St. Francis Xavier University
Department of Computer Science

CSCI 541: Theory of Computing
Lecture 0: Introduction and Mathematical Preliminaries
Fall 2021

1 Introduction

When you think about the theory of computation, what comes to mind?

In fact, what *is* the "theory of computation"? Computers are real machines; how can we reason abstractly about something that we use every day?

Imagine that we remove all of the physical constraints of a computer: finite memory, finite storage, finite compute time. In doing so, we would end up with an ideal computer that can solve any problem we throw at it... or so it would seem.

In this course, we will build up from the simplest possible abstraction of a "computer" to the aforementioned ideal computer, and in doing so we will learn about the fundamental limits of computation itself. What are computers truly capable of? What makes some problems harder to solve than others? What sort of problems can computers solve at all? What sort of problems cannot ever be fully answered by a computer, no matter how much time or how many resources we devote to them? (And no, we're not talking about problems like "what is the answer to the ultimate question of life, the universe, and everything?" That problem is easily solved by Deep Thought.)

Before we embark on this journey, I hope you abandon all preconceived notions of theory being a dry or boring topic. While it may take a little time to build up our vocabulary and notation, some of the results we will study in this course are truly deep and enlightening. Theory is a unique area of study in that it touches literally every other area of computer science in some way, and I invite you to find and explore the connections between the material we learn here and the material from another areas of computer science that interest you.

2 Mathematical Preliminaries

It may be the understatement of the term to say that computer science and mathematics are closely related. This is no less true in theoretical computer science, where we essentially frame the notion of computation itself in terms of mathematical language. Fortunately, almost all of the mathematics that we need to know for this course has been introduced to you in previous courses and years. Here, we will briefly review what we need to know, and we will introduce some of the key terminology used in this course.

2.1 Sets and Sequences

The concepts underpinning almost everything we will learn in this course are some of the most elementary in all of mathematics: sets and sequences.

Sets, Properties, and Operations

Let us begin with a very simple definition: that of a *set*.

Definition 1 (Set)**.** A set is a collection of elements.

If an element a belongs to a set S, then we write $a \in S$. Otherwise, we write $a \notin S$. Typically, we do not repeat elements of a set. If a set needs to contain more than one indistinguishable copy of an element, then we refer to that set as a *multiset*.

The *cardinality* or *size* of a set S, denoted $|S|$, is equal to the number of elements of S. If $|S| = n$ for some finite $n \geq 0$, then we say that S is a *finite set*. Otherwise, we say that S is an *infinite set*. The set with size zero is called the *empty set*, and we denote it by $\emptyset$. At the other extreme, the set of all elements we wish to consider is referred to as the *universal set* or just the *universe*, and is occasionally represented by $\mathcal{U}$.

We may define sets either by listing their elements explicitly, or by describing some property or properties of each element. Naturally, if the set is infinite, we prefer to use the latter method.

Example 2. The set of all positive odd integers less than 20 is

$$S_{\text{odd}<20} = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}.$$

The set of all positive odd integers is

$$S_{\text{odd}} = \{n \mid n = 2k + 1, k \geq 0\}.$$

The set of all positive even integers is

$$S_{\text{even}} = \{n \mid n = 2k, k \geq 0\}.$$

There are some sets that we use frequently enough to warrant their own notation. The set of *natural numbers* is $\mathbb{N} = \{0, 1, 2, \dots\}$, and the set of *integers* is $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$. Computer scientists are also particularly interested in the set of *binary digits* $\mathbb{B} = \{\mathtt{0}, \mathtt{1}\}$.

Given two sets S and T, if every element of S is also an element of T, we say that S is a *subset* of T and we write $S \subseteq T$. If, additionally, T contains at least one element that S does not contain, then we say that S is a *proper subset* of T and we write $S \subset T$. If no element of S is an element of T and vice versa, then we say that S and T are *disjoint*.

Example 3. The set S_{odd} is a proper subset of $\mathbb{N}$. The set $\mathbb{N}$ is a proper subset of $\mathbb{Z}$. The sets S_{odd} and S_{even} are disjoint.

