}$ using Σ. For convenience, we take $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{\$}\}$. Let Enc be the encoding of $\mathbb{N} \times \mathbb{N}$ as described in Example (Ternary encoding of pairs of naturals). Let Enc' be the encoding of $\mathbb{N}$ as described in Example (Binary encoding of naturals). Note that Enc' leaves the symbol \$ unused in the encoding. We now define the function problem f corresponding to g. If $w \in \Sigma^*$ is a word that corresponds to a valid encoding of a pair of numbers (x, y) (i.e., $\text{Enc}(x, y) = w$), then define $f(w)$ to be $\text{Enc}'(x + y)$. If $w \in \Sigma^*$ is *not* a word that corresponds to a valid encoding of a pair of numbers (i.e., w is not in the image of Enc), then define $f(w)$ to be \$. In the co-domain, the \$ symbol serves as an "error" indicator.

Important (Function problem as mapping instances to solutions)**.** A function problem is often derived from a function $g : I \to S$, where I is a set of objects called *instances* and S is a set of objects called *solutions*. The derivation is done through encodings $\text{Enc} : I \to \Sigma^*$ and $\text{Enc}' : S \to \Sigma^*$. With these encodings, we can create the function problem $f : \Sigma^* \to \Sigma^*$. In particular, if $w = \langle x \rangle$ for some $x \in I$, then we define $f(w)$ to be $\text{Enc}'(g(x))$.

$$\begin{tikzcd} I \arrow[r, "g"] \arrow[d, "\text{Enc}"'] & S \arrow[d, "\text{Enc}'"] \\ \Sigma^* \arrow[r, "f"] & \Sigma^* \end{tikzcd}$$

One thing we have to be careful about is defining $f(w)$ for a word $w \in \Sigma^*$ that does not correspond to an encoding of an object in I (such a word does not correspond to an instance of the function problem). To handle this, we can identify one of the strings in Σ^* as an *error* string and define $f(w)$ to be that string.

Definition (Decision problem)**.** Let Σ be an alphabet. Any function $f : \Sigma^* \to \{0, 1\}$ is called a *decision problem* over the alphabet Σ. The co-domain of the function is not important as long as it has two elements. One can think of the co-domain as $\{\text{No}, \text{Yes}\}$, $\{\text{False}, \text{True}\}$ or $\{\text{Reject}, \text{Accept}\}$.

Example (Primality testing as a decision problem)**.** Consider the function $g : \mathbb{N} \to \{\text{False}, \text{True}\}$ such that $g(x) = \text{True}$ if and only if x is a prime number. We can view g as a decision problem over an alphabet Σ once we fix an encoding of the domain $\mathbb{N}$ using Σ. Take $\Sigma = \{\texttt{0}, \texttt{1}\}$. Let Enc be the encoding of $\mathbb{N}$ as described in Example (Binary encoding of naturals). We now define the decision problem f corresponding to g. If $w \in \Sigma^*$ is a word that corresponds to an encoding of a prime number, then define $f(w)$ to be 1. Otherwise, define $f(w)$ to be 0. (Note that in the case of $f(w) = 0$, either w is the encoding of a composite number, or w is not a valid encoding of a natural number.)

Note (Decision problem as mapping instances to 0 or 1s)**.** As with a function problem, a decision problem is often derived from a function $g : I \to \{0, 1\}$, where I is a set of instances. The derivation is done through an encoding $\text{Enc} : I \to \Sigma^*$, which allows us to define the decision problem $f : \Sigma^* \to \{0, 1\}$. Any word $w \in \Sigma^*$ that does not correspond to an encoding of an instance is mapped to 0 by f.

Instances that map to 1 are often called yes-instances and instances that map to 0 are often called no-instances.

4 Decision Problems as Languages

In this section, we introduce the concept of a *language*, which is a very standard/common way of representing decision problems.

Definition (Language)**.** Any (possibly infinite) subset $L \subseteq \Sigma^*$ is called a *language* over the alphabet Σ.

