Chapter 1

Sets, Tuples and Sequences

1.1 Sets

Set theory is perhaps the most fundamental mathematical theory, on which all of the mathematics is built. A *set* is usually taken to be a basic term and, as such, we do not define what a set is. We can imagine a set to be a collection of objects that is seen as a single unit. The objects in the set are called *elements*. Elements can be any type of objects (numbers, letters, symbols, functions, other sets. . .) and there is no requirement that all the elements in the set have to be of the same type.

The symbol $\in$ denotes membership of an element to a set. For example, if the number 1 is an element of the set A, we write $1 \in A$. Similarly, non-membership is denoted by $\notin$. Therefore, if the number 1 is not an element of a set B, we write $1 \notin B$.

Empty set is a special set that has no elements. We will denote the empty set by $\emptyset$.

Remark 1.1: Notation

As is standard in the literature, we will use capital letters such as A, B or X to denote sets, and lowercase letters, such as a, b, c to denote individual elements.

Remark 1.2: Notation for Some Commonly Used Sets

There are a few sets that we will encounter very frequently in our study. These are so important that they are given special names, and even special symbols. Thus, the set of *integer numbers* will be denoted by $\mathbb{Z}$. This is the set of all whole numbers, positive or negative $(0, -1, 1, -2, 2, -3, 3, \dots)$. The set of natural numbers will be denoted by $\mathbb{N}$. This is the set of all positive integer numbers, so $1, 2, 3, 4, \dots$. Unlike many people in computer science, we do not consider 0 to be a natural number (and we take certain pride in being "rebels" in this way). When we want to include 0, we will talk about the set of natural numbers and zero. Occasionally, we may use the set of *rational numbers* (denoted by $\mathbb{Q}$). This is the set of all numbers that can be written as a fraction where both the numerator and the denominator are integer numbers. Finally, the set of *real numbers* will be denoted by $\mathbb{R}$.

1.1.1 Describing Sets

Sets can be described in a number of ways. Firstly, we can describe a set by explicitly listing all the elements of the set between the curly brackets { and }. Thus a set that contains integer numbers 1, 2 and 3 can be denoted by $\{1, 2, 3\}$. This is, obviously, convenient only if the set in question only has a small number of elements. We would quickly run into trouble if we try to describe a set of all integer numbers less than 10000 in this way. Furthermore, describing infinite sets (such as, for example, a set of all even integer numbers) would be impossible using this method. Alternatively, we will sometimes describe sets by listing the first few members, followed by . . . , which indicates that the remaining elements of the set follow the same pattern. If the set is finite, we will also denote the last element of the set; otherwise, we will assume that the set is infinite. For example, the set

$$\{1, 2, 3, 4, \dots, 10000\}$$

denotes the set of all positive integer numbers from 1 to 10000. On the other hand, the set

$$\{1, 2, 3, 4, \dots\}$$

denotes the set of all positive integer numbers (hence, an infinite set of numbers).

However, even this way of representing the sets is not enough in most of the cases. For example, while it is reasonably easy to understand that the set

$$\{2, 4, 6, 8, 10, \dots\}$$

represents the set of all positive even integer numbers, the situation is less clear for the set

$$\{1, 4, 9, 16, \dots\}.$$

It takes some mental gymnastics to derive that this is the set of squares of integer numbers. Even more complicated situation is, for example, for a set

$$\{6, 12, 22, 36, \dots\}.$$

It is not at all obvious that these are the numbers of the form $2n + 4$, where n is a positive integer. For these (and many other sets), the easiest to understand and the most convenient way of describing a set is by denoting a property that its elements satisfy; the set itself would then consist of all elements that satisfy this property. For example, the set of all integer numbers less than 10000 can be described as

$$\{x | x \text{ is an integer and } x < 10000\}.$$

The elements of the above set are then all elements x that are (i) integers; and, (ii) numbers less than 10000. Another example is

$$\{n^2 | n \text{ is an integer }\},$$

which describes a set of all squares of integer numbers. Note that we have described the same set using the notation $\{1, 4, 9, 16, \dots\}$.

1.1.2 Subset and Equality of Sets

Definition 1.1: Subset

Set A is a *subset* of set B (denoted by $A \subset B$) if every element of A is also an element of B.

