Discrete Mathematics in a Nutshell or Theoretical Foundations of Computer Science

Yuliya Lierler
University of Nebraska at Omaha

Dedicated to
my teacher Vladimir Lifschitz

Preface

This textbook has been largely inspired by the style of teaching practiced by Vladimir Lifschitz during my years at the University of Texas at Austin. It was Spring 1999 when I took CS 336 titled "Analysis of Programs" and devoted to proofs of program correctness. Vladimir Lifschitz class surveyed mathematical techniques useful in the analysis and verification of programs. In his teaching, Dr. Lifschitz relied on a set of concise lecture notes consisting of mathematical definitions and problem statements. The students were tasked to study the definitions, and use these to tackle the given problems at home. Then, a lecture hall would turn into an interactive classroom, where students would present their solutions on a blackboard. This style of teaching mathematics is attributed to Robert Lee Moore, a topologist who practiced this art at the University of Pennsylvania when he began teaching in 1911.

This textbook is meant to assist the instructors in introducing elements of Moore's style of teaching into their classroom. It can also be used in a more traditional settings, where the content of the textbook serves the basis for classical lectures while the students can be tasked from time to time with the exercises sprinkled across the text. As a textbook it not only lists the definitions but also illustrates these on examples and provides solutions to some of the stated problems. This text attempts to present material as concisely as possible. In addition, each section contains exercises. These are meant to be solved by students. The content of the book should be all that the students need to tackle these problems. As such the text can be used for the self study of the material. The conciseness of the text should not allude that the material presented is of great simplicity, rather the students are invited to experience the beauty and complexity of the material by putting their thoughts and effort into solving exercises.

In my lectures, I follow the material of the textbook almost verbatim. Often, I invite students during the class time or home to tackle the relevant exercises individually for a while, and then group the students together to check on their joint understanding of a problem and solutions they propose. At last a solution is requested to be shared with the whole class on the blackboard.

This textbook is the result of conversion of lecture notes for the course titled "Theoretical Foundations of Computer Science" at the University of Nebraska Omaha, where I have been teaching this course for over a decade. Some of the material was inspired by the lecture notes on mathematical logic by Vladimir Lifschitz posted at `http://www.cs.utexas.edu/~vl/teaching/388Lnotes.pdf` as well as his lecture notes for University of Texas at Austin CS 311 by the same author.

Conventions Across the textbook, the *italics font* is utilized primarily to mark the concepts that are being defined. Hence if you are looking for a definition of a concept look for this term being written in italics.

Contents

1 Basic Structures: Sets, Functions, Sequences, Sums

1.1 Sets Basics

We will study "naive set theory", which is defined in natural language (unlike axiomatic set theories defined using formal logic). A *set* is an unordered collection of "objects". The *objects* in a set are called the *elements*, or *members*, of the set. A set is said to *contain* its elements.

The *membership* question is the primary operation on a set. That is, given a set A and an element x, we would like to know if x is a member of A. The set membership operator is the symbol $\in$ and we write $x \in A$, when x is a member of A; and $x \notin A$, when x is not a member of A.

Specifying Sets A set can be specified in several ways:

- exhaustive enumeration: $\{3, 5, 7, 9, 11, 13\}$. Note how curly brackets are used to mark the fact that we are speaking about a set. In this example, the set containing exactly six elements is considered, namely, 3, 5, 7, 9, 11, and 13 (which happened to be all odd numbers between 2 and 14).
- ellipsis ...: $\{3, 5, 7, \ldots, 14\}$. Note that ... hide the details of how elements of a set are "generated". Ellipsis can be used in good faith when the function for generating the next element in a set is simple (understood by the audience). For instance, in this case we specified the same set as in the previous bullet.
- Set builder notation: $\{x \mid x \text{ is odd and } 2 < x < 14\}$. To the right of the |-symbol we list a condition so that whenever an element satisfies this condition it is considered to be a member of the specified set. It is easy to see that the set specified in this bullet coincides with the set specified above.

Some common sets and their "names":

$\mathbb{N} \stackrel{def}{=} \{0, 1, 2, \ldots\}$	the set of natural numbers
$Z \stackrel{def}{=} \{\ldots, -2, -1, 0, 1, 2, \ldots\}$	the set of integers
$Z^+ \stackrel{def}{=} \{1, 2, \ldots\}$	the set of positive integers
$Q \stackrel{def}{=} \{p/q \mid p \in Z \text{ and } q \in Z \text{ and } q \neq 0\}$	the set of rational number
R	the set of real numbers

By symbol $\emptyset$ we denote the set that contains no elements. We say that a set is a *singleton* if it consists of exactly one element. For instance, set $\{\emptyset\}$ is a singleton set, whose only member is an empty set.

Relational Operators Two sets are *equal* if and only if they have the same elements; or, in other words, sets A and B are equal, denoted $A = B$, when for any object x, the following holds $x \in A$ if and only if $x \in B$. For example, sets $\{1, 2, 2, 3\} = \{2, 1, 3\} = \{1, 2, 3\}$.

In the sequel, we often separate elements of the sets by empty spaces in place of commas so that we can write $\{1, 2\}$ and $\{1\ \ 2\}$ to denote the same set consisting of two elements 1 and 2.

Exercise 1. *Which of the following pairs of sets are equal:*

- $\{a\ \ b\ \ c\}$ *and* $\{b\ \ a\ \ c\}$
- $\{a\ \ b\ \ c\}$ *and* $\{b\ \ b\ \ a\ \ c\}$
- $\{a\ \ b\ \ c\}$ *and* $\{a\ \ \{b\}\ \ c\}$
- $\{a\ \ b\ \ c\}$ *and* $\{a\ \ b\ \ c\ \ \emptyset\}$

We say that set A is a *subset* of set B, denoted $A \subseteq B$, when for every object x if $x \in A$ then $x \in B$. We say that set A is a *proper (strict) subset* of set B, denoted $A \subset B$, when $A \subseteq B$ and there exists an element x in B such that it is not an element of A. For instance, $\{1\} \subseteq \{1, 2\} \subseteq \{1, 2\}$, while $\{1\} \subset \{1, 2\} \not\subset \{1, 2\}$.

It is easy to see that for every set A, $\emptyset \subseteq A$ and $A \subseteq A$.

Exercise 2. *Which of the following statements hold:*

- $a \in \{a \;\; b \;\; c\}$
- $\{a \;\; b\} \in \{a \;\; b \;\; c\}$
- $\{a \;\; b\} \in \big\{\{a \;\; b\} \;\; c\big\}$
- $\emptyset \subseteq \big\{\{a \;\; b\} \;\; c\big\}$
- $\{a \;\; b\} \subset \{a \;\; b \;\; c\}$
- $\{a \;\; b\} \subset \{a \;\; b\}$
- $\{a \;\; b\} \subseteq \{a \;\; b\}$

Two Primitive Operations: Cardinality and Powerset For a finite set A, the *cardinality* of A, denoted $|A|$, is the number of (distinct) elements in A. For a set A, we call the set of all its subsets the *powerset of* A, denoted $\mathcal{P}(A)$. For example,

$$|\{1\}| = |\big\{\{a \;\; b\}\big\}| = |\{\emptyset\}| = 1,$$

$$|\emptyset| = 0,$$

$$\mathcal{P}\big(\{1 \;\; 2\}\big) = \big\{\emptyset \;\; \{1\} \;\; \{2\} \;\; \{1,2\}\big\}.$$

Exercise 3. *Compute the cardinalities of the following sets*

- $\{a \;\; b \;\; c\}$
- $\{a \;\; b \;\; 5\}$
- $\big\{a \;\; b \;\; \{\Gamma \;\; \Theta\}\big\}$
- $\Big\{\big\{\{a \;\; b \;\; c\}\big\}\Big\}$
- $\mathcal{P}\big(\{a \;\; b\}\big)$

Cartesian Products The order of elements in a collection is often important. To capture order we introduce a new concept called an *ordered tuple*. The *ordered n-tuple* $(a_1, a_2, \ldots, a_n)$ is the ordered collection that has a_1 as its first element, a_2 as its second element, $\ldots$, and a_n as its n^{th} element.

For ordered tuples $(a_1, \ldots a_n)$ and $(b_1, \ldots b_m)$ we say that they are *equal*, written $(a_1, \ldots a_n) = (b_1, \ldots b_m)$, when

- $n = m$ and
- for every i, $1 \leq i \leq n$, it holds that $a_i = b_i$.

For instance, $(1,2) = (1,2)$, $(1,2) \neq (1,2,3)$, and $(1,2) \neq (2,1)$.

For sets A and B, the *Cartesian product* of A and B, denoted by $A \times B$, is the set of all ordered pairs (a,b), where $a \in A$ and $b \in B$. For instance, let A be $\{1,2\}$ and B be $\{3,4,5\}$ then

$$A \times B = \big\{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\big\}.$$

The Cartesian product of the sets $A_1, A_2, \ldots, A_n$, denoted $A_1 \times A_2 \times \cdots \times A_n$, is the set

$$\{(a_1, a_2, \ldots, a_n) \;\; | \;\; \text{for every } i \text{ so that } \;\; (1 \leq i \leq n) \;, a_i \in A_i\}.$$

For a set A, A^n denotes a Cartesian product $A \times \cdots \times A$ of n elements. For instance,

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

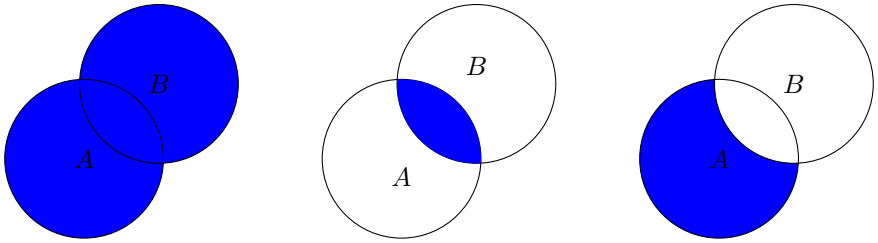


Figure 1: Venn diagrams for $A \cup B$, $A \cap B$, and $A \setminus B$, respectively.

1.2 Sets Operations

For sets A and B, the *union* of the sets A and B, denoted by $A \cup B$, is the set that contains these elements that are either in A or in B, or in both. For sets A and B, the *intersection* of the sets A and B, denoted by $A \cap B$, is the set that contains these elements that are in both A and B. For sets A and B, the *difference* between A and B, denoted by $A \setminus B$, is the set that contains these elements that are in A but not in B.

Exercise 4. *Which of the following pairs of of sets are equal*

- $\{a \;\; b \;\; c\} \cap \{a \;\; b \;\; c\}$ *and* $\{b \;\; a \;\; c\} \cup \{b \;\; a \;\; c\}$
- $\{a \;\; b \;\; c\} \cap \{a \;\; b\}$ *and* $\{b \;\; b \;\; a \;\; c\} \cap \{b \;\; a\}$
- $\{a \;\; b \;\; c\} \cap \{a \;\; b\}$ *and* $\{a \;\; \{b\} \;\; c\} \cap \{a \;\; b\}$
- $\{b \;\; c\} \cup \{a \;\; b\}$ *and* $\{a \;\; c\} \cup \{a \;\; b\}$
- $\{a \;\; b \;\; \emptyset\} \cap \{\emptyset\}$ *and* $\emptyset \cap \{\emptyset\}$

Venn diagrams can be used to graphically illustrate the semantics of set operations. Figure 1 presents Venn diagrams for $A \cup B$, $A \cap B$, and $A \setminus B$, respectively. The results of these operations is marked in blue.