Example (Language of even length strings)**.** Let Σ be an alphabet. Then $L = \{w \in \Sigma^* : |w| \text{ is even}\}$ is a language.

Example (A language with one word)**.** Let $\Sigma = \{\texttt{0}, \texttt{1}\}$. Then $L = \{\texttt{101}\}$ is a language.

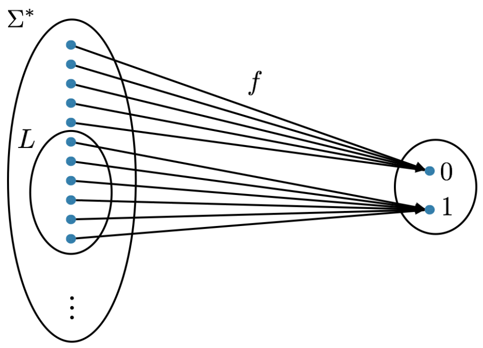
Example (Σ^* as a language)**.** Let Σ be an alphabet. Then $L = \Sigma^*$ is a language.

Example (Empty set as a language)**.** Let Σ be an alphabet. Then $L = \varnothing$ is a language.

Note (Size of a language)**.** Since a language is a set, the *size of a language* refers to the size of that set. A language can have finite or infinite size. This is not in conflict with the fact that every language consists of *finite-length* strings.

Remark ($\{\epsilon\}$ vs $\varnothing$)**.** The language $\{\epsilon\}$ is not the same language as $\varnothing$. The former has size 1 whereas the latter has size 0.

Important (Correspondence between decision problems and languages)**.** There is a one-to-one correspondence between decision problems and languages. Let $f : \Sigma^* \to \{0, 1\}$ be some decision problem. Now define $L \subseteq \Sigma^*$ to be the set of all words in Σ^* that f maps to 1. This L is the language corresponding to the decision problem f. Similarly, if you take any language $L \subseteq \Sigma^*$, we can define the corresponding decision problem $f : \Sigma^* \to \{0, 1\}$ as $f(w) = 1$ if and only if $w \in L$. We consider the set of languages and the set of decision problems to be the same set of objects.



It is important to get used to this correspondence since we will often switch between talking about languages and talking about decision problems without explicit notice.

Remark (The default alphabet)**.** When talking about decision/function problems or languages, if the alphabet is not specified, it is assumed to be the binary alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

Remark (Why languages?)**.** You may be wondering why we make the definition of a language rather than always viewing a decision problem as a function from Σ^* to $\{0, 1\}$. The reason is that some operations and notation are more conveniently expressed using sets rather than functions, and therefore in certain situations, it can be nicer to think of decision problems as sets.

5 Language Operations

In this section we go over some common operations on languages. Many of them are derived from string operations that we covered earlier in the chapter.

Definition (Reversal of a language)**.** The *reversal of a language* $L \subseteq \Sigma^*$, denoted L^R, is the language

$$L^R = \{w^R \in \Sigma^* : w \in L\}.$$

Example (Reversal of $\{\epsilon, 1, 1010\}$)**.** The reversal of the language $\{\epsilon, \mathtt{1}, \mathtt{1010}\}$ is $\{\epsilon, \mathtt{1}, \mathtt{0101}\}$.

Definition (Concatenation of languages)**.** The *concatenation of languages* $L_1, L_2 \subseteq \Sigma^*$, denoted L_1L_2 or $L_1 \cdot L_2$, is the language

$$L_1L_2 = \{uv \in \Sigma^* : u \in L_1, v \in L_2\}.$$

Example (Concatenation of $\{\epsilon, \mathtt{1}\}$ and $\{\mathtt{0}, \mathtt{01}\}$)**.** The concatenation of languages $L_1 = \{\epsilon, \mathtt{1}\}$ and $L_2 = \{\mathtt{0}, \mathtt{01}\}$ is the language

$$\{\mathtt{0}, \mathtt{01}, \mathtt{10}, \mathtt{101}\}.$$

Example (Concatenation of $\{\mathtt{0}\}$ and Σ^*)**.** The concatenation of $L_1 = \{\mathtt{0}\}$ with $L_2 = \Sigma^* = \{\mathtt{0}, \mathtt{1}\}^*$ is the set of all binary strings that start with a 0 symbol.