Thus, for example, if $A = \{1, 2\}$ and $B = \{1, 2, 3, 4\}$, then $A \subset B$. Note also that it is true that $A = \{1, 2, 3, 4\}$ is a subset of $B = \{1, 2, 3, 4\}$. The fact that $A \subset B$ does not mean that A and B cannot, in fact, be equal. It is also worth noting that the empty set is a subset of every set, and also that every set is a subset of itself.

Remark 1.3: Symbol for a Subset: $\subset$ or $\subseteq$?

In some literature, the symbol $\subseteq$ is used as a subset symbol, to emphasise that the two sets can be equal. There, the symbol $\subset$ is used for a *proper* subset. That is, a subset that is not equal to the set. Therefore, using that notation, $\{1, 2\} \subseteq \{1, 2, 3, 4\}$, $\{1, 2, 3, 4\} \subseteq \{1, 2, 3, 4\}$, $\{1, 2\} \subset \{1, 2, 3, 4\}$ but it is not true that $\{1, 2, 3, 4\} \subset \{1, 2, 3, 4\}$. We will, however, not use this notation. In the rare cases when we want to emphasise that A is a subset of B, but that A cannot be equal to B, we will do so using words.

In the discussion above, we mentioned the equality of sets. However, we have so far not defined what do we mean when we say that two sets are equal. Intuitively, two sets are equal when they have exactly the same elements. Which is to say, when every element of the first set is in also in the second set, but also every element of the second set is also in the first set. Note that this means exactly that the two sets are equal if the first set is a subset of the second set and the second set is a subset of the first set.

Definition 1.2: Equality of Sets

Set A is equal to set B if (i) every element of set A is also an element of the set B; and (ii) every element of set B is also an element of the set A. In other words, for sets A and B, $A = B$ if $A \subset B$ and $B \subset A$.

The definition of the equality of two sets in terms of each of them being a subset of the other is how we will usually show equality of sets. Showing directly that two sets have exactly the same elements is many times not so easy, and it is far easier to split proving the equality of sets A and B into two parts - the first part that proves that $A \subset B$ and the second part that proves that $B \subset A$.

In our discussion so far, we have not considered whether the elements in a set can be "repeated", i.e. whether $\{1, 1, 2, 3\}$ is a valid set and whether it is different from a set $\{1, 2, 3\}$. Now that we have introduced the notion of equality of two sets, we can see that these two sets are, indeed equal - exactly the same elements $1, 2$ and 3 are the elements of both sets. We could, therefore, allow repetition of elements in denoting a set - it wouldn't present us any problems, but it also wouldn't give us anything new. Therefore, in our discussion, we will always denote sets in a way that the elements in them are unique. Finally, the order in which we write the elements of a set also does not matter - sets $\{1, 2, 3\}$ and $\{3, 2, 1\}$ are equal. For this reason, we will usually write sets using some kind of 'natural' ordering of their elements.

1.1.3 Union and Intersection of Sets

Definition 1.3: Union of Two Sets

Union of two sets A and B (denoted by $A \cup B$) is a set that comprises of all the elements that are in A or in B (or in both).

Thus, $\{1, 2\} \cup \{3, 4\} = \{1, 2, 3, 4\}$ and

$$\{1, 2, 3\} \cup \{2, 3, 4\} = \{1, 2, 3, 4\}.$$

Note that the union of two sets is a *commutative*, meaning that $A \cup B = B \cup A$, and an *associative* operation, meaning that $(A \cup B) \cup C = A \cup (B \cup C)$.

We have defined a union between two sets, but in pretty much the same way we can define a union of finitely many, or even infinitely many sets.

Definition 1.4: Finite Union of Sets

Union of finitely many sets $A_1, A_2, \ldots, A_n$ (denoted by $A_1 \cup A_2 \cup \cdots \cup A_n$ or $\bigcup_{i=1}^{n} A_n$) is a set that comprises of all the elements that are in any of the sets $A_1, A_2, \ldots, A_n$.