Principle of inclusion-exclusion: The cardinality of the union of two arbitrary sets A and B respects the following condition:

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Set Identities The following identities hold and summarize some interesting properties about set operations, where A, B, and C stand for arbitrary sets:

$$\begin{aligned}
&A \cup \emptyset = A \\
&A \cap \emptyset = \emptyset \\
&A \cup A = A \\
&A \cap A = A \\
&A \cup B = B \cup A \\
&A \cap B = B \cap A \\
&(A \cup B) \cup C = A \cup (B \cup C) \\
&(A \cap B) \cap C = A \cap (B \cap C) \\
&A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \\
&A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \\
&A \cup (A \cap B) = A \\
&A \cap (A \cup B) = A
\end{aligned}$$

You may find Venn diagrams instrumental in analyzing the listed identities.

Exercise 5. *Which of the following statements hold:*

- $(A \cup B) \cap \emptyset = A \cup B$
- $(A \cup B) \cup \emptyset = A \cup B$
- $(B \cap C) \cup A = (A \cup B) \cup (A \cup C)$
- $(A \setminus B) \cup (A \cap B) = A$

Generalized Unions and Intersections Given n sets, $A_1, \ldots, A_n$ we may utilize big-$\bigcup$ and big-$\bigcap$ notation to denote the union and the intersection of all of these sets, respectively:

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$$

and

$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n.$$

As an example, consider that for any natural number i, we define the set N_i as follows $\{0, 1, 2, 3, \ldots, i\}$. As a result, $N_0 = \{0\}$, $N_1 = \{0, 1\}$, $N_2 = \{0, 1, 2\}$,.... Then,

$$\bigcup_{i=0}^{n} N_i = \{0, 1, 2, 3, \ldots n\} = N_n,$$

$$\bigcup_{i=n}^{\infty} N_i = \{0, 1, 2, 3, \ldots\} = \mathbb{N}.$$

Exercise 6. *What is the cardinality of the set* $\bigcap_{i=0}^{\infty} N_i$*?*

Exercise 7. *Is it the case that for any $k < n$, the following equality holds*

$$\bigcup_{i=k}^{n} N_i = \bigcup_{i=k+1}^{n} N_i$$

1.3 Functions

For nonempty sets A and B, a *(total) function* f from A to B, denoted $f : A \to B$, is an assignment/mapping/transformation of exactly one element of B to each element of A. We often drop word total when referring to a total function. Let f denote a function from A to B. We say that A is the *domain* of f and B is the *codomain* of f; also, we say that f *maps* A to B. We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A. If $f(a) = b$, we say that

- b is the *image* of a and
- a is the *preimage* of b.

For instance, let $f : Z \to Z$ denote the function $f(x) = x^2$. In this example, set Z of integers plays a role of both domain and codomain of this function.

To proceed with the presentation we define the concepts of a string and its prefix. A *string* is a sequence of symbols $s_1 \dots s_n$ (it may be an empty sequence in which case we call it an empty string, denoted ϵ). For a string $s_1 \dots s_n$, a string of the form $s_1 \dots s_i$ so that $0 \leq i \leq n$ is called a *prefix*. For instance, all possible prefixes of bit string 001 are ϵ, 0, 00, 001.

The presented definition of a function is general because it does not assume that the domain and the codomain consist of numbers. In the following examples, the domain of each function is the set $\mathbf{S}$ of all bit strings, namely,

$$\mathbf{S} = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}.$$

1. Function l from $\mathbf{S}$ to $\mathbb{N}$: $l(x)$ is the length of x. For instance, $l(00110) = 5$.
2. Function z from $\mathbf{S}$ to $\mathbb{N}$: $z(x)$ is the number of occurrences of zeros in x. For instance, $z(00110) = 3$.
3. Function n from $\mathbf{S}\backslash\{\epsilon\}$ to $\mathbb{N}$: $n(x)$ is the number represented by x in binary notation. For instance, $n(00110) = 6$.
4. Function e from $\mathbf{S}$ to $\mathbf{S}$: $e(x)$ is the string $1x$. For instance, $e(00110) = 100110$.
5. Function r from $\mathbf{S}$ to $\mathbf{S}$: $r(x)$ is the string x reversed. For instance, $r(00110) = 01100$.
6. Function p from $\mathbf{S}$ to $\mathcal{P}(\mathbf{S})$: $p(x)$ is the set of prefixes of x. For instance,

$$p(00110) = \{\epsilon, 0, 00, 001, 0011, 00110\}.$$

The *graph* of a function f is the set of all ordered pairs of the form $\langle x, f(x)\rangle$. For instance, consider the function f from $\mathbb{N}$ to $\mathbb{N}$ defined by the formula $f(n) = 2n + 1$. The graph of this function is the set

$$\{\langle 0, 1\rangle,\ \langle 1, 3\rangle,\ \langle 2, 5\rangle, \langle 3, 7\rangle, \dots\}.$$

It is clear that the graph of any function from A to B is a subset of the Cartesian product $A \times B$.

It is sometimes convenient to talk about a function and its graph as if it were the same thing. For instance, instead of writing $f(n) = 2n + 1$, we can write:

$$f = \{\langle 0, 1\rangle,\ \langle 1, 3\rangle,\ \langle 2, 5\rangle, \langle 3, 7\rangle, \dots\}.$$

This convention allows us to give yet another definition of a function, one that refers to sets of pairs: For any sets A and B, a *function from A to B* is a set $f \subseteq A \times B$ such that for every element x of A there exists a unique element y of B for which $\langle x, y\rangle \in f$; that element is denoted by $f(x)$.

Figure 2 presents graphs of three functions visually. These functions are from the set of real numbers to the set of real numbers defined as follows $f(x) = 2x + 1$ (in red), $f(x) = x^2$ (in blue), and $f(x) = x^3$ (in green). Such graphical representation often helps the analysis of properties of functions.

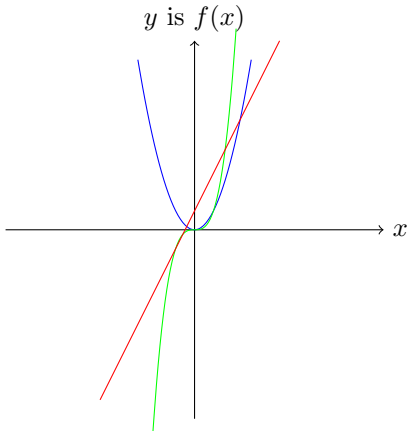


Figure 2: Function graphs: $f(x) = 2x + 1$ (in red), $f(x) = x^2$ (in blue), and $f(x) = x^3$ (in green)

One-to-One, Onto Functions, Composition and Inverse A function f is *one-to-one*, or *injective*, if and only if $f(a) = f(b)$ implies that $a = b$ for all a and b in the domain of f. For instance, function $f : Z \to Z$ where $f(n) = n^3$ is one-to-one; whereas function $f : Z \to Z$ where $f(n) = n^2$ is not one-to-one (indeed, consider the fact that $f(-1) = f(1) = 1$).

Exercise 8. *Let set A_1 be $\{\alpha, \beta\}$ and set B_1 be $\{1, 2\}$. A table below defines two functions $f_1 : A_1 \to B_1$ and $f_2 : A_1 \to B_1$:*

x	$f_1(x)$	$f_2(x)$
α	1	1
β	2	1

Is function f_1 one-to-one? Is function f_2 one-to-one. Explain why?

A function f from A to B is called *onto*, or *surjective*, if and only if for every element $y \in B$ there is an element $x \in A$ with $f(x) = y$. For instance, function $f : R \to R$ where $f(n) = n^3$ is onto; whereas function $f : R \to R$ where $f(n) = n^2$ is not onto.

Exercise 9. *Consider functions f_1 and f_2 defined in Exercise 8. Is function f_1 onto? Is function f_2 onto.* Explain why?

A function f is a *one-to-one correspondence*, or a *bijection*, if it is both one-to-one and onto. Function $f : R \to R$ where $f(n) = n^3$ is a bijection.

Exercise 10. *Consider functions f_1 and f_2 defined in Exercise 8. Is function f_1 a bijection? Is function f_2 a bijection.* Explain why?

If f is a function from A to B, and g is a function from B to C, then the *composition* of these functions is the function h from A to C defined by the formula $h(x) = g(f(x))$. This function is denoted by $g \circ f$.

If f is a bijection from A to B then the *inverse* of f is the function g from B to A such that, for every $x \in A$, $g(f(x)) = x$. This function is denoted by f^{-1}.

Exercise 11. *Consider function f_1 defined in Exercise 8. Compute f_1^-.*

Increasing/decreasing Functions A function $f : A \to B$ where $A \subseteq R$ and $B \subseteq R$ is called

increasing	when	for every x and y in A, if $x < y$ then $f(x) \leq f(y)$
strictly increasing	when	for every x and y in A, if $x < y$ then $f(x) < f(y)$
decreasing	when	for every x and y in A, if $x < y$ then $f(x) \geq f(y)$
strictly decreasing	when	for every x and y in A, if $x < y$ then $f(x) > f(y)$

Notice that if a function is strictly increasing/decreasing then it must be one-to-one. For instance, function $f : Z \to Z$ where $f(n) = n^3$ is strictly increasing, whereas function $f : Z \to Z$ where $f(n) = n^2$ is neither increasing nor decreasing. Function $f(n) = 1$ is an example of a function that is both increasing and decreasing but neither strictly increasing nor strictly decreasing.

Partial Functions A *partial function* f from a set A to a set B is an assignment to each element x in a subset of A, called the domain of definition of f, of a unique element in y in B. We say that f is *undefined* for elements in A that are not in the domain of definition of f. For example, function $f : R \to R$ where $f(n) = 1/n$ is partial. This function is undefined for 0.

Definitions By Cases A function can be defined by cases, when several formulas are used for calculating the value of the function for various arguments, for instance, below several functions are defined from R to R:

$$f(x) = \begin{cases} x^2, & \text{if } x \geq 0, \\ 0, & \text{otherwise;} \end{cases}$$

$$g(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{otherwise;} \end{cases}$$

$$|x| = \begin{cases} x, & \text{if } x \geq 0, \\ -x, & \text{otherwise;} \end{cases}$$

$$\text{sgn}(x) = \begin{cases} -1, & \text{if } x < 0, \\ 0, & \text{if } x = 0, \\ 1, & \text{if } x > 0. \end{cases}$$

1.4 Sequences and Summations

A *sequence* is a function from a subset of the set of integers (e.g., usually the natural numbers or the positive integers) to a set.

Definitions by cases can be used also for defining sequences. For instance, the sequence

n	1	2	3	4	5	6	...
A_n	2	5	2	5	2	5	...

can be defined by the formulas

$$A_n = \begin{cases} 2, & \text{if } n \text{ is odd,} \\ 5, & \text{otherwise.} \end{cases} \tag{1}$$

The sequence A_n can be defined also by a single formula:

$$A_n = \frac{7 + 3(-1)^n}{2}. \tag{2}$$

We now construct a formal argument to illustrate that formula (2) indeed captures the definition (1) of the sequence A_n. Consider two cases. *Case 1:* n is odd. Then

$$A_n = 2 \text{ per definition;} \quad \frac{7 + 3(-1)^n}{2} = \frac{7 + 3 \cdot (-1)}{2} = \frac{4}{2} = 2.$$

Case 2: n is even. Then

$$A_n = 5 \text{ per definition;} \quad \frac{7 + 3(-1)^n}{2} = \frac{7 + 3 \cdot 1}{2} = \frac{10}{2} = 5.$$

So formula (2) holds in both cases. This argument is an illustration of proof by cases.