Example (Concatenation with $\{\epsilon\}$)**.** The concatenation of the language $\{\epsilon\}$ and any language L is L.

Example (Concatenation with $\varnothing$)**.** The concatenation of the language $\varnothing$ with any language L is $\varnothing$.

Definition (Powers of a language)**.** For $n \in \mathbb{N}$, the n'th *power of a language* $L \subseteq \Sigma^*$, denoted L^n, is the language obtained by concatenating L with itself n times, that is,[4]

$$L^n = \underbrace{L \cdot L \cdot L \cdots L}_{n \text{ times}}.$$

Equivalently,

$$L^n = \{u_1 u_2 \cdots u_n \in \Sigma^* : u_i \in L \text{ for all } i \in \{1, 2, \ldots, n\}\}.$$

Example ($\{\texttt{1}\}^3$)**.** The 3rd power of $\{\texttt{1}\}$ is the language $\{\texttt{111}\}$.

Example ($\{\epsilon, \texttt{1}\}^3$)**.** The 3rd power of $\{\epsilon, \texttt{1}\}$ is the language $\{\epsilon, \texttt{1}, \texttt{11}, \texttt{111}\}$.

Example (L^0)**.** The 0th power of any language L is the language $\{\epsilon\}$, even when $L = \varnothing$.

Definition (Star operation on a language)**.** The *star of a language* $L \subseteq \Sigma^*$, denoted L^*, is the language

$$L^* = \bigcup_{n \in \mathbb{N}} L^n.$$

Equivalently,

$$L^* = \{u_1 u_2 \cdots u_n \in \Sigma^* : n \in \mathbb{N}, u_i \in L \text{ for all } i \in \{1, 2, \ldots, n\}\}.$$

Note that we always have $\epsilon \in L^*$. Sometimes we may prefer to not include ϵ by default, so we define

$$L^+ = \bigcup_{n \in \mathbb{N}^+} L^n.$$

Example (Σ^*)**.** Given an alphabet Σ, consider the language $L = \Sigma \subseteq \Sigma^*$[5]. Then L^* is equal to Σ^*.

Example ($\{\texttt{00}\}^*$)**.** If $L = \{\texttt{00}\}$, then L^* is the language consisting of all words containing an even number of `0`'s and no other symbol.

Example ($(\{\texttt{00}\}^*)^*$)**.** Let L be the language consisting of all words containing an even number of `0`'s and no other symbol. Then $L^* = L$.

Exercise (Can you distribute star over intersection?)**.** Prove or disprove: If $L_1, L_2 \subseteq \{\texttt{a}, \texttt{b}\}^*$ are languages, then $(L_1 \cap L_2)^* = L_1^* \cap L_2^*$.

Hint. Disprove the statement by providing a counterexample.

Solution. We disprove the statement by providing a counterexample. Let $L_1 = \{\texttt{a}\}$ and $L_2 = \{\texttt{aa}\}$. Then $L_1 \cap L_2 = \varnothing$, and so $(L_1 \cap L_2)^* = \{\epsilon\}$. On the other hand, $L_1^* \cap L_2^* = L_2^* = \{\texttt{aa}\}^*$. ■

[4]We can omit parentheses as the order in which the concatenation $\cdot$ is applied does not matter.

[5]Technically L is a set of strings and Σ is a set of symbols, so the equality notation is not entirely accurate. Hopefully the intention is clear however: symbols can be viewed as length-1 strings.

Exercise (Can you distribute concatenation over union and intersection?)**.** Prove or disprove the following:

1. If $A, B, C \subseteq \{\mathtt{a}, \mathtt{b}\}^*$ are languages, then $A(B \cup C) = AB \cup AC$.
2. If $A, B, C \subseteq \{\mathtt{a}, \mathtt{b}\}^*$ are languages, then $A(B \cap C) = AB \cap AC$.

Hint. Prove the first using a double-containment argument. Give a counter-example for the second.