Definition 1.5: Infinite Union of Sets

Union of infinitely many sets $A_1, A_2, \ldots$ (denoted by $\bigcup_{i=1}^{\infty} A_n$) is a set that comprises of all the elements that are in any of the sets $A_1, A_2, \ldots$.

For example, $\{1, 2\} \cup \{2, 3\} \cup \{3, 4\} = \{1, 2, 3, 4\}$ and

$$\{1, 2\} \cup \{1, 3\} \cup \{1, 4\} \cup \cdots = \bigcup_{i=1}^{\infty} \{1, i\} = \{1, 2, 3, 4, \ldots\}.$$

Definition 1.6: Intersection of Two Sets

Intersection of two sets A and B (denoted by $A \cap B$) is a set that comprises of all elements that are both in set A and in set B.

Thus, $\{1, 2, 3\} \cap \{2, 3, 4\} = \{2, 3\}$. The intersection of two sets is also commutative and associative.

In the same way that we defined a finite and infinite union of sets, we can also define a finite and infinite intersection of sets.

Definition 1.7: Finite Intersection of Sets

Intersection of finitely many sets $A_1, A_2, \ldots, A_n$ (denoted by $A_1 \cap A_2 \cap \cdots \cap A_n$ or $\bigcap_{i=1}^{n} A_n$) is a set that comprises of all the elements that are in *all* of the sets $A_1, A_2, \ldots, A_n$.

Definition 1.8: Infinite Intersection of Sets

Intersection of infinitely many sets $A_1, A_2, \ldots$ (denoted by $\bigcap_{i=1}^{\infty} A_n$) is a set that comprises of all the elements that are in all of the sets $A_1, A_2, \ldots$.

For example, $\{1,2,3\} \cap \{2,3,4\} \cap \{3,4,5\} = \{3\}$ and

$$\{1,2\} \cap \{1,3\} \cap \{1,4\} \cap \cdots = \bigcap_{i=1}^{\infty} \{1,i\} = \{1\}.$$

1.1.4 Power Set, Set Difference and Set Complement

Definition 1.9: Power Set

Power Set of a set A (denoted by $\mathcal{P}(A)$) is the set of all subsets of A.

$\mathcal{P}(A)$ is a set of sets, which means that its elements are themselves sets. For a set A, $\emptyset$ and A are always subsets of A, therefore the power set of a set A always has at least two elements (unless, of course, A itself is an empty set, in which case its power set has exactly one element). Let, for example, $A = \{2,5,9\}$. Then

$$\mathcal{P}(A) = \{\emptyset, \{2\}, \{5\}, \{9\}, \{2,5\}, \{2,9\}, \{5,9\}, \{2,5,9\}\}.$$

Definition 1.10: Set Difference

The difference between the sets A and B (denoted by $A \setminus B$) is the set that consists of all elements that are in the set A, but not in the set B.

Thus, if $A = \{1,2,3\}$ and $B = \{2,3,4\}$, then $A \setminus B = \{1\}$. Note that $B \setminus A = \{4\}$. Therefore, the difference between the two sets is not a commutative operation, as it is not always true that $A \setminus B = B \setminus A$. In fact, if $A \setminus B = B \setminus A$, then it has to be $A = B$ and $A \setminus B$ (as well as $B \setminus A$) is $\emptyset$.

Definition 1.11: Set Complement

The complement of a set A (within some larger set B), denoted by A^{C}, is the set $B \setminus A$.

"Larger" in the above definition refers to the subset relation. That is, we mean that B is larger than A if $A \subset B$. If $A = \{1,2\}$ and $B = \{1,2,3,4\}$, then $A^{\mathsf{C}} = \{3,4\}$.

1.2 Tuples and Sequences

A *tuple* is a finite collection of objects in some order. Compared to the sets, the order of elements in the tuple is important and the elements in the tuple can be repeated. That is, two tuples that contain the same elements in different order are different tuples. To differentiate between tuples and sets, a tuple is usually denoted by a parentheses around the list of elements. For example, $A = (1,5,9,2,14)$. As we said, this is a different tuple than $B = (5,1,9,2,14)$, even though they have the same elements.