Note that, for all sets S, $\emptyset \subseteq S \subseteq \mathcal{U}$. We can also define set equality in terms of subsets: two sets S and T are equal if both $S \subseteq T$ and $T \subseteq S$.

The *power set* of a set S, denoted by $\mathcal{P}(S)$, is the set of all subsets of S. If S is a finite set, then $|\mathcal{P}(S)| = 2^{|S|}$.

Example 4. Let $S = \{1, 2, 3\}$. Then $\mathcal{P}(S) = \{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$, and $|\mathcal{P}(S)| = 2^3 = 8$.

We can define operations on sets. The *union* of two sets S and T, denoted $S \cup T$, is the set containing all elements that are in S or in T (or in both). The *intersection* of S and T, denoted $S \cap T$, is the set containing all elements that are in both S and T. The *complement* of a set S, denoted S^{C}, is the set containing all elements from our universe that are not in S. Lastly, the *difference* of S and T, denoted $S \setminus T$, is the set of all elements of S that are not also in T.

Example 5. Recall the sets S_{odd} and S_{even} from our earlier examples. We have that $S_{\text{odd}} \cup S_{\text{even}} = \mathbb{N}$, while $S_{\text{odd}} \cap S_{\text{even}} = \emptyset$. Additionally, $S_{\text{odd}}^{\mathsf{C}} = S_{\text{even}}$ and vice versa. Finally, $\mathbb{N} \setminus S_{\text{odd}} = S_{\text{even}}$.

Note that we can define our complement operation in terms of our difference operation by taking $S^{\mathsf{C}} = \mathcal{U} \setminus S$. From this, we also see that $\mathcal{U}^{\mathsf{C}} = \emptyset$ and $\emptyset^{\mathsf{C}} = \mathcal{U}$.

Sequences and Operations

In a set, order does not matter. Thus, for example, the sets $\{1, 2, 4, 8\}$ and $\{2, 4, 8, 1\}$ are equivalent. If we need to preserve some order in a collection of elements, we must instead use a *sequence*.

Definition 6 (Sequence)**.** A sequence is an ordered set of elements.

We distinguish notationally between sets and sequences in the following way: sets are surrounded by {braces}, while sequences are surrounded by (parentheses). We occasionally refer to a finite sequence as a *tuple*, or as a *k-tuple* if we know the sequence contains k elements. If we have a sequence of two elements, we may specifically refer to it as an *ordered pair*. Additionally, as opposed to sets, we may have repeated elements in a sequence.

Example 7. The well-known Fibonacci sequence F_n is defined as follows: we start with $F_0 = F_1 = 1$, and for each $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$. This gives us

$$F_n = (1, 1, 2, 3, 5, 8, 13, 21, 34, \dots).$$

Example 8. One example of a less well-known sequence is the Thue-Morse sequence T_n, which is defined as follows: $T_n = 0$ if the number of ones in the binary representation of n is even, and $T_n = 1$ if the number of ones is odd. This gives us

$$T_n = (0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, \dots).$$

Sequences of integers are an extremely well-studied topic; the On-line Encyclopedia of Integer Sequences (`https://oeis.org`) contains over 340 000 examples of sequences with citations to related papers.

With this notion of sequences, we can define another operation: the *Cartesian product*. Given two sets S and T, their Cartesian product, denoted $S \times T$, is the set of all ordered pairs where the first element of the pair comes from S and the second element comes from T. Formally speaking, $S \times T = \{(a, b) \mid a \in S \text{ and } b \in T\}$.

Example 9. Let $S = \{1, 2, 3\}$ and $T = \{2, 4, 6\}$. Then

$$S \times T = \{(1, 2), (1, 4), (1, 6), (2, 2), (2, 4), (2, 6), (3, 2), (3, 4), (3, 6)\}.$$

We can take the Cartesian product of a set S with itself, and we denote this by $S \times S = S^2$. We can also write $S^2 \times S = S \times S^2 = S^3$, and so on. More generally, taking the Cartesian product of a set S with itself k times is written S^k.