Solution.

1. Let $L = A(B \cup C)$ and $R = AB \cup AC$. We prove $L = R$ by a double-containment argument.

   We start with $L \subseteq R$. In order to establish this, we will show that if $w \in L$ then $w \in R$. So consider an arbitrary $w \in L$. Then by definition of concatenation, $w = uv$, where $u \in A$ and $v \in B \cup C$. Therefore v is in either B or C. Without loss of generality, assume $v \in B$ (the argument when $v \in C$ is analogous). Then $w \in AB$, and therefore also in R.

   Next we prove $R \subseteq L$. In order to establish this, we will show that if $w \in R$ then $w \in L$. So consider an arbitrary $w \in R$. This means either $w \in AB$ or $w \in AC$. Without loss of generality, assume $w \in AB$ (the argument when $w \in AC$ is analogous). Then using the definition of concatenation, we can write w as uv where $u \in A$ and $v \in B$. This implies that $v \in B \cup C$. So once again, by the definition of concatenation, we conclude $w = uv$ is in $A(B \cup C) = L$.

2. This statement is false. First, try to write out a proof similar to the previous part, and see if you can spot where the argument goes wrong.

   As a counter-example, consider $A = \Sigma^* = \{\mathtt{a}, \mathtt{b}\}^*$, $B = \{\mathtt{a}\}$ and $C = \{\epsilon\}$ so that $B \cap C = \varnothing$. Then $A(B \cap C) = \Sigma^* \cdot \varnothing = \varnothing$. However, $AB \cap AC = (\Sigma^* \cdot \{\mathtt{a}\}) \cap (\Sigma^* \cdot \{\epsilon\}) = \Sigma^* \cdot \{\mathtt{a}\} \neq \varnothing$.

■

Exercise (Can you interchange star and reversal?)**.** Is it true that for any language L, $(L^*)^R = (L^R)^*$? Prove your answer.

Hint. The statement is true. To prove it, argue both $(L^*)^R \subseteq (L^R)^*$ and $(L^R)^* \subseteq (L^*)^R$.

Solution. We will prove that for any language L, $(L^*)^R = (L^R)^*$. To do this, we will first argue $(L^*)^R \subseteq (L^R)^*$ and then argue $(L^R)^* \subseteq (L^*)^R$.

To show the first inclusion, it suffices to show that any $w \in (L^*)^R$ is also contained in $(L^R)^*$. We do so now. Take an arbitrary $w \in (L^*)^R$. Then for some $n \in \mathbb{N}$, $w = (u_1 u_2 \dots u_n)^R$, where $u_i \in L$ for each $i \in \{1, 2, \dots, n\}$. Note that $w = (u_1 u_2 \dots u_n)^R = u_n^R u_{n-1}^R \dots u_1^R$, and $u_i^R \in L^R$ for each i. Therefore $w \in (L^R)^*$.

To show the second inclusion, it suffices to show that any $w \in (L^R)^*$ is also contained in $(L^*)^R$. We do so now. Take an arbitrary $w \in (L^R)^*$. This means that for some $n \in \mathbb{N}$, $w = v_1 v_2 \dots v_n$, where $v_i \in L^R$ for each $i \in \{1, 2, \dots, n\}$. For each i, define $u_i = v_i^R$ (and so $u_i^R = v_i$). Note that each $u_i \in L$ because $v_i \in L^R$. We can now rewrite w as $w = u_1^R u_2^R \dots u_n^R$, which is equal to $(u_n u_{n-1} \cdots u_1)^R$. Since each $u_i \in L$, this shows that $w \in (L^*)^R$.

Since we have shown both $(L^*)^R \subseteq (L^R)^*$ and $(L^R)^* \subseteq (L^*)^R$, we conclude that $(L^*)^R = (L^R)^*$. ■

Exercise (Is single star equivalent to double star?)**.** Prove or disprove: For any language L, $L^* = (L^*)^*$.

Hint. Prove the statement using a double-containment argument.

Solution. We prove the statement with double-containment argument, that is, we will show $L^* \subseteq (L^*)^*$ and $(L^*)^* \subseteq L^*$.