The *length* of a tuple is the number of positions in that tuple. Thus, the length of the sequence A above is 5. Defining the length of a tuple as the number of elements in it is incorrect, since, for example, in the tuple $C = (1,1,1)$ there is only one element, 1, but there are three positions. A sequence of the length k is called a $k-tuple$. Thus, the sequence A is a 5-tuple. 2-tuples are usually called *pairs*.

Definition 1.12: Tuple Equality

Tuples A and B are equal if they are of the same length and have the same elements in the same positions.

A *sequence* is similar to tuple, except that it can also be infinite. Every tuple is also a sequence, but an infinite sequence is not a tuple. In this text, we will typically use the term sequence, even if we know that the number of positions in it is finite, and is therefore a tuple.

1.2.1 Cartesian Product of Sets

Now that we have introduced a notion of a tuple, we can introduce a very important concept in the set theory - a concept of *Cartesian product* of two sets.

Definition 1.13: Cartesian Product of Sets

Cartesian product of sets A and B is a set of all pairs, where the first element of the pair comes from the set A and the second element of the pair comes from the set B.

Let $A = \{1, 2\}$ and $B = \{x, y\}$, then $A \times B = \{(1, x), (1, y), (2, x), (2, y)\}$. A Cartesian product of a set with itself is also possible. Therefore, $A \times A = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. Note also that the order of the sets matters - $A \times B$ is different from $B \times A$.

It is easy to extend this notion to the product of more than two sets. For example, a Cartesian product of three sets is the set of all 3-tuples, where the first element is from the first set, the second element is from the second set and the third element is from the third set. If $A = \{1, 2\}$, $B = (x, y)$ and $C = (5, 6)$, then

$$A \times B \times C = \{(1, x, 5), (1, x, 6), (1, y, 5), (1, y, 6), (2, x, 5), (2, x, 6), (2, y, 5), (2, y, 6)\}.$$

A^k denotes a Cartesian product of the set with itself k times. Let $A = \{1, 2\}$. Then

$$A^3 = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 1), (2, 1, 2), (2, 2, 1), (2, 2, 2)\}.$$

Chapter 2

Alphabets and Languages

The concept of a language plays a central role in the Theory of Computation. All of the computation models that we are going to introduce in this work will have a single aim of recognising (or accepting) a certain language. This may seem very limiting, as we rarely see computers as devices that merely recognise constructs of some language, but we will see that this isn't the case as much as it may look like. Many realistic problems in computer science can be represented as problems of recognising a certain language.

If we specifically look at programming languages, we can observe many parallels between them and natural languages (which is, indeed, why they are called 'languages'). In natural languages, we typically deal with three different entities - letters, words and sentences. The set of letters that we deal with is typically relatively small (26 in the English language). Sequences of letters make up words. Not all sequences of letters are words (in fact, most of them are not). Sequences of words, together with punctuation symbols, make up sentences. Again, not all sequences of words form a gramatically-valid sentences. The same is the case in programming languages. There we also have letters (or symbols), which is typically a set of ASCII or UNICODE symbols. Some sequences of letters form meaningful units in the language (such as keywords `int` or `return`, identifiers such as `a` or `x1` or string literals such as `"Maths is cool"`), but others do not. Some sequences of these meaningful units form valid statements (such as, for example, `int x = 4;` in C++), but others do not. For example, `int 4 x = ;` is not a valid statement.

When we write a code in some programming language, the first thing that the computer needs to ensure is that the code indeed represents the valid program in that programming language. This is a language recognition problem. The computer needs to recognise whether the code belongs to the language of all valid programs. In any sensible system, this is done mechanically, without any need for human intervention. This is the case because there are strict rules to determine whether some piece of code represents a valid program or not, and these rules are precise enough so that a machine can follow them. In the Theory of Computation, we will deal with much more general languages, but our goal is still to represent languages in a way that the decision of whether something belongs to the language or not is based on explicitly states rules, and not guesswork.

The *Theory of Formal Languages* is the field of mathematics that studies *syntactical* aspects of languages. Rules for languages (in terms of what constructs belong to them) are explicitly stated. We are also interested in the *form* of the constructs that belong to a language, not their *meaning*. That is, we are purely interested in whether some construct belongs to a language or not. Using our programming languages example, we would only be concerned whether some piece of code represents a valid program, not what the result of the execution of that code is.