Summation For a sequence $a_0, \ldots, a_n$ of numbers and a range $l..u$ $(0 \le l \le u \le n)$, a *summation*/$\sum$ is a compact notation for representing the sum of some terms in a sequence so that

$$\sum_{j=l}^{u} a_j = a_l + a_{l+1} + \cdots + a_u.$$

Notational variations are possible, say, $\sum_{l \le j \le u} a_j$. The variable j is called the *index of summation* (it can be renamed if desired), say,

$$\sum_{j=l}^{u} a_j = \sum_{i=l}^{u} a_i = \sum_{k=l}^{u} a_k.$$

In many cases, the lower bound l is 0 or 1. However, this is not required. This notation may have reminded you the big-$\bigcup$ and big-$\bigcap$ notation used earlier.

The index of summation need not be a subscript; it can be used to identify a term that participates in the summation. For example, we can write $\sum_{j=1}^{5} j^2$; this expression evaluates to

$$\sum_{j=1}^{5} j^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55.$$

Exercise 12. *What the value of the following summation* $\sum_{j=1}^{3} 3$.

Exercise 13. *Do the following statements hold:*

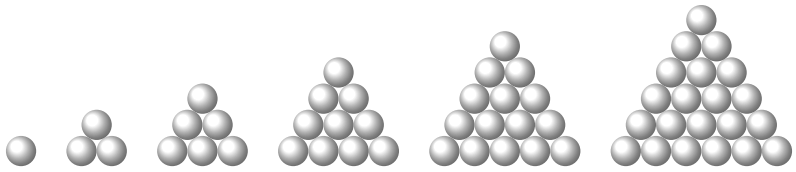


Figure 3: Triangular numbers visually: $T_1 \dots T_6$.

- $\sum_{j=1}^{5} j^2 = \sum_{k=0}^{4} (k+1)^2$
- $\sum_{j=1}^{5} j^2 = 1^2 + \sum_{j=2}^{5} j^2 = (\sum_{j=1}^{4} j^2) + 5^2$

Double Summation $\sum$-notation can be used in complex nested expressions. In such cases the rule is to evaluate the most inner expression first. For example,

$$\sum_{i=1}^{4} \sum_{j=1}^{3} j \cdot i = \sum_{i=1}^{4} (1 \cdot i + 2 \cdot i + 3 \cdot i) = \sum_{i=1}^{4} 6 \cdot i = 6 \cdot 1 + 6 \cdot 2 + 6 \cdot 3 + 6 \cdot 4 = 60$$

Summation Over Sets One can also specify a sum over the elements of a set using the following notation:

$$\sum_{x \in A} f(x)$$

where f is a function from elements in set A to numbers. For instance,

$$\sum_{x \in \{0,2,4\}} x = 0 + 2 + 4.$$

Exercise 14. *What the value of the following summation* $\sum_{x \in \{0,2,4\}} 3$.

Triangular Numbers and Their Relatives In the definitions below, n is a nonnegative integer.

The *triangular number* T_n is the sum of all integers from 1 to n:

$$T_n = \sum_{i=1}^{n} i = 1 + 2 + \dots + n.$$

For instance, $T_4 = 1 + 2 + 3 + 4 = 10$. As we chose n to be nonnegative integer, value 0 is a valid choice for n; T_0 amounts to the summation of no terms and hence evaluates to 0. It turns out that a triangular number is a number that can be represented by a pattern of dots arranged in an equilateral triangle with the same number of dots on each side. Figure 3 presents this visualization for $T_1, T_2, \dots, T6$.

By B_n we denote the number of ways to choose two elements out of n. For instance, if we take 5 elements a, b, c, d, e, then there will be 10 ways to choose two:

$$a, b; \quad a, c; \quad a, d; \quad a, e; \quad b, c; \quad b, d; \quad b, e; \quad c, d; \quad c, e; \quad d, e.$$

We conclude that $B_5 = 10$.

By S_n we denote the sum of the squares of all integers from 1 to n:

$$S_n = \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \dots + n^2.$$

For instance, $S_4 = 1^2 + 2^2 + 3^2 + 4^2 = 30$.

By C_n we denote the sum of the cubes of all integers from 1 to n:

$$C_n = \sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \cdots + n^3.$$

For instance, $C_4 = 1^3 + 2^3 + 3^3 + 4^3 = 100$.

The *harmonic number* H_n is defined by the formula

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}.$$

For instance, $H_3 = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} = \frac{11}{6}$.

The *factorial* of n is the product of all integers from 1 to n:

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \cdots \cdot n$$

($\prod$ stands for product, similarly as $\sum$ stands for sum). For instance, $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$. Note that $0! = 1$.

Triangular numbers can be calculated using the formula

$$T_n = \frac{n(n+1)}{2}.$$

To prove this claim, we start with two expressions for T_n:

$$\begin{array}{rcccccccccccccc} T_n & = & 1 & + & 2 & + & 3 & + & \cdots & + & (n-2) & + & (n-1) & + & n \\ T_n & = & n & + & (n-1) & + & (n-2) & + & \cdots & + & 3 & + & 2 & + & 1. \end{array}$$

If we add them column by column, we'll get:

$$\begin{aligned} 2T_n &= (n+1) + (n+1) + (n+1) + \cdots + (n+1) + (n+1) + (n+1) \\ &= n(n+1), \end{aligned}$$

and it remains to divide both sides by 2.

There are also other ways to prove the formula for T_n. Consider these identities:

$$\begin{array}{rcccccc} (1+1)^2 & = & 1^2 & + & 2 \cdot 1 \cdot 1 & + & 1^2, \\ (2+1)^2 & = & 2^2 & + & 2 \cdot 2 \cdot 1 & + & 1^2, \\ (3+1)^2 & = & 3^2 & + & 2 \cdot 3 \cdot 1 & + & 1^2, \\ & \cdots & & & & & \\ (n+1)^2 & = & n^2 & + & 2 \cdot n \cdot 1 & + & 1^2. \end{array}$$

If we add them column by column, we'll get:

$$2^2 + 3^2 + 4^2 + \cdots + (n+1)^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 + 2T_n + n.$$

Subtract $2^2 + 3^2 + \cdots + n^2$ from both sides:

$$(n+1)^2 = 1^2 + 2T_n + n.$$

Expand the left-hand side and subtract $n + 1$ from both sides:

$$n^2 + n = 2T_n.$$

It remains to divide both sides by 2. Later, we will see yet another way to prove this claim using the proof method called induction.

The relationship between the number B_n and triangular numbers is described by the formula

$$B_n = T_{n-1} \qquad (n \geq 1)$$

It follows that

$$B_n = \frac{(n-1)n}{2}.$$

Exercise 15. *Take* 6 *elements* a, b, c, d, e, f. *How many ways are there to choose two distinct elements of these?*

There is a simple formula for C_n:

$$C_n = (T_n)^2 = \frac{n^2(n+1)^2}{4}.$$

There are no simple precise formulas found for harmonic numbers and factorials.

2 Elements of Mathematical Logic and Proofs

Kenneth H. Rosen textbook on *Discrete Mathematics and its Applications* starts its presentation with the following paragraphs

> The rules of logic specify the meaning of mathematical statements. For instance, these rules help us understand and reason with statements such as *"There exists an integer that is not the sum of two squares"* and *"For every positive integer n, the sum of the positive integers not exceeding n is $n(n+1)/2$"*. Logic is the basis of all mathematical reasoning, and of all automated reasoning. It has practical applications to the design of computing machines, to the specification of systems, to artificial intelligence, to computer programming, to programming languages, and to other areas of computer science, as well as to many other fields of study.
>
> To understand mathematics, *we must understand what makes up a correct mathematical argument, that is, a proof.* Once we prove a mathematical statement is true, we call it a theorem. A collection of theorems on a topic organize what we know about this topic. To learn a mathematical topic, a person needs to actively construct mathematical arguments on this topic, and not just read exposition. Moreover, knowing the proof of a theorem often makes it possible to modify the result to fit new situations.

Logic can be used in programming, and it can be applied to the analysis and automation of reasoning about software and hardware. This is why it is sometimes considered a part of theoretical computer science. Since reasoning plays an important role in intelligent behavior, logic is closely related to artificial intelligence.

The short book by the German philosopher Gottlob Frege (1848–1925) with the long title *Ideography, a Formula Language, Modeled upon that of Arithmetic, for Pure Thought* (1879), introduced notation that is somewhat similar to what is now called first-order formulas. Frege wrote:

> I believe that I can best make the relation of my ideography to ordinary language clear if I compare it to that which the microscope has to the eye. Because of the range of its possible uses and the versatility with which it can adapt to the most diverse circumstances, the eye is far superior to the microscope... But, as soon as scientific goals demand great sharpness of resolution, the eye proves to be insufficient.
>
> ... I am confident that my ideography can be successfully used wherever special value must be placed on the validity of proofs, as for example when the foundations of the differential and integral calculus are established.

In logic and in linguistics, we distinguish between two languages: the one that is the object of study and the one that we use to talk about that object. The former is called the *object language*; the latter is the *metalanguage*. Below, the object language is the formal language of propositional formulas. The metalanguage is the usual informal language of mathematics and theoretical computer science, which is a mixture of the English language and mathematical notation. The importance of distinguishing between the object language and the metalanguage was emphasized by the mathematician and logician Alfred Tarski (1902–1983), who taught logic at Berkeley since 1942.

To summarize, logic is the study of reasoning. The British mathematician and philosopher George Boole (1815–1864) is the man who made logic mathematical. His book *The Mathematical Analysis of Logic* was published in 1847. Mathematical logic has been inspired by the search of formal system to distinguish between a correct and incorrect mathematical argument. One of the goals in this class is to learn what makes a mathematical argument correct. We start this inquiry by learning basics about propositional and first-order logic.

Preliminaries Let us review a notion of a *recursive or inductive definition*.

We use a recursive definition to define the elements in a set in terms of other elements in the set. For instance, we can define a set of Natural numbers as follows:

Base case :	0 is a Natural number
Inductive case :	a successor of n is a natural number, if n is a Natural number

A recursive definition of a function defines values of the functions for some inputs in terms of the values of the same function for other inputs. For example, the factorial function $n!$ is defined as follows:

$$0! = 1.$$
$$(n+1)! = (n+1) \cdot n!.$$

The definition may also be thought of as giving a procedure describing how to construct the function $n!$, starting from $n = 0$ and proceeding onwards with $n = 1,\ n = 2,\ n = 3$ etc..

Exercise 16. *Compute* 3!. *Compute* 5!.

2.1 Syntax and Semantics of Propositional Formulas

A *proposition* is a declarative sentence (i.e., a sentence that declares a fact) that is either true or false, but not both.

Sample propositions:

- Swimming at the New Jersey shore is allowed.
- The moon is made of green cheese.
- Kids are home.
- It is evening.
- Mom is happy.

Sample non-propositions:

- How old are you?
- $x + 2x = 3$
- Go straight!

Propositional Formulas A *propositional signature* is a set of symbols called *atoms* or *propositional variables*. (In examples, we will assume that p, q, r are atoms.) Atoms, or propositional symbols are used to encode propositions.

The symbols

$$\wedge \quad \vee \quad \to \quad \leftrightarrow \quad \neg \quad \bot \quad \top$$

are called *propositional connectives*. Among them, the symbols

- $\wedge$ *(conjunction)*, $\vee$ *(disjunction)*, $\to$ *(implication)* and $\leftrightarrow$ *(equivalence)* are called *2-place*, or *binary* connectives;
- $\neg$ *(negation)* is a *1-place*, or *unary* connective;
- $\bot$ *(false)* and $\top$ *(true)* are *0-place*.

In K. Rosen textbook symbol $\top$ is denoted by letter T and symbol $\bot$ is denoted by letter F.