Example 10. The set $\mathbb{Z} \times \mathbb{Z} = \mathbb{Z}^2$ yields the set of all ordered pairs of integers, which is used in the Cartesian coordinate system or the "xy-plane".

The Terminology of Theory

In theoretical computer science, we will often use special names when referring to some of the concepts in this section. Here, we will introduce some of these special names.

A set of symbols, like $\{\texttt{a}, \texttt{b}\}$ or $\{\texttt{0}, \texttt{1}\}$, is called an *alphabet*. Note that the symbols in each of the previous alphabets were written in a typewriter font; this will be done to highlight the fact that we are considering an alphabet instead of a plain set. Often, we represent an alphabet by the Greek letter Σ. If an alphabet Σ contains only one symbol, then we say Σ is a *unary* alphabet.

If we put symbols from an alphabet together, we can create *words* or *strings*. For example, consider the English lowercase alphabet $\{\texttt{a}, \texttt{b}, \dots, \texttt{z}\}$. Some words we can create using this alphabet are `cat`, `computer`, and `pneumonoultramicroscopicsilicovolcanoconiosis`. Note that, like alphabets, we are writing words in the typewriter font for reasons of distinguishability. We often use lowercase variables like w, x, y, or z to denote words.

A word is really a sequence in disguise, since re-ordering distinct symbols of a word would produce a different word. We strip all of the sequence-y notation away in order to make words easier to read. Thus, for our purposes, the sequence $(\texttt{c}, \texttt{a}, \texttt{t})$ is exactly the same as the word `cat`.

Furthermore, we can consider sets of words, and we call such sets *languages*. Much like with plain sets, we can either list the words in a language explicitly, or we can describe a language in terms of some property or properties of each word therein. For example, over the English lowercase alphabet, the language of words with three consecutive double letters is $\{\texttt{bookkeep}, \texttt{bookkeeper}, \texttt{bookkeepers}, \texttt{bookkeeping}\}$. Much like sets, languages can be either finite or infinite. We often use uppercase variables like L to denote languages.

Lastly, we have some special terminology for operations applied to words and languages. The number of symbols in a word w, denoted by $|w|$, is the word's *length*. If the length of a word is zero, then we refer to it as the *empty word* and denote it by ϵ. We can use a notation similar to that used for length to denote the number of occurrences of a certain symbol in a word; for example, if we want to count the number of `a`s in the word $w = \texttt{francisxavier}$, we would write $|w|_{\texttt{a}} = 2$. Given an alphabet Σ, the set of all words over Σ of length k is denoted Σ^k; note that this is exactly the same as taking the repeated Cartesian product of Σ with itself k times, if we strip all of the sequence-y notation away. The set of all words over Σ with length zero or greater is denoted by $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$, and the set of all words over Σ with nonzero length is denoted $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$.

2.2 Relations and Functions

Relations and functions associate elements from one set to elements of another set. The first set ("from") is called the *domain*, and the second set ("to") is called the *range* or the *codomain*.

Relations

A (binary) *relation* between two sets S and T is simply a set consisting of ordered pairs from the Cartesian product $S \times T$.

Definition 11 (Binary relation)**.** A binary relation R from a set S to a set T is a subset of $S \times T$.

We generalize from binary relations to k-ary relations in the usual way.

Example 12. Let $S = \{1, 2, 3\}$, and take $S \times S = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)\}$. The relation of "less than or equal to", denoted $R_{\leq}$, consists of all ordered pairs (a, b) where $a \leq b$; that is,

$$R_{\leq} = \{(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}.$$

Example 13. Let P be the set of professors in the Department of Computer Science, let C be the set of computer science courses, and let $T = \{\text{fall}, \text{winter}\}$. Define a relation $R_{\text{sched}} \subseteq P \times C \times T$ where a tuple $(p, c, t) \in R$ if Professor p is teaching course c in term t. Then, for example,

$$\{(\text{Smith}, 541, \text{fall}), (\text{Smith}, 554, \text{winter})\} \subseteq R_{\text{sched}}.$$

Finding the complete set of tuples in R_{sched} is left as an exercise for the Registrar's Office.