2.1 Symbol, Alphabet, String and Language

Definition 2.1: Alphabet

Alphabet is a *finite* set of units (*symbols*) out of which structures are built.

Elements of alphabet are sometimes called *letters* or *characters*. We are almost always going to denote the alphabet of interest with Σ.

Definition 2.2: String

String (or *word*) over some alphabet is a finite sequence of symbols from that alphabet.

Rather than writing strings using the sequence notation, we will write them using a juxtaposition. We will write, for example, the string (d, o, o, r) as door. But it is important to remember that strings are, essentially, sequences, therefore all the usual rules for sequences apply for strings. For example, the length of a string is the number of positions in it. The length of a string w is denoted by $|w|$. Also, two strings are equal if they have the same length, and have the same symbols in the same position.

Remark 2.1: Denoting Symbols and String

Although we will always be clear whether something denotes a symbol or a string, we are typically going to use the letters towards the beginning of an alphabet (such as $a, b, c, \dots$) for symbols, and the letters towards the end of alphabet (such as v, w, x, y, z) for strings. Therefore, we will typically talk about "symbol a from alphabet Σ" and "string w over the alphabet Σ".

Empty string is a string that has no symbols. Empty string is a string over any alphabet. We denote the empy string by ϵ. The length of the empty string is 0 ($|\epsilon| = 0$). To avoid any possible confusion, the letter ϵ will not be a part of any alphabet we deal with (otherwise there would be a confusion whether ϵ represents a string of length 0 or a symbol from that alphabet).

Definition 2.3: Language

Language over some alphabet is some set of strings over that alphabet.

Since the empty set is also a set, $\emptyset$ is one example of a language (albeit not terribly interesting one) over any alphabet. Additionally, since ϵ is a string over any alphabet, $\{\epsilon\}$ is another example of a language over any alphabet. Note that $\emptyset$ and $\{\epsilon\}$ are two different languages. The former has zero elements, whereas the latter has one.

Let us take as an example the English language. We can see an English language as a set of valid English words. In this case, the alphabet would be the set of all English letters, plus characters - and '.

$$\Sigma = \{a, b, c, d, e, \dots, -, '\}.$$

The language L of all English words would then be a set of all the words that comprise letters from Σ for which there is an entry in a standard dictionary:

$$L = \{\text{all the main entries in a standard dictionary}\}.$$

Note that this language is large, but finite. We we could, theoretically, list all the strings that belong to the language.

Consider now a language of all valid sentences of the English language. The alphabet of the sentences of the English language is a set of all words in the English language, plus a blank space, plus the usual punctuation symbols. Therefore, the language from the previous example now serves as an alphabet of our new language. This is possible, as both alphabet and language are sets. The language of the sentences in the English language is then a set of all strings of the above alphabet that are valid sentences in English. Note that this language is infinite - we cannot list all the valid sentences of English language. Take, for example

- I bought one apple
- I bought two apples
- I bought three apples
- ...

The question then can be asked how we can describe the language that is infinite? The answer is - in pretty much the same way that we describe infinite sets, which is by stating the properties that every string from the language satisfies. In the case of the sentences of the English language, this could involve listing all the grammar rules for deriving sentences.

We said that we are interested purely in syntatic considerations, i.e. the question of whether a given string belongs to a given language. So what about a sentence, for example, "I bought three Fridays"?. This sentence is gramatically (syntatically) correct, therefore it belongs to our language. Its meaning (semantics) does not make any sense, but that is not our concern. For our purposes, this is as good a sentence as “I have bough a new laptop computer” - they both belong to the language we are interested in.

2.1.1 String and Language Concatenation

Definition 2.4: String Concatenation (Informal)

Given two strings, their *concatenation* is a new string obtained when one of them is appended to the other. Concatenation of strings v and w is denoted by vw.

This is quite an informal definition, as we didn't precisely define what does it mean to 'append' one string to another. We could give a more precise definition in terms of sequences, but this is unnecessarily complicating things when the intuitive definition is clear enough. Therefore, we will not do this.