We start with $L^* \subseteq (L^*)^*$. Let $K = L^*$, so we are trying to show $K \subseteq K^*$. This is true since by definition, $K^* = K^0 \cup K \cup K^2 \cup \ldots$.

For the other direction, let's take an arbitrary word $w \in (L^*)^*$ and show that w must be in L^*. Since $w \in (L^*)^*$, by definition, $w = v_1 v_2 \ldots v_n$ for some $v_i \in L^*$ and $n \in \mathbb{N}$. Since each v_i is in L^*, each can be written as $u_{i1} u_{i2} \ldots u_{in_i}$ for some $u_{ij} \in L$ and $n_i \in \mathbb{N}$. Therefore

$$w = u_{11} u_{12} \ldots u_{1n_1} u_{21} u_{22} \ldots u_{2n_2} \quad \ldots$$

where each u_{ij} is in L. This means $w \in L^*$, as desired. ■

Definition (Set of all languages)**.** We denote by ALL the set of all languages (over the default alphabet $\Sigma = \{0, 1\}$).

6 Check Your Understanding

Problem. 1. Given an alphabet Σ, what is the definition of Σ^*?

2. What is the definition of a *language*?
3. Is $\varnothing$ a language? If it is, what is the decision problem corresponding to it?
4. Is Σ^* a language? If it is, what is the decision problem corresponding to it?
5. Given two languages L_1 and L_2, what are the definitions of $L_1 L_2$ and L_1^*?
6. True or false: The language $\{011, 101, 110\}^*$ is the set of all binary strings containing twice as many 1's as 0's.
7. True or false: The set $\{\varnothing\}$ is a language, where $\varnothing$ denotes the empty set.
8. True or false: An infinite language must contain infinite-length strings.
9. True or false: A language can contain infinite-length strings.
10. Define finite languages L_1 and L_2 such that $|L_1 L_2| < |L_1||L_2|$.
11. Define finite languages L_1 and L_2 such that $|L_1 L_2| = |L_1||L_2|$.
12. True or false: For any language L and any $n \in \mathbb{N}$, we have $L^n \subseteq L^{n+1}$.
13. True or false: For any language L and any $n \in \mathbb{N}$, we have $L^n \subseteq L^*$.
14. Let Σ be an alphabet. For which languages $L \subseteq \Sigma^*$ is it true that $L \cdot L$ is infinite?
15. Let Σ be an alphabet. For which languages $L \subseteq \Sigma^*$ is it true that L^* is infinite?
16. What is the definition of an *encodable* set? Which sets are encodable? How do we deal with sets that are not encodable?
17. What is the motivation behind defining an encoding as a mapping to finite-length strings as opposed to both finite and infinite-length strings?
18. What is the definition of a *function problem*? Give one example of a function problem.
19. What is the definition of a *decision problem*? Give one example of a decision problem.
20. Briefly explain the correspondence between decision problems and languages.
21. Write down the primality testing decision problem as a language.
22. Why do we focus on decision problems in theoretical computer science?

7 High-Order Bits

Important. Here are the important things to keep in mind from this chapter.

1. All the definitions and the notation introduced in this are fundamental, and we will use them liberally without reminders. As we move to more advanced topics, it is important that the definitions in this chapter do not add to the mental burden and make things overwhelming for you. Even when you are totally comfortable with the definitions, we will be covering concepts that are quite challenging. Therefore, you want to be able to spend all your mental energy on digesting the new material without getting confused by the definitions and terminology from previous chapters. If you invest the time to really understanding the definitions now, it will save you time later.

2. There is only one data type in theoretical computer science: strings. Every other type of data (e.g. numbers, images, graphs, videos) can be encoded with strings. Algorithms/computers work with these encodings.

3. A decision problem is a special kind of function problem. In a decision problem, for every possible input, the output is either True or False. Our focus for the next few chapters will be on decision problems.

4. There is a one-to-one correspondence between decision problems and languages. It is important to be able to easily switch between a language and the corresponding decision problem in your mind.