Take a propositional signature σ which contains neither the propositional connectives nor the parentheses $(,)$. The alphabet of propositional logic consists of the atoms from σ, the propositional connectives, and the parentheses. By a *string* we understand a finite string of symbols in this alphabet. We define when a string is a *(propositional) formula* (or, *compound proposition* by K. Rosen textbook) recursively, as follows:

- every atom is a formula,
- both 0-place connectives are formulas,

- if F is a formula then $\neg F$ is a formula,
- for any binary connective $\odot$, if F and G are formulas then $(F \odot G)$ is a formula.

For instance,

$$\neg(p \to q)$$

and

$$(\neg p \to q) \tag{3}$$

are formulas; the string

$$\neg p \to q \tag{4}$$

is not a formula. But we now introduce a convention according to which (4) can be used as an abbreviation for (46). We will abbreviate formulas of the form $(F \odot G)$ by dropping the outermost parentheses in them. We will also agree that $\leftrightarrow$ has a lower binding power than the other binary connectives. For instance,

$$p \vee q \leftrightarrow p \to r$$

will be viewed as shorthand for

$$((p \vee q) \leftrightarrow (p \to r)).$$

Finally, for any formulas $F_1, F_2, \ldots, F_n$ $(n > 2)$,

$$F_1 \wedge F_2 \wedge \cdots \wedge F_n$$

will stand for

$$(\cdots (F_1 \wedge F_2) \wedge \cdots \wedge F_n).$$

The abbreviation $F_1 \vee F_2 \vee \cdots \vee F_n$ will be understood in a similar way.

Semantics of Propositional Formulas The symbols f and t are called *truth values*. An *interpretation* of a propositional signature σ is a function from σ into $\{\mathsf{f}, \mathsf{t}\}$. If σ is finite then an interpretation can be defined by the table of its values, for instance:

p	q	r
f	f	t

(5)

The semantics of propositional formulas that we are going to introduce defines which truth value is assigned to a formula F by an interpretation I.

As a preliminary step, we need to associate functions with all unary and binary connectives: a function from $\{\mathsf{f}, \mathsf{t}\}$ into $\{\mathsf{f}, \mathsf{t}\}$ with the unary connective $\neg$, and a function from $\{\mathsf{f}, \mathsf{t}\} \times \{\mathsf{f}, \mathsf{t}\}$ into $\{\mathsf{f}, \mathsf{t}\}$ with each of the binary connectives. These functions are denoted by the same symbols as the corresponding connectives, and defined by the following tables:

x	$\neg(x)$
f	t
t	f

x	y	$\wedge(x,y)$	$\vee(x,y)$	$\to(x,y)$	$\leftrightarrow(x,y)$
f	f	f	f	t	t
f	t	f	t	t	f
t	f	f	t	f	f
t	t	t	t	t	t

For any formula F and any interpretation I, the truth value F^I that is *assigned* to F by I is defined recursively, as follows:

- for any atom F, $F^I = I(F)$,
- $\bot^I = \mathsf{f}$, $\top^I = \mathsf{t}$,
- $(\neg F)^I = \neg(F^I)$,
- $(F \odot G)^I = \odot(F^I, G^I)$ for every binary connective $\odot$.

If $F^I = \mathsf{t}$ then we say that the interpretation I *satisfies* F (symbolically, $I \models F$).

Consider an exercise: Find a formula F of the signature $\{p, q, r\}$ such that (5) is the only interpretation satisfying F. One such formula is $(\neg p \wedge \neg q) \wedge r$. Indeed, let us denote (5) by I_1. Then,

$$
\begin{array}{l}
((\neg p \wedge \neg q) \wedge r)^{I_1} = \\
\wedge((\neg p \wedge \neg q)^{I_1}, r^{I_1}) = \\
\wedge(\wedge((\neg p)^{I_1}, (\neg q)^{I_1}), \mathsf{t}) = \\
\wedge(\wedge(\neg(p^{I_1}), \neg(q^{I_1})), \mathsf{t}) = \\
\wedge(\wedge(\neg(\mathsf{f}), \neg(\mathsf{f}), \mathsf{t}) = \\
\wedge(\wedge(\mathsf{t}, \mathsf{t}), \mathsf{t}) = \\
\wedge(\mathsf{t}, \mathsf{t}) = \mathsf{t}
\end{array}
$$

If the underlying signature is finite then the set of interpretations is finite also, and the values of F^I for all interpretations I can be represented by a finite table. This table is called the *truth table* of F. For instance, the exercise above can be stated as follows: Find a formula with the truth table

p	q	r	F
f	f	f	f
f	f	t	t
f	t	f	f
f	t	t	f
t	f	f	f
t	f	t	f
t	t	f	f
t	t	t	f

where first three truth values of each raw represent a possible interpretation of the signature $\{p, q, r\}$, and the last column shows respective truth values assigned to a formula by a respective interpretation.

Propositional formulas and truth tables were introduced in 1921 by the American mathematician and logician Emil Post (1897–1954). Post is known, along with Alan Turing, as one of the creators of theoretical computer science.

2.2 Tautologies, Equivalence, Satisfiability and Entailment

Tautologies A propositional formula F is a *tautology* if every interpretation satisfies F. It is easy to check, for instance, that each of the formulas

$$\begin{array}{c}(p \to \bot) \leftrightarrow \neg p,\\ (p \to q) \vee (q \to p),\\ ((p \to q) \to p) \to p,\\ (p \to (q \to r)) \to ((p \to q) \to (p \to r))\end{array} \tag{6}$$

is a tautology. Here we illustrate this fact for the case of the formula

$$(p \to \bot) \leftrightarrow \neg p. \tag{7}$$

Consider a truth table for this formula

p	$\neg p$	$\bot$	$p \to \bot$	$(p \to \bot) \leftrightarrow \neg p$
f	t	f	t	t
t	f	f	f	t

(8)

It shows that for every possible interpretation of the signature $\{p\}$ of this formula, the truth value that is assigned to $(p \to \bot) \leftrightarrow \neg p$ is t.

Exercise 17. *Show that the last three formulas in* (6) *are tautologies. Construct truth tables in your argument.*

A propositional formula F is a *contradiction* if there is no interpretation that satisfies F.

Exercise 18. *Does statement*

for any tautology F, formula $\neg F$ is a contradiction

hold?

Equivalent Formulas A formula F is *equivalent* to a formula G (symbolically, $F \equiv G$) if, for every interpretation I, $F^I = G^I$. In other words, the metalanguage expression $F \equiv G$ means that formula $F \leftrightarrow G$ is a tautology. For instance,

$$(p \to \bot) \equiv \neg p. \tag{9}$$

Recall that we used truth table (8) to illustrate that a formula (7) is a tautology. Obviously, the same truth table can be used to illustrate that two formulas in (9) are equivalent. Truth table (8) enumerates all possible cases. It is a form of what mathematicians call *reasoning by cases*. Below we provide a narrative exemplifying reasoning by cases for arguing that equivalence (9) holds. This argument bypasses the explicit reference to truth table (8) (but it is easy to reconstruct this truth table by studying the argument):

Consider any interpretation I. *Case 1:* $p^I = \mathsf{t}$. Then

$$\begin{array}{l}(p \to \bot)^I = \to (p^I, \bot^I) = \to (\mathsf{t}, \mathsf{f}) = \mathsf{f}\\ =\\ (\neg p)^I = \neg(p^I) = \neg(\mathsf{t}) = \mathsf{f}\end{array}$$

Case 2: $p^I = \mathsf{f}$. Then

$$\begin{array}{l}(p \to \bot)^I = \to (p^I, \bot^I) = \to (\mathsf{f}, \mathsf{f}) = \mathsf{t}\\ =\\ (\neg p)^I = \neg(p^I) = \neg(\mathsf{f}) = \mathsf{t}\end{array}$$

Also, it is easy to see that the following equivalence holds

$$\neg\bot \equiv \top \tag{10}$$

Indeed, given any interpretation I,

$$(\neg\bot)^I = \neg(\bot^I) = \neg(\mathsf{f}) = \mathsf{t} = \top^{\mathsf{I}}.$$

Similarly, following equivalence holds

$$F \wedge \neg F \equiv \bot, \tag{11}$$

where F is an arbitrary formula. Indeed, consider any interpretation I. *Case 1:* $F^I = \mathsf{t}$. Then

$$(F \wedge \neg F)^I = \wedge(F^I, \neg(F^I)) = \wedge(\mathsf{t}, \neg(\mathsf{t})) = \wedge(\mathsf{t}, \mathsf{f}) = \mathsf{f} = \bot^{\mathsf{I}}.$$

Case 2: $F^I = \mathsf{f}$. Then

$$(F \wedge \neg F)^I = \mathsf{f} = \bot^{\mathsf{I}}.$$

Exercise 19. *Prove that*

$$F \vee \neg F \equiv \top \tag{12}$$

holds.

Conjunction and disjunction are associative:

$$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H),$$
$$(F \vee G) \vee H \equiv F \vee (G \vee H).$$

Exercise 20. *Does equivalence connective $\leftrightarrow$ have a similar property?*

Conjunction distributes over disjunction:

$$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H);$$

disjunction distributes over conjunction:

$$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H).$$

Exercise 21. *Do these connectives distribute over equivalence connective $\leftrightarrow$?*

De Morgan's laws follow

$$\neg(F \wedge G) \equiv \neg F \vee \neg G,$$
$$\neg(F \vee G) \equiv \neg F \wedge \neg G$$

Implication distributes over conjunction:

$$F \to (G \wedge H) \equiv (F \to G) \wedge (F \to H).$$

Exercise 22. *To simplify a formula means to find an equivalent formula that is shorter. Simplify the following formulas:*

(i) $F \leftrightarrow \neg F$

(ii) $F \vee (F \wedge G)$,

(iii) $F \wedge (F \vee G)$,

(iv) $F \vee (\neg F \wedge G)$.

(v) $F \wedge (\neg F \vee G)$.

Exercise 23. *For each of the formulas*

$$p \wedge q,\ p \vee q,\ p \leftrightarrow q,\ \neg p, \top$$

find an equivalent formula that contains no connectives other than $\rightarrow$ *and* $\bot$.

Exercise 24. *For each of the formulas*

$$p \rightarrow q,\ p \wedge q$$

find an equivalent formula that contains no connectives other than $\leftrightarrow$ *and* $\vee$.

A *subformula* of a formula F is a substring of F such that this substring is also a formula. For instance, $(p \rightarrow \bot)$ is a subformula of $p \wedge (p \rightarrow \bot)$. Obviously, formula F is a subformula of itself.

It turns out that for any equivalent formulas G and G', substituting G' for subformula G in formula F results in an equivalent formula to F. For instance, substituting $\neg p$ for $(p \rightarrow \bot)$ in

$$p \wedge (p \rightarrow \bot) \tag{13}$$

results in a formula $p \wedge \neg p$ which is equivalent to (13).

This observation gives us additional method for demonstrating that formulas are equivalent. We can show that formula F is equivalent to formula G by applying a sequence of equivalent transformations on its subformulas. For instance, by (9), (10), and (11) it follows that

$$\neg(p \wedge (p \rightarrow \bot)) \equiv \top.$$

Indeed,

$$\neg(p \wedge (p \rightarrow \bot)) \equiv \neg(p \wedge \neg p) \equiv \neg(\bot) \equiv \top.$$

Satisfiability and Entailment A set Γ of formulas is *satisfiable* if there exists an interpretation that satisfies all formulas in Γ, and *unsatisfiable* otherwise. It is easy to check that a non-empty finite set $\{F_1, \ldots, F_n\}$ is unsatisfiable iff the negation of the conjunction of its elements, i.e., a formula $\neg(F_1 \wedge \cdots \wedge F_n)$, is a tautology.