Suppose $\Sigma = \{0\}$ is an alphabet and $v = 00$ and $w = 000$ are two strings over that alphabet. Then $vw = 00000$. In this case, $wv = 00000$ also. Suppose, on the other hand, that $\Sigma = \{0, 1\}$ is an alphabet and $v = 00$ and $w = 111$ are two strings over that alphabet. Then $vw = 00111$, but $wv = 11100$. Therefore, the order of strings in the concatenation matters.

We will denote by w^2 a concatenation of the string w with itself. In general, w^k will be w appended to itself $k - 1$ times. w^0 will always be ϵ, for every string w.

Definition 2.5: Substring of a String

String v is a substring of a string w if v is contained in w. More formally, string v is a substring of a string w if there exist strings x and y such that $w = xvy$.

The definition of a substring of a string is intuitive enough - string v is a subset of string w if it appear somewhere in that string or, formally, if w can be written as xvy, for some (possibly empty) strings x and y. The fact that both the part before the appearance of v (x) and the part after it (y) in w can be empty implies that every string is a substring of itself. Also, the empty string ϵ is a substring of every string.

Definition 2.6: Language Concatenation

Given two languages L_1 and L_2, their concatenation is a language comprised of all strings that are obtained as a concatenation of some string of L_1 and some string of L_2.
More formallym, given two languages L_1 and L_2, their concatenation is the language L_1L_2 defined as

$$L_1L_2 = \{vw | v \in L_1, w \in L_2\}.$$

As in the case of the strings, we will denote by L^2 the concatenation of the language ando itself, and by L^k the concatenation of the language and itself k times. For example, let $L_1 = \{00, 1\}$ and $L_2 = \{2, 20\}$. Then

$$L_1L_2 = \{002, 0020, 12, 120\}.$$

On the other hand, $L_1^2 = \{0000, 001, 100, 11\}$. L_1^3 would be the concatenation of L_1^2 and L.

Language concatenation is, in some respect, similar to the Cartesian product operator in sets, as in both we take each element of one set and each element of the other set and apply an operation to them. However, if the set A has m elements, and set B has n elements, then the Cartesian product $A \times B$ always has exactly $m \cdot n$ elements. On the other hand, if we have two languages L_1 and L_2, their concatenation might have less than $|L_1| \cdot |L_1|$ elements. This is because some strings might be repeated. For example, if $L = \{0, 00\}$, then $L^2 = LL = \{00, 000, 0000\}$, because 000 is repeated (and we know that there are no repetitions in sets).

2.1.2 Alphabet and Language Closure

Definition 2.7: Closure of an Alphabet

Let Σ be an alphabet. The *closure of the alphabet* Σ, denoted by Σ^* is the set of all strings (of any length, including 0) over the alphabet Σ

Operator $*$ is sometimes called *Kleene star*. Σ^* is always infinite, for any (nonempty) alphabet Σ. Also, Σ^* always includes the empty word. As an example, let $\Sigma = \{0\}$. Then $\Sigma^* = \{\epsilon, 0, 00, 000, 0000, 00000, \dots\}$. Or we can write $\Sigma^* = \{0^n | n = 0, 1, 2, 3, \dots\}$. As another example, let $\Sigma = \{0, 1\}$. Then $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$. Finally, if $\Sigma = \{0, 1, 2\}$, then

$$\Sigma^* = \{\epsilon, 0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, \dots\}.$$

Definition 2.8: Closure of a Language

Let L be a language. The *closure of the language* L, denoted by L^* is the set of all strings that are concatenations of any number of words (including 0) from L.

The $*$ operator on languages is also sometimes called *Kleene star*. For example, if $L = \{0, 11\}$, then

$$L^* = \{\epsilon, 0, 11, 00, 011, 110, 1111, 0011, 0110, 01111, 1100, 11011, 111111, \dots\}.$$

Note that the language and alphabet closure are two different concepts, even though they look (and, indeed, are) very similar. Closure of some alphabet Σ, Σ^*, is simply a formal way of saying "all strings over the alphabet Σ". L^* is a language of all strings that can be formed from the strings from L. L^* is almost always infinite, but there exist two cases of languages L for which L^* is not infinite. It is left as an exercise to the reader to discover them.