We can define a number of properties of a relation R on a set S depending on which ordered pairs belong to the relation. Let a_1, a_2, and a_3 each be elements of S. Then

- R is *reflexive* if, for all $a \in S$, $(a, a) \in R$;
- R is *symmetric* if, whenever $(a_1, a_2) \in R$, $(a_2, a_1) \in R$;
- R is *antisymmetric* if, whenever $(a_1, a_2) \in R$ and $(a_2, a_1) \in R$, $a_1 = a_2$; and
- R is *transitive* if, whenever $(a_1, a_2) \in R$ and $(a_2, a_3) \in R$, $(a_1, a_3) \in R$.

Note that, despite their names, the properties of symmetry and antisymmetry are not mutually exclusive. A relation may be both symmetric and antisymmetric.

Equivalence Relations

If certain properties hold of a relation, then we give it a special name.

Definition 14 (Equivalence relation)**.** A relation R is an equivalence relation if R is reflexive, symmetric, and transitive.

The name "equivalence relation" comes from the fact that, if R is reflexive, symmetric, and transitive, then any two elements a_1 and a_2 are equivalent with respect to R if $(a_1, a_2) \in R$.

Equivalence relations partition sets into *equivalence classes*, or subsets where all elements in the subset are equivalent with respect to the given relation.

Example 15. Let $S = \{0, 1, \ldots, 10, 11\}$. The relation of "congruent modulo 3", denoted R_{mod3}, consists of all ordered pairs (a, b) where $a \equiv b \pmod 3$. We show that R_{mod3} is an equivalence relation.

- R_{mod3} is reflexive, since $a - a = 0$ for all $a \in S$ and $0 = 3(0)$, so $a \equiv a \pmod 3$.
- R_{mod3} is symmetric, since if $(a, b) \in R_{\text{mod3}}$, then $a \equiv b \pmod 3$ and therefore $a - b = 3k$ for some integer k. At the same time, $b - a = 3(-k)$, so $b \equiv a \pmod 3$ and, therefore, $(b, a) \in R_{\text{mod3}}$.
- R_{mod3} is transitive, since if we have $(a, b), (b, c) \in R_{\text{mod3}}$, then $a \equiv b \pmod 3$ and $b \equiv c \pmod 3$, which means $a - b = 3k$ and $b - c = 3\ell$ for some integers k and ℓ. However, we then have $a - c = (a - b) + (b - c) = 3k + 3l = 3(k + l)$, and so $a \equiv c \pmod 3$. Thus, $(a, c) \in R_{\text{mod3}}$.

Moreover, the equivalence relation R_{mod3} partitions the set S into three equivalence classes: $\{0, 3, 6, 9\}$ (remainder 0), $\{1, 4, 7, 10\}$ (remainder 1), and $\{2, 5, 8, 11\}$ (remainder 2).

Functions

A *function* from a set S to a set T is a special kind of relation where each element of S is mapped to exactly one element of T. All functions are relations, but not all relations are functions.

Definition 16 (Function)**.** A function f from a set S to a set T is a subset of $S \times T$ such that, for each $a \in S$, there exists exactly one $b \in T$ such that $(a, b) \in f$.

If f is a function from a set S to a set T, then we often denote this by the shorthand notation $f \colon S \to T$. If $(a, b) \in f$, then we write $f(a) = b$.

Example 17. The exponential function, $f(x) = 2^x$, is a classic example of a function $f \colon \mathbb{N} \to \mathbb{N}$ where we have $f(0) = 1$, $f(1) = 2$, $f(2) = 4$, $f(3) = 8$, and so on.

Similar to relations, we can define a number of properties of a function f from a set S to a set T.