For any atom A, the literals A, $\neg A$ are said to be *complementary* to each other.

A set Γ of formulas *entails* a formula F (symbolically, $\Gamma \models F$), if every interpretation that satisfies all formulas in Γ satisfies F also. Note that the notation for entailment uses the same symbol as the notation for satisfaction introduced earlier, the difference being that the expression on the left is an interpretation (I) in one case and a set of formulas (Γ) in the other. The formulas entailed by Γ are also called the *logical consequences* of Γ.

Exercise 25. *Prove that* $F_1, \ldots, F_n \models G$ *iff* $(F_1 \wedge \cdots \wedge F_n) \rightarrow G$ *is a tautology. (In the first of these expressions, we dropped the braces* $\{\}$ *around* $F_1, \ldots, F_n$*.)*

It is easy to check that for any set Γ of formulas and any formula F, $\Gamma \models F$ iff the set $\Gamma \cup \{\neg F\}$ is unsatisfiable.

Exercise 26. *Show that for any set* Γ *of formulas and any formula* F, $\Gamma \models F$ *iff the set* $\Gamma \cup \{\neg F\}$ *is unsatisfiable.*

2.3 Inference

Representing English Sentences by Propositional Formulas Before we continue the study of propositional logic, we will attempt translating some declarative sentences from English into the language of propositional formulas. The translation exercises below are not precisely stated mathematical questions: there is no way to prove, in the mathematical sense, that a translation is adequate. For example, natural language expressions are often ambiguous and their interpretation may depend on subtle details of a context that may go beyond elements (concepts) captured by the expressions themselves. Expressions in propositional logic are unambiguous.

In these translation exercises, the underlying signature is

$$\{p, q, r\}, \tag{14}$$

where we take each propositional symbol to represent following English expressions

p	Kids are home
q	It is evening
r	Mom is happy

(15)

The table below provides sample translations of some English sentences given propositional logic connectives and the propositional signature (14).

It is evening and kids are not home	$q \wedge \neg p$
It is evening but kids are not home	$q \wedge \neg p$
It is either evening or mom is not happy	$q \vee \neg r$
If kids are home then mom is happy	$p \rightarrow r$
Mom is happy only if kids are home	$r \rightarrow p$

(16)

Notice how the first two sentences are mapped into the same expression in propositional logic. This is an illustration of how a translation from a natural language to a formal language is typically a "lossy" process. There is no connective in propositional logic that is able to reflect all the information carried by the natural language connective *but*.

In the third sentence, *either, or* connective can be read in two ways – "exclusive" or "non-exclusive". Our translation binds us to a single "non-exclusive" reading.

Translation of "if then" and " only if" natural language conditional connectives is captured by the implication connective of propositional logic. The implication connective may not at all times reflect our intuitions about if-then statements and how we interpret these. For example, statement *If a moons is made of green cheese then Earth is square* does not seem to carry a truth-conditional meaning, i.e., it seems "gibberish". Yet if we take propositional symbols p and q to denote "a moons is made of green cheese" and " Earth is square" respectively then $p \rightarrow q$ is a statement that is evaluated to $\mathtt{t}$ given that p is $\mathtt{f}$.

Despite the outlined issues with the mapping from natural language statements into a formal language of propositional logic, capturing English expressions as formal language statements allows us (i) to present these in unambiguous way and (ii) to apply formal notions of interpretation and inference to them. In the next section we will speak of a validity of a natural language argument by grounding this question into a notion of a validity of inference in propositional logic. We then discuss how tradition of mathematical proofs builds on these notions.

Argument and Argument Form A (natural language) *argument* is a sequence of statements

$$s_1, s_2, \ldots, s_n.$$

The last statement s_n is called a *conclusion*. The statements $s_1, \ldots s_n$ are called *premises*. For instance,

	If kids are home then mom is happy
	Kids are home
$\therefore$	Mom is happy

(17)

is an argument. Note that we list each statement of this argument in a separate line and separate its conclusion from its premises by a vertical bar. In the sequel, we will often present arguments (as well as argument forms that we introduce next) in such format.

In a valid argument the conclusion of the argument must follow from the truth of argument's premises: an argument is *valid* if and only if it is impossible for all the premises to be true and the conclusion to be false. In previous section we illustrated how we can translate (ground) natural language statements into corresponding propositional formulas. We will now show how grounding natural language statements of an argument into propositional formulas allows us to turn an analysis of a validity of an argument into a formal question about semantic properties of these formulas. We start by introducing a notion of an argument form and defining in precise mathematical terms its validity. We then show how argument forms are used as mathematical abstractions of natural language arguments. Thus, establishing the validity of an argument form immediately translates into the fact that its natural language argument counterpart is valid.

An *argument form* is a sequence of propositional formulas

$$F_1, F_2, \ldots, F_n.$$

Intuitively, formula F_n is called a *conclusion* and $F_1, \ldots, F_{n-1}$ are called *premises*.

An argument form

$$\begin{array}{ll} & F_1 \\ & \cdots \\ & F_{n-1} \\ \hline \therefore & F_n \end{array} \tag{18}$$

is *valid* when for every interpretation I that satisfies $F_1, \ldots, F_{n-1}$ (i.e., $F_1^I = \cdots = F_{n-1}^I = \mathsf{t}$), I satisfies F_n also (i.e., $F_n^I = \mathsf{t}$.) In other words, an argument form (18) is valid if its premises $F_1, \ldots F_{n-1}$ entail its conclusion F_n (symbolically, $F_1, \ldots F_{n-1} \models F_n$). It turns out that

$$F_1, \ldots F_{n-1} \models F_n \text{ if and only if } (F_1 \wedge \cdots \wedge F_{n-1}) \rightarrow F_n \text{ is a tautology.}$$

Consequently, an argument form (18) is valid when $(F_1 \wedge \cdots \wedge F_{n-1}) \rightarrow F_n$ is a tautology.

Exercise 27. *Show that given formulas* $F_1, \ldots F_{n-1} \models F_n$,

$$F_1, \ldots F_{n-1} \models F_n \textit{ if and only if } (F_1 \wedge \cdots \wedge F_{n-1}) \rightarrow F_n \textit{ is a tautology.}$$

The validity of an argument form can be demonstrated by use of a truth table. For instance, consider an argument form

$$\begin{array}{ll} & F \\ & F \rightarrow G \\ \hline \therefore & G \end{array} \tag{19}$$

Its truth table follows

F	G	$F \rightarrow G$
$\underline{\mathsf{t}}$	t	$\underline{\mathsf{t}}$
t	f	f
f	t	t
f	f	t

(20)

Let I be any interpretation that evaluates both premises of argument form (19) to true. The first raw of truth table (20) represents this case. Interpretation I evaluates conclusion G of (19) to true also. Indeed, $G^I = \mathsf{t}$. By the definition of validity, argument form (19) is valid.

A Deduction System and Valid Argument Forms We can always use truth tables to establish a validity of an argument form. Yet, for complex argument forms this approach may be intractable. Alternatively, "deductive systems" – collections of inference rules and axioms – can be used to illustrate the validity of an argument form. A "derivation" in a deductive system shows how a formula (a conclusion) can be derived from a set of other formulas (premises), often called hypothesis, using the postulates (inference rules

Valid Argument Form	Name
F / $\therefore\ \neg\neg F$	Double negation introduction
$\neg\neg F$ / $\therefore\ F$	Double negation elimination
F, $F \to G$ / $\therefore\ G$	Modus ponens (the law of detachment)
$\neg G$, $F \to G$ / $\therefore\ \neg F$	Modus tollens
$F \to G$, $G \to H$ / $\therefore\ F \to H$	Hypothetical syllogism
$F \vee G$, $\neg F$ / $\therefore\ G$	Disjunctive syllogism
F / $\therefore\ F \vee G$	Addition
$F \wedge G$ / $\therefore\ F$	Simplification
F, G / $\therefore\ F \wedge G$	Conjunction
$\neg G \to \neg F$ / $\therefore\ F \to G$	Contraposition
$F \vee G$, $\neg F \vee H$ / $\therefore\ G \vee H$	Resolution

Figure 4: Valid argument forms.

and axioms) of the system. Such derivations are called proofs. We introduce a sample deductive system to show how we can argue the validity of argument forms by means of this deduction system. In this deductive system valid argument forms presented in Figure 4 constitute inference rules or postulates of the system. We now make these ideas precise.

Exercise 28. *Use truth tables to establish the fact that the argument forms Double negation introduction, Double negation elimination, Modus ponens, Modus tollens, Hypothetical syllogism, Disjunctive syllogism, Addition, Simplification, Conjunction, Resolution, Contraposition are valid.*

Let Γ be a set of formulas. A *valid argument form derivation from* Γ (or, vaf-derivation from Γ) is a list $G_1, \ldots G_m$ $(m \geq 1)$ of formulas such that, for every $i \leq m$, formula G_i

1. is in Γ, or
2. is of the form $F \vee \neg F$ or of the form $F \to F$, or
3. is derived by a valid argument form in Figure 4 whose conclusion is G_i and all premises occur in $G_1, \ldots, G_{i-1}$.

A vaf-derivation whose last formula is G is said to be a *vaf-derivation of* G.

For instance, let Γ be set $\{F, F \to G\}$ of formulas then three lists

$$F$$
$$F \to G,\ F$$
$$F \to G,\ F,\ G$$

are among vaf-derivations from Γ. The vaf-derivation $F \to G,\ F,\ G$ is also a vaf-derivation of G. We now present this vaf-derivation horizontally annotating its elements:

1. $F \to G$	an element of Γ (clause 1 of the vaf-derivation definition)
2. F	an element of Γ
3. G	Modus ponens on 1 and 2 (clause 3 of the vaf-derivation definition)

These annotations make it clear why we call this list of formulas a vaf-derivation from Γ.

Fact For any set Γ of formulas and any formula G, if there is a derivation of G from Γ then

i. Γ entails G,

ii. $\bigwedge_{F \in \Gamma} F \to G$ is a tautology, where we understand $\bigwedge_{F \in \Gamma} F$ as a conjunction of elements in Γ, and

iii. $\dfrac{\Gamma}{\therefore \quad G}$ is a valid argument form.

Thus, to argue the validity of an argument form (18) it is sufficient to find a vaf-derivation of conclusion F_n of (18) from the set $\{F_1, \ldots, F_{n-1}\}$ composed of the premises of (18). It is also worth to note that we can extend the list of argument forms in Figure 4 by arbitrary *valid* argument forms and the claims above would still hold.

Grounding Natural Language Arguments into Argument Forms By following the translation process outlined in Section 2.3, we can create argument forms for given arguments. For example, an argument form

$$\begin{array}{ll} & p \to r \\ & p \\ \hline \therefore & r \end{array} \tag{21}$$

corresponds to argument (17) if we follow agreement that propositional symbols p, r stand for English expressions as presented in (15). It is easy to see that there is a vaf-derivation of r from premises of (21). Indeed, consider sequence:

1.	$p \to r$	Premise (Assumption)
2.	r	Premise
3.	r	Modus ponens 1 and 2

Consequently, argument form (21) is valid. Under assumption that argument form (21) adequately captures natural language argument (17) we conclude that (17) is valid.

Similarly, an argument

$$\begin{array}{ll} & \text{Kids are home.} \\ \hline \therefore & \text{Kids are home or mom is not happy.} \end{array} \tag{22}$$

corresponds to the following argument form

$$\begin{array}{ll} & p \\ \hline \therefore & p \vee \neg r \end{array} \tag{23}$$

The vaf-derivation of $p \vee r$ from p follows:

1.	p	Premise (Assumption)
2.	$p \vee \neg r$	Addition 1

Consequently, argument form (23) is valid. Under assumption that argument form (23) adequately captures natural language argument (22) we conclude that (22) is valid.

From Arguments to Mathematical Proofs

In Mathematics,

- a *conjecture* is a statement believed to be true,
- a *theorem* is a statement that can be shown to be true,
- a *proof* is an evidence supporting that a statement is true,
- an *axiom (postulates)* are statements that are assumed to be true.[1]

Construction of a mathematical proof relies on (i) definitions of concepts in question, (ii) axioms, (iii) valid argument forms or inference rules.

In a formal proof process, one starts by formulating a conjecture as an argument that is then mapped into an argument form. Next, one may use truth tables, natural deduction system, or notion of a vaf-derivation to verify whether an argument form is valid. Such verification process translates into evidence supporting the conjecture. Thus the conjecture becomes a theorem. Such process is used by formal automated theorem proving systems.

In English, textbook mathematical proofs many of these steps are implicit. One or another valid argument form (listed in Figure 4) is often used in producing an English counterpart of vaf-derivation, but no explicit reference is given. Sometimes, English expressions such as *therefore, thus, consequently, we derive, it follows that* indicate the use of a valid argument form in deriving a new statement. Similarly, "well-known" properties of specific mathematical theories are at times also taken for granted in constructing arguments and are not mentioned explicitly.

To illustrate the difference between a formal proof process and a typical mathematical proof let us consider following conjecture:

> Given that (i) *if kids are home then mom is happy*, (ii) *it is either not evening or kids are home*, and (iii) *it is evening*, the statement (iv) *mom is happy* holds.

Sample formal proof process. Let propositional symbols p, q, r stand for English expressions as presented in (15). Then the statement (argument) of the conjecture in question corresponds to the following argument form

$$\begin{array}{ll} & p \to r \\ & \neg q \vee p \\ & q \\ \hline \therefore & \mathrm{r} \end{array}$$

The vaf-derivation below illustrates that this argument form is valid (or, in other words, that formula $((p \to r) \wedge (\neg q \vee p) \wedge q) \to r$ is a tautology):

[1] For instance, recall Peano's Axioms `http://mathworld.wolfram.com/PeanosAxioms.html` – axioms for the natural numbers:

1. Zero – 0 – is a number.
2. If n is a number, the successor of n, denoted by n' or $n + 1$, is a number.
3. Zero is not the successor of a number.
4. Two numbers of which the successors are equal are themselves equal.
5. (induction axiom.) If zero has the property P [Basis], and if the successor of every number with this property P has this property P also [Induction Step], then all numbers have this property.

1. q	Premise (Assumption)
2. $\neg\neg q$	Double negation introduction 1
3. $\neg q \vee p$	Premise
4 p	Disjunctive syllogism 2,3
5 $p \to r$	Premise
6. r	Modus ponens 4,5

Consequently, the conjecture in question holds.

Sample textbook mathematical proof. From the fact that (ii) and (iii) hold, we conclude that *kids are home*. From this conclusion and given fact (i) it follows that statement (iv) holds.

2.4 Elements of Predicate Logic

Propositional logic is inadequate to express the meaning and support natural inferences for many statements in mathematics and natural language. For example, recall one of the axioms about integers:

> (i) *The set of integers Z is closed under the operations of addition and multiplication, that is, the sum and product of any two integers is an integer.*

Consider a statement

> (ii) *n and m are integers.*

What would you conclude from (i) and (ii)? *Predicate logic* is the field of mathematics that

(1) allows to express the meaning of above statements and

(2) supports the natural inference suggesting that *$n + m$ and $n * m$ are integers.*

Section 2.5 provides a formal account for predicate logic: definitions of its syntax and semantics. In this section we discuss the intuitions behind the language of predicate logic.

Connectives of propositional logic — $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ — are also connectives in predicate logic. Two *quantifiers*

$$\forall \text{ (“for all”)}, \quad \exists \text{ (“there exists”)}$$

are important additions to the predicate logic.

The logical formula

$$\forall x \left((x+2)^2 = x^2 + 4x + 4\right)$$

expresses that the equality $(x + 2)^2 = x^2 + 4x + 4$ holds for all values of x. The logical formula

$$\exists x(x^2 + 2x + 3 = 0)$$

expresses that the equation $x^2 + 2x + 3 = 0$ has at least one solution. The assertion “there exists a negative number x such that its square is 2” can be written as

$$\exists x(x < 0 \wedge x^2 = 2).$$

The symbol $\forall$ is called the *universal* quantifier; the symbol $\exists$ is the *existential* quantifier.

Statements involving variables such as $x < 0$ or $x^2 = 2$ are neither true nor false when the values of variables are not specified. Thus they are not propositions. Yet, the quantifiers of predicate logic can be used to produce propositions from such statements. Indeed, expression

$$\forall x(x < 0) \tag{24}$$

forms a proposition “for all values of x, x is such that it is less than 0”. If *domain (of discourse)* or *universe of discourse* is such that x can take any value in $\mathbb{N}$ (natural numbers), then proposition (24) is false: consider $x = 1$.

The statement $x < 0$ or, in other words,

$$x \text{ is less than } 0, \tag{25}$$

has two parts. The first part is a variable x – the subject of the statement. The second part is the “predicate”:

$$\textit{is less than } 0 \tag{26}$$

that refers to a property that the subject of the statement can have. We can denote statement (25) by $P(x)$, where P denotes (26). A propositional statement that is parametrized on one or more variables is called a *predicate*. The statement $P(x)$ is also called the value of the *propositional function* P at x. Once a value is assigned to x, the statement $P(x)$ becomes a proposition and has a truth value. For example, the truth value of $P(1)$ is false, whereas the truth value of $P(-1)$ is true.

In predicate logic, the truth tables for connectives from propositional logic together with the rules for interpreting universal and existential quantifiers show how we can determine whether a formula is true.

Statement	When True?
$\forall x P(x)$	$P(v)$ is true for every value v of x (in the considered domain)
$\exists x P(x)$	There is a value v of x for which $P(v)$ is true
Statement	**When False?**
$\forall x P(x)$	There is a value v of x for which $P(v)$ is false
$\exists x P(x)$	$P(v)$ is false for every value v of x

Witnesses and Counterexamples How can we argue that a logical formula beginning with $\exists x$ is true? This can be done by specifying a value of x that satisfies the given condition. Such a value is called a *witness*. For instance, to argue that the assertion

$$\exists x(4 < x^2 < 5)$$

is true, we can use the value $x = 2.1$ as a witness.

How can we argue that a logical formula beginning with $\forall x$ is false? This can be done by specifying a value of x that does *not* satisfy the given condition. Such a value is called a *counterexample*. For instance, to argue that the assertion

$$\forall x(x^2 \geq x) \tag{27}$$

is false, we can use the value $x = \frac{1}{2}$ as a counterexample.

The assertion that formula (27) is false can be expressed in two ways: by the negation of formula (27):

$$\neg\forall x(x^2 \geq x)$$

and also by saying that there exists a counterexample:

$$\exists x\neg(x^2 \geq x).$$

This is an instance of a general fact: the combinations of symbols

$$\neg\forall x \quad \text{and} \quad \exists x\neg$$

have the same meaning. This is one of the two *De Morgan's laws for quantifiers*. The table below presents these laws:

Negation	Equivalent Statement	When is Negation True
$\neg\forall x P(x)$	$\exists x\neg P(x)$	There is a value v of x for which $P(v)$ is false
$\neg\exists x P(x)$	$\forall x\neg P(x)$	For every value v of x, $P(v)$ is false
Negation	**Equivalent Statement**	**When is Negation False**
$\neg\forall x P(x)$	$\exists x\neg P(x)$	$P(v)$ is true for every value v of x
$\neg\exists x P(x)$	$\forall x\neg P(x)$	There is a value v of x for which $P(v)$ is true

In the following example we use the notation $m|n$ to express that the integer m divides the integer n with no remainder (or, in other words, that n is a multiple of m, e.g., $4, 6$ are multiples of 2). We want to show that the assertion

$$\forall n(2|n \vee 3|n)$$

("every integer is a multiple of 2 or a multiple of 3") is false. What value of n can be used as a counterexample? We want the formula $2|n \vee 3|n$ to be false. According to the truth table for disjunction, this formula is false only when both $2|n$ and $3|n$ are false. In other words, n should be neither a multiple of 2 nor a multiple of 3. The simplest counterexample is $n = 1$.

The formula

$$\exists mn(m^2 + n^2 = 10)$$

expresses that 10 can be represented as the sum of two complete squares. It is different from the formulas with quantifiers that we have seen before in that the existential quantifier is followed here by two variables, not one. To give a witness justifying this claim, we need to find a pair of values m, n such that $m^2 + n^2 = 10$. For instance, the pair of values $m = 3$, $n = 1$ provides a witness.

Free and Bound Variables When a formula begins with $\forall x$ or $\exists x$, we say that the variable x is *bound* in it. If a quantifier is followed by several variables then all of them are bound. When a variable is not bound then we say that it is *free* in the formula. For instance, in the formula

$$\exists i(j = i^2) \tag{28}$$

the variable i is bound, and the variable j is free. In the formula

$$\exists ij(j^2 + j^2 = 29) \tag{29}$$

both i and j are bound.

The difference between free and bound variables is important because the truth value of a formula depends on the values of its free variables, but doesn't depend on the values of its bound variables. For instance, formula (28) expresses that j is a complete square; whether or not it is true depends on the value of its free variable j. Formula (29) expresses that 29 can be represented as the sum of two squares; we don't need to specify the values of any variables before we ask whether this formula is true.

Replacing a bound variable by a different variable doesn't change the meaning of a formula. For instance, the formula $\exists k(j = k^2)$ has the same meaning as (28): it says that j is a complete square.

Quantifiers are not the only mathematical symbols that bind variables. One other example is Sigma notation for sums of numbers. For instance, the sum of the squares of the numbers from 1 to n can be written as

$$\sum_{i=1}^{n} i^2.$$

Here n is a free variable, and i is a bound variable. The value of this expression depends on n, but not on i.

2.5 Predicate Logic, Formally

Predicate Formulas "Predicate formulas" generalize the concept of a propositional formula defined earlier.

A *predicate signature* is a set of symbols of two kinds—*object constants* and *predicate constants*—with a nonnegative integer, called the *arity*, assigned to every predicate constant. A predicate constant is said to be *propositional* if its arity is 0. Propositional constants are similar to atoms in propositional logic. A predicate constant is *unary* if its arity is 1, and *binary* if its arity is 2. For instance, we can define a predicate signature

$$\{a, P, Q\} \tag{30}$$

by saying that a is an object constant, P is a unary predicate constant, and Q is a binary predicate constant.

Take a predicate signature σ that does not include any of the following symbols:

- the *object variables* $x, y, z, x_1, y_1, z_1, x_2, y_2, z_2, \ldots,$
- the propositional connectives,
- the *universal quantifier* $\forall$ and the *existential quantifier* $\exists$,
- the parentheses and the comma.

The alphabet of predicate logic consists of the elements of σ and of the four groups of additional symbols listed above. A *string* is a finite string of symbols in this alphabet.

A *term* is an object constant or an object variable. A string is called an *atomic formula* if it is a propositional constant or has the form

$$R(t_1, \ldots, t_n)$$

where R is a predicate constant of arity n ($n > 0$) and $t_1, \ldots, t_n$ are terms. For instance, if the underlying signature is (30) then $P(a)$ and $Q(a, x)$ are atomic formulas.

We define when a string is a *(predicate) formula* recursively, as follows:

- every atomic formula is a formula,
- both 0-place connectives are formulas,
- if F is a formula then $\neg F$ is a formula,
- for any binary connective $\odot$, if F and G are formulas then $(F \odot G)$ is a formula,
- for any quantifier K and any variable v, if F is a formula then KvF is a formula.

For instance, if the underlying signature is (30) then

$$(\neg P(a) \vee \exists x(P(x) \wedge Q(x, y)))$$

is a formula.

When we write predicate formulas, we will drop some parentheses and use other abbreviations introduced in previous section. A string of the form $\forall v_1 \cdots \forall v_n$ ($n \geq 0$) will be written as $\forall v_1 \cdots v_n$, and similarly for the existential quantifier.

An occurrence of a variable v in a formula F is *bound* if it occurs in a part of F of the form KvG; otherwise it is *free* in F. For instance, in the formula

$$\exists y P(x, y) \wedge \neg \exists x P(x, x) \tag{31}$$

the first occurrence of x is free, and the other three are bound; both occurrences of y in (31) are bound. We say that v is *free (bound)* in F if some occurrence of v is free (bound) in F. A formula without free variables is called a *closed formula*, or a *sentence*.

Representing English Sentences by Predicate Formulas Before we continue the study of the syntax of predicate logic, it is useful to get some experience in translating sentences from English into the language of predicate formulas. The translation exercises below are different from the other problems in that they are not precisely stated mathematical questions: there is no way to prove, in the mathematical sense, that a translation is adequate (at least until the semantics of predicate logic is defined). There is sometimes no clear-cut difference between good and bad translations; it may happen that one translation is somewhat less adequate than another but still satisfactory.

In these translation exercises, the underlying signature is (30). We will think of object variables as ranging over the set ω of nonnegative integers, and interpret the signature as follows:

- a represents the number 10,
- $P(x)$ represents the condition "x is a prime number,"
- $Q(x, y)$ represents the condition "x is less than y."

As an example, the sentence *All prime numbers are greater than* x can be represented by the formula

$$\forall y(P(y) \to Q(x, y)). \tag{32}$$

Exercise 29. *Represent the given English sentences by predicate formulas. (a) There is a prime number that is less than 10. (b) x equals 0. (c) x equals 9.*

Exercise 30. *Represent the English sentence by predicate formula: There are infinitely many prime numbers.*

Substitution Let F be a formula and v a variable. The result of the *substitution* of a term t for v in F is the formula obtained from F by replacing each free occurrence of v by t. When we intend to consider substitutions for v in a formula, it is convenient to denote this formula by an expression like $F(v)$; then we can denote the result of the substitution of a term t for v in this formula by $F(t)$. For instance, if we denote formula (31) by $F(x)$ then $F(a)$ stands for

$$\exists y P(a, y) \wedge \neg \exists x P(x, x).$$

Let $F(x)$ be formula (32) proposed above as a translation for the condition "all prime numbers are greater than x." A formula of the form $F(t)$, where t is a term, *usually* expresses the same condition applied to the value of t. For instance, $F(a)$ is

$$\forall y(P(y) \to Q(a, y)),$$

which means that all prime numbers are greater than 10; $F(z_2)$ is

$$\forall y(P(y) \to Q(z_2, y)),$$

which means that all prime numbers are greater than z_2. There is one exception, however. The formula $F(y)$, that is,

$$\forall y(P(y) \to Q(y, y)),$$

expresses the (incorrect) assertion "every prime number is less than itself." The problem with this substitution is that y, when substituted for x in $F(x)$, is "captured" by a quantifier. To express the assertion "all prime numbers are greater than y" by a predicate formula, we would have to use a bound variable different from y and write, for instance,

$$\forall z(P(z) \to Q(y, z)).$$

To distinguish "bad" substitutions, as in the last example, from "good" substitutions, we introduce the following definition. A term t is *substitutable* for a variable v in a formula F if

- t is a constant, or
- t is a variable w, and no part of F of the form KwG contains an occurrence of v which is free in F.

For instance, a and z_2 are substitutable in (32) for x, and y is not substitutable.

Semantics of Predicate Formulas The semantics of propositional formulas described in Section 1 defined which truth value F^I is assigned to a propositional formula F by an interpretation I. Our next goal is to extend this definition to predicate formulas. First we need to extend the definition of an interpretation to predicate signatures.

An *interpretation* I of a predicate signature σ consists of

- a non-empty set $|I|$, called the *universe* of I,
- for every object constant c of σ, an element c^I of $|I|$,
- for every propositional constant R of σ, an element R^I of $\{\mathsf{f}, \mathsf{t}\}$,
- for every predicate constant R of σ of arity $n > 0$, a function R^I from $|I|^n$ to $\{\mathsf{f}, \mathsf{t}\}$.

For instance, the second paragraph of the section on representing English sentences by predicate formulas can be viewed as the definition of an interpretation of signature (30). For this interpretation I,

$$|I| = \omega,$$

$$a^I = 10,$$

$$P^I(n) = \begin{cases} \mathsf{t}, & \text{if } n \text{ is prime,} \\ \mathsf{f}, & \text{otherwise,} \end{cases} \tag{33}$$

$$Q^I(m,n) = \begin{cases} \mathsf{t}, & \text{if } m < n, \\ \mathsf{f}, & \text{otherwise.} \end{cases}$$

The semantics of predicate logic introduced below defines the truth value F^I only for the case when F is a *sentence*. As in propositional logic, the definition is recursive. For propositional constants, there is nothing to define: the truth value R^I is part of the interpretation I. For other atomic sentences, we can define

$$R(t_1, \dots, t_n)^I = R^I(t_1^I, \dots, t_n^I).$$

(Since $R(t_1, \dots, t_n)$ is a sentence, each term t_i is an object *constant*, and consequently t_i^I is part of I). For propositional connectives, we can use the same clauses as in propositional logic. But the case of quantifiers presents a difficulty. One possibility would be to define

$$\exists w F(w)^I = \mathsf{t} \text{ iff, for some object constant } c,\ F(c)^I = \mathsf{t}$$

and similarly for the universal quantifier. But this formulation is unsatisfactory: it disregards the fact that some elements of the universe $|I|$ may be not represented by object constants. For instance, in the example above 10 is the only element of the universe ω for which there is a corresponding object constant in signature (30); we expect to find that

$$(\exists x Q(x, a))^I = \mathsf{t}$$

(there exists a number that is less than 10) although

$$Q(a, a)^I = \mathsf{f}$$

(10 does not have this property). Because of this difficulty, some additional work is needed before we define F^I.

Consider an interpretation I of a predicate signature σ. For any element ξ of its universe $|I|$, select a new symbol ξ^*, called the *name* of ξ. By σ^I we denote the predicate signature obtained from σ by adding all names ξ^* as additional object constants. For instance, if σ is (30) and I is (33) then

$$\sigma^I = \{a, 0^*, 1^*, 2^*, \dots, P, Q\}.$$

The interpretation I can be extended to the new signature σ^I by defining

$$(\xi^*)^I = \xi$$

for all $\xi \in |I|$. We will denote this interpretation of σ^I by the same symbol I.

We will define recursively the truth value F^I that is *assigned* to F by I for every sentence F of the extended signature σ^I; that includes, in particular, every sentence of the signature σ. For any propositional constant R, R^I is part of the interpretation I. Otherwise, we define:

- $R(t_1, \dots, t_n)^I = R^I(t_1^I, \dots, t_n^I)$,
- $\bot^I = \mathsf{f}$, $\top^I = \mathsf{t}$,
- $(\neg F)^I = \neg(F^I)$,
- $(F \odot G)^I = \odot(F^I, G^I)$ for every binary connective $\odot$,
- $\forall w F(w)^I = \mathsf{t}$ iff, for all $\xi \in |I|$, $F(\xi^*)^I = \mathsf{t}$,
- $\exists w F(w)^I = \mathsf{t}$ iff, for some $\xi \in |I|$, $F(\xi^*)^I = \mathsf{t}$.

As in propositional logic, we say that I *satisfies* F, and write $I \models F$, if $F^I = \mathsf{t}$.

We now show that interpretation (33) satisfies $\exists x Q(x, a)$. Let I_1 denote (33). By the definition, I_1 satisfies $\exists x Q(x, a)$ if $(\exists x Q(x, a))^{I_1} = \mathsf{t}$. This is the case iff for some $\xi \in |I_1|$, $(Q(\xi^*, a))^{I_1} = \mathsf{t}$. Let ξ be 9, then

$$(Q(9^*, a))^{I_1} = Q^{I_1}(9, a^{I_1}) = Q^{I_1}(9, 10) = \mathsf{t}.$$

Exercise 31. *Let $F(x)$ be formula (32). Show that, for every $n \in \omega$, (33) satisfies $F(n^*)$ iff $n < 2$.*

Satisfiability If there exists an interpretation satisfying a sentence F, we say that F is *satisfiable*. A set Γ of sentences is *satisfiable* if there exists an interpretation that satisfies all sentences in Γ.

Exercise 32. *Assume that P is a predicate constant of arity one. (a) Is set composed of sentences $P(a)$, $\exists x \neg P(x)$ satisfiable? (b) Is set composed of sentences $P(a)$, $\forall x \neg P(x)$ satisfiable?*

Entailment and Equivalence A set Γ of sentences *entails* a sentence F, or is a *logical consequence* of Γ (symbolically, $\Gamma \models F$), if every interpretation that satisfies all sentences in Γ satisfies F.

Exercise 33. *Determine whether the sentences*

$$\exists x P(x),\ \exists x Q(x)$$

entail

$$\exists x (P(x) \wedge Q(x)).$$

A sentence F is *logically valid* if every interpretation satisfies F. This concept is similar to the notion of a tautology.

The *universal closure* of a formula F is the sentence $\forall v_1 \cdots v_n F$, where $v_1, \dots, v_n$ are all free variables of F. About a formula with free variables we say that it is *logically valid* if its universal closure is logically valid. A formula F is *equivalent* to a formula G (symbolically, $F \equiv G$) if the formula $F \leftrightarrow G$ is logically valid.

Example 1. *Are formulas*

$$\forall x (P(x) \to Q(x))$$

and

$$\forall x P(x) \to \forall x Q(x)$$

equivalent.

Proof. We will illustrate that these formulas are not equivalent. In other words, we will illustrate that the formula

$$(\forall x(P(x) \to Q(x))) \leftrightarrow (\forall x P(x) \to \forall x Q(x)) \tag{34}$$

is not logically valid. To illustrate the later it is sufficient to find an interpretation that does not satisfy (34).

Let us define interpretation I as follows: its universe $|I|$ consists of two elements 1 and 2, functions P^I and Q^I are defined as follows:

element x of $\|I\|$	$P^I(x)$	$Q^I(x)$
1	t	f
2	f	f

It is easy to see that $(\forall x(P(x) \to Q(x)))^I = \mathsf{f}$. Indeed,

$$(P(1^*) \to Q(1^*))^I =\!\to (P^I(1), Q^I(1)) =\!\to (\mathsf{t}, \mathsf{f}) = \mathsf{f}.$$

On the other hand, $((\forall x P(x) \to \forall x Q(x)))^I = \mathsf{t}$. Indeed, $(\forall x P(x))^I = \mathsf{f}$, since $P(2^*)^I = P^I(2) = \mathsf{f}$. Similarly, $(\forall x P(x))^I = \mathsf{f}$. Thus, $((\forall x P(x) \to \forall x Q(x)))^I =\!\to ((\forall x P(x))^I, (\forall x Q(x))^I) =\!\to (\mathsf{f}, \mathsf{f}) = \mathsf{t}$.

It follows that

$$\begin{array}{l}((\forall x(P(x) \to Q(x))) \leftrightarrow (\forall x P(x) \to \forall x Q(x)))^I = \\ \leftrightarrow ((\forall x(P(x) \to Q(x)))^I, (\forall x P(x) \to \forall x Q(x))^I) = \\ \leftrightarrow (\mathsf{f}, \mathsf{t}) = \mathsf{f}.\end{array}$$

Consequently, I does not satisfy (34). □

Example 2. *Assume that A is a sentence and thus has no free variables. Does the equivalence*

$$(\forall x P(x)) \vee A \equiv \forall x(P(x) \vee A) \tag{35}$$

hold.

Proof. To show that (35) holds, we have to illustrate that formula

$$((\forall x P(x)) \vee A) \leftrightarrow (\forall x(P(x) \vee A)) \tag{36}$$

is logically valid. By definition, formula (36) is logically valid if every interpretation satisfies (36). Let I be any interpretation of a predicate signature of formulas $P(x)$ and A. We now illustrate that I satisfies (36), or in other words

$$(((\forall x P(x)) \vee A) \leftrightarrow (\forall x(P(x) \vee A)))^I = \mathsf{t}.$$

It is sufficient to illustrate that $(\forall x P(x)) \vee A)^I = (\forall x(P(x) \vee A))^I$.

Case 1. Assume that $(\forall x P(x))^I = \mathsf{t}$. Consequently, for every element ξ in $|I|$,

$$P(\xi^*)^I = \mathsf{t}. \tag{37}$$

Then, by the definition of function $\vee(x, y)$,

$$((\forall x P(x)) \vee A)^I = \mathsf{t}.$$

On the other hand, expression $(\forall x(P(x) \vee A))^I = \mathsf{t}$ if for every element ξ in $|I|$, $\vee(P(\xi^*)^I, A^I) = \mathsf{t}$. Indeed,for every element ξ in $|I|$, condition (37) holds thus $\vee(P(\xi^*)^I, A^I) = \vee(\mathsf{t}, \mathsf{A}^\mathsf{I}) = \mathsf{t}$.

Case 2. $(\forall x P(x))^I = \mathsf{f}$. Consequently,

$$((\forall x P(x)) \vee A)^I = \vee((\forall x P(x))^I, A^I) = \vee(\mathsf{f}, \mathsf{A}^\mathsf{I}) = \mathsf{A}^\mathsf{I}.$$

It also follows that there is an element ξ in $|I|$, such that

$$P(\xi^*)^I = \mathsf{f}. \tag{38}$$

On the other hand, $(\forall x(P(x) \vee A))^I = \mathsf{t}$ if for every element $\xi' \in |I|$, $(P(\xi'^*) \vee A)^I = \mathsf{t}$. By (38) and the definition of function $\vee(x, y)$ it follows that $(\forall x P(x) \vee A)^I = \mathsf{t}$ if and only if $A^I = \mathsf{t}$. Consequently, $(\forall x P(x) \vee A)^I = A^I$. □

Exercise 34. *For each of the formulas*

$$P(x) \to \exists x P(x),$$
$$P(x) \to \forall x P(x)$$

determine whether it is logically valid.

2.6 Elements of Mathematical Proofs

Exhaustive Proofs, Proofs by Cases, Trivial Proofs. In mathematics, theorems often have the form $\forall x(P(x) \to Q(x))$. Consider the following mathematical statement

For any integers x, if x is such that $-1 \leq x \leq 1$ then $x^3 = x$.

Intuitively, the implication

$$\forall x(-1 \leq x \leq 1 \to x^3 = x),$$

where x is an integer represents this statement in logical notation. This statement is easy to prove, because only three values of x satisfy its *antecedent* (left-hand-side of $\to$) $-1 \leq x \leq 1$, and we can check for each of them individually that it satisfies the *consequent* (right-hand-side of $\to$) $x^3 = x$ as well. For the values of integers that do not satisfy antecedent the proof is "trivial" or "vacuous". Indeed, implication always evaluates to true when its antecedent evaluates to false. This kind of reasoning is called "proof by cases or proof by exhaustion." We now present a proof for the statement above.

Proof. Consider any integer n.
Case 1: $n < -1$. Vacuous or Trivial proof. Indeed, condition $-1 \leq n \leq 1$ is false. Then, $(-1 \leq n \leq 1) \to n^3 = n$ evaluates to true.
Case 2: $n = -1$. $n^3 = -1 = n$.
Case 3: $n = 0$. $n^3 = 0 = n$.
Case 4: $n = 1$. $n^3 = 1 = n$.
Case 5: $n > 1$. Vacuous proof. □

Proof by exhaustion is not applicable if the antecedent is satisfied for infinitely many values of variables. For instance, the implication

$$\forall x(-1 \leq x \leq 1 \to -1 \leq x^3 \leq 1)$$

(x is a variable for real numbers) cannot be proved by exhaustion.

Direct and Indirect Proofs Recall that the integer n is *even* if there exists an integer k such that $n = 2k$. The integer n is *odd* if there exists an integer k such that $n = 2k + 1$. Every integer is either even or odd, and no integer is both even and odd. The real number r is *rational* if there exist integers p and q with $q \neq 0$ such that $r = p/q$. A real number that is not rational is called *irrational*.

A structure of a common direct proof of theorems of the form

$$\forall x(P(x) \to Q(x))$$

follows.

1. Consider any value v of x in the domain of discourse.
2. Assume property P holds about v.
3. Use valid argument forms or rules of inference to conclude that property Q holds about v.

We now use a direct prove to show that for any number x if x is an integer then x is a rational number (or, in other words, every integer is a rational number).

Proof. Consider any integer n. Obviously, $n = n/1$ where n and 1 are integers such that $1 \neq 0$. By the definition of a rational number, n is rational. □

Similarly, we will now show that if x is a rational number then $3x$ is also a rational number.

Proof. Consider any rational number n. By the definition of a rational number there exist integers p and q ($q \neq 0$) such that $n = p/q$. It follows that $3n = 3p/q$. Consequently, $3n$ is a rational number as both $3p$ and q are integers. □

Exercise 35. *Show that if x is a rational number then $3 + x$ is also a rational number.*

Exercise 36. *Show that if x is an even number then x^2 is also an even number.*

Exercise 37. *Show that if x is an odd number then x^3 is also an odd number.*

Exercise 38. *Show that the sum of two odd numbers is even.*

Proofs that do not begin with the premise/antecedent of theorem statement and end with the conclusion are called *indirect proofs*. Recall a valid argument form called contraposition (see Figure 4).

A proof by contraposition of $\forall x(P(X) \to Q(x))$ is as follows:

1. Consider any value v of x in the domain of discourse.
2. Assume property Q *does not hold* about v.
3. Use valid argument forms or rules of inference to conclude that property P *does not hold* about v.
4. By contraposition, $P(v) \to Q(v)$ holds.

We now use prove by contraposition to illustrate that *for any integer x if $3x + 2$ is odd, then x is odd.*

Proof. Consider any integer n. Assume that n is not odd. In other words, n is even. By the definition of an even integer there exists an integer k such that $n = 2k$. Thus,

$$3n + 2 = 6k + 2 = 2(3k + 1).$$

Consequently, $3n + 2$ is an even number also. By contraposition we illustrated that if $3n + 2$ is odd, then n is odd. □

Similarly we can show that *for any real number n if $3n$ is an irrational number then n is also an irrational number.*

Proof. Consider any real number n. Assume that n is a rational number. We illustrated earlier that under such assumption, $3n$ is also a rational number. By contraposition we derive that if $3n$ is irrational then n is also irrational. □

"Proof by Contradiction" is another common indirect proof technique. It often takes the following form: we assume that the statement of a theorem is false. Then we derive a contradiction, which illustrates that the statement of the theorem is true.

We now use prove by contradiction to illustrate that *for any integer x if $3x + 2$ is odd, then x is odd.*

Proof. Consider any integer n. By contradiction. Assume that statement "if $3n + 2$ is odd, then n is odd" does not hold. In other words, the statement "$3n + 2$ is not odd or n is odd" does not hold, or statement "$3n + 2$ is odd and n is not odd" holds. Thus, n is even. We illustrated earlier that from the fact that n is even it follows that $3n + 2$ is even. This contradicts our assumption that "$3n + 2$ is odd and n is not odd". □

Exercise 39. *Show that if x^2 is odd then x is odd.*

Exercise 40. *Show that if the sum of two numbers is odd then one of these numbers must be even.*

3 Growth of Functions

3.1 Calculus Notion of Function Growth Rate

Calculus Preliminaries L'Hopital's rule states:

$$\lim \frac{f(x)}{g(x)} = \lim \frac{f'(x)}{g'(x)}$$

Formulas for some derivatives follow:

$$(x^c)' = cx^{c-1}$$
$$(c^x)' = c^x ln(c)$$
$$(cf(x))' = cf'(x)$$
$$(f(x) + g(x))' = (f'(x) + g'(x))$$
$$(f(x)g(x))' = (f'(x)g(x) + g'(x)f(x))$$

where c is a constant, and x is a variable.

The number e is a mathematical constant that is the base of the natural log arithm ln. It is approximately equal to 2.71828, and is the $\lim_{n\to\infty}(1 + 1/n)^n$.

Growth Let $A_1, A_2, \ldots$ and $B_1, B_2, \ldots$ be two increasing sequences of positive numbers. We can decide which of them grows faster by looking at the limit of the ratio $\frac{A_n}{B_n}$ as n goes to infinity, if this limit exists. We say that

$$A \text{ grows faster than } B \text{ if} \qquad \lim \frac{A_n}{B_n} = \infty,$$
$$A \text{ and } B \text{ grow at the same rate if} \qquad 0 < \lim \frac{A_n}{B_n} < \infty,$$
$$B \text{ grows faster than } A \text{ if} \qquad \lim \frac{A_n}{B_n} = 0.$$

In the special case when

$$\lim \frac{A_n}{B_n} = 1$$

we say that A is *asymptotically equal* to B.

Examples:

$log\ _2 n$	$\sqrt[3]{n}$	$\sqrt{n}$	n	n^2	n^3	1.1^n	2^n	3^n	$n!$	n^n

slower faster

Let us illustrate that sequence $P_n = a_2n^2 + a_1n + a_0$, where a_i is a positive real number and $n \geq 1$, grows at the same rate as n^2. Indeed,

$$\lim \tfrac{P_n}{n^2} = \lim \tfrac{a_2n^2+a_1n+a_0}{n^2} =$$
$$\lim(a_2 + \tfrac{a_1}{n} + \tfrac{a_0}{n^2}) = a_2 + 0 + 0 = a_2$$

There is another way to illustrate the same fact by means of L'Hopital's rule. It is more complicated, but more general:

$$\lim \tfrac{P_n}{n^2} = \lim \tfrac{a_2n^2+a_1n+a_0}{n^2} =$$
$$\lim \tfrac{(a_2n^2+a_1n+a_0)'}{(n^2)'} =$$
$$\lim \tfrac{(a_2n^2+a_1n+a_0)''}{(n^2)''} =$$
$$\lim \tfrac{(2a_2n+a_1)'}{(2n)'} =$$
$$\lim \tfrac{2a_2}{2} = a_2.$$

Harmonic Numbers Recall that the harmonic numbers H_n are defined by the formula

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

The sequence H_n is asymptotically equal to $\ln n$. When n grows, the difference $H_n - \ln n$ approaches a finite limit, which is called *Euler's constant* and is denoted by γ. It is approximately equal to .577. The expression $\ln n + \gamma$ is a good approximation to H_n for large values of n.

Striling's Formula The sequence of factorials $n!$ grows approximately as fast as $\left(\frac{n}{e}\right)^n$. More precisely, $n!$ is asymptotically equal to

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

3.2 Big-O Notation

About two sequences A