

Alphabets and Languages

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)



Alphabets and Languages

Symbols and Strings

A **string** over an alphabet is a finite sequence of symbols from that alphabet. Note that the word "sequence" was used rather than "set;" the order of symbols in a string is significant.

Examples of strings over the alphabets above include:

- *abba* is a string over Σ_1
- *supercalafragilisticexpalidocius* is a string over Σ_2
- 10010100 is a string over Σ_3

The **empty string**, denoted Λ , is the string with no symbols.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

Symbols and Strings

An **alphabet** is a finite set of symbols. These symbols can be anything, but usually they will be symbols like a , b , 0, 1, #, etc. The symbol Σ is usually used to denote the alphabet.

Examples of alphabets include:

- $\Sigma_1 = \{a, b\}$
- $\Sigma_2 = \{a, b, c, \dots, z\}$
- $\Sigma_3 = \{0, 1\}$

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

Symbols and Strings

A word about strings and symbols...

What is a ? Is it a string or a symbol?

It can be either and usually we'll have to tell which it is from the context. There is a convention that helps us here, however. Symbols from the beginning of our Roman alphabet are usually just that, symbols.

The lowercase Greek letter σ is used to represent a symbol; sort of a symbol variable.

The letters w , x , y , and z often are used to represent strings; these are string variables.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Symbols and Strings

The **length** of a string is the number of symbols in the sequence comprising the string. We denote the length of a string w as $|w|$.

If w is the string *abba* then $|w| = |abba| = 4$.

Notice that, since Λ has no symbols in its sequence, $|\Lambda| = 0$.

We can refer to symbols in specific positions within a string:

- $w(1) = a$
- $w(2) = w(3) = b$
- $w(4) = a$

Notice that, unlike array notation in C or C++, the first index of a string is 1 not 0.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Symbols and Strings

We will use the notation w^i to indicate a string of length $i|w|$ which is obtained by repeatedly concatenating w on to an initially empty string i times.

$$\begin{aligned} w^0 &= \Lambda \\ w^{i+1} &= w^i \circ w, \quad i \geq 0 \end{aligned}$$

For example, if $w = abbab$ then $w^3 = abbababbababbab$.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Symbols and Strings

We can create new strings from existing strings using **concatenation**. The concatenation of two strings x and y is written xy or $x \circ y$.

For example, *abba* \circ *baaba* yields the string *abbabaaba*.

Obviously $|xy| = |x| + |y|$.

Concatenation is associative: given the three strings x , y and z we have that $(xy)z = x(yz)$.

[Since parentheses are often used to group symbols or strings, it is customary to exclude them from alphabets. The string (x) is the same as the string x .]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Symbols and Strings

A string y is a **substring** of a string w if there are strings x and z such $w = xyz$. Either or both of the strings x and z may be the empty string Λ .

Every string is a substring of itself.

Λ is a substring of every string, including Λ .

The string x is a **prefix** of w if there is a string y such that $w = xy$.

The string z is a **suffix** of w if there is a string y such that $w = yz$.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Symbols and Strings

The **reversal** of a string w is denoted w^R or $reverse(w)$ and has the property that

$$w^R(i) = w(n-i+1), \quad i = 1, \dots, n$$

where $n = |w|$. This just means that w^R is w backward.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

While alphabets are finite, languages can be finite or infinite. The language $L_1 = \Sigma^*$ is an infinite language while $L_2 = \{abb, baa\}$ is a finite language.

Note that while an alphabet Σ is a set, we usually assume that some arbitrary order has been imposed on it. Doing this lets us list strings in a language over this alphabet in a systematic way. When we list the strings in a language we usually list them in the following order:

- shorter strings are listed before longer strings,
- strings of the same length are listed **lexicographically**.

Two strings u and v are ordered lexicographically when u comes before v if $u(i) < v(i)$ where i is the first position where symbols in the strings differ.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

The **star closure** or **universal language** of the alphabet Σ is denoted Σ^* and is the set of all strings that can be formed using the symbols from Σ .

A **language** is a subset of Σ^* for some alphabet Σ .

The symbol \emptyset is used to denote the **empty language**, the language with no strings. Notice that this is different than the language $\{\Lambda\}$, the language that contains only the empty string.

Usually we'll describe a language with notation like

$$L = \{w \in \Sigma^* \mid w \text{ has some property } P\}$$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

For example, let $\Sigma = \{a, b\}$ and consider the language $L = \Sigma^*$. Then

$$L = \{\Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots\}$$

One advantage of doing this is that, even in the case of infinite languages that cannot be listed, the strings in the language can be surmised by the pattern of strings that are listed.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

Since languages are sets of strings, all the familiar set operations apply to them. Given languages L_1 and L_2 we can construct new languages:

$$L_1 \cup L_2, \quad L_1 \cap L_2$$

The **complement** of a language L is

$$L' = \Sigma^* - L$$

and is the language containing all strings (that can be formed from Σ) that are not in L .

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

The **Kleene star** (pronounced "KLAY-nee star") or **Kleene closure** of a language L is denoted L^* .

A string w is in L^* if

$$w = w_1 \circ w_2 \circ \dots \circ w_n$$

when $w \in \Sigma^*$ and

$$w_1, w_2, \dots, w_n \in L$$

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

The **concatenation of the languages** L_1 and L_2 is denoted

$$L = L_1 L_2$$

and is the set of all strings of the form uv where $u \in L_1$ and $v \in L_2$.

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)

Alphabets and Languages

[prev](#) | [slides](#) | [next](#)

Languages

The star closure of a set of symbols or strings contains strings of length zero or greater. There is another useful closure, the **positive** or **plus** closure, that acts just like the star closure except that strings in it are composed of one or more strings from the generating set.

For example, if $\Sigma = \{a, b\}$ then

$$\Sigma^* = \{ \Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots \}$$

$$\Sigma^+ = \{ a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots \}$$

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) [16](#)