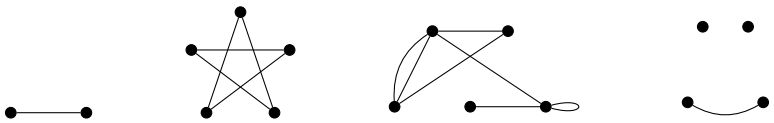
- f is *injective* (or *one-to-one*) if, for all $a_1, a_2 \in S$ where $a_1 \neq a_2$, $f(a_1) \neq f(a_2)$;
- f is *surjective* (or *onto*) if, for all $b \in T$, there exists $a \in S$ such that $f(a) = b$; and
- f is *bijective* if it is both injective and surjective.

2.3 Graphs

A graph is a rather straightforward mathematical concept: it is a structure that consists of *vertices* (or *nodes*) connected by *edges*. Indeed, we often define a graph directly in terms of its vertex set and edge set.

Definition 18. A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where each element of E is a pair $\{u, v\}$ of vertices $u, v \in V$.

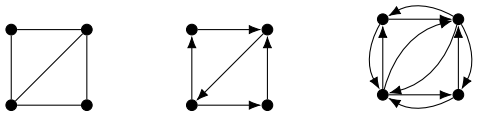
Example 19. Each of the following are graphs:



Definition 18 is worded in a very general way, and it allows for things such as multiple edges between the same vertices or edges joining a vertex to itself (called *loops*).

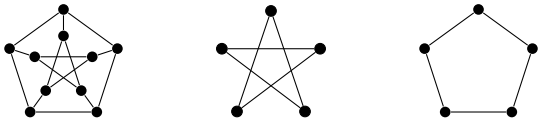
Note that Definition 18 also assumes that the set of edges E consists of unordered pairs. By this definition, if an edge exists between vertices u and v, then another edge implicitly exists between v and u. If we instead require that E consists of ordered pairs, then the existence of an edge between vertices u and v does not necessarily imply the existence of an edge between v and u. We say that such graphs are *directed*, because the direction of each edge in the set E matters when we move between vertices.

Example 20. Out of the following graphs, the first graph is undirected and both the second and third graphs are directed. The third graph is the directed equivalent of the first graph.



If we have two graphs $G = (V, E)$ and $H = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$, then we say that H is a *subgraph* of G. In other words, a subgraph is a copy of G with vertices or edges (or both) removed. Note that if we remove a vertex, then we cannot leave behind any edges that touched the removed vertex.

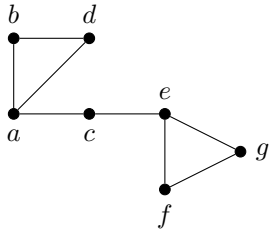
Example 21. The leftmost graph contains the two rightmost graphs as subgraphs.



Given a graph $G = (V, E)$, two vertices $u, v \in V$, and a natural number n, a *path* of length n from vertex u to vertex v is a sequence of edges $\{e_1, \ldots, e_n\}$ where $e_1, \ldots, e_n \in E$ and where $e_1 = \{u, w_1\}, \ldots, e_n = \{w_{n-1}, v\}$ where $w_1, \ldots, w_{n-1} \in V$. Moreover, a path of length n from a vertex u to a vertex v is called a *circuit* (or *cycle*) if $u = v$ and $n > 0$. If no edge in the path or circuit is included more than once, then we say that the path or circuit is *simple*.

Lastly, we say that a graph $G = (V, E)$ is *connected* if, for every pair of vertices $u, v \in V$, there exists a path in G between u and v.

Example 22. In the following graph, we see that (among many others) there exist paths a–b, a–c–e–g, c–e–c–a, and f–g–f; simple paths b–a–c–e, d–a–b, and f–e–c; and circuits a–b–d–a and e–f–g–e.



As one final remark, in this course, we will draw graphs a little differently than how a graph theorist or a mathematician might draw them. Many of the graphs we will draw will have labelled vertices—similar to the vertices in the previous example—as well as labelled edges. Indeed, our graphs will more closely resemble the following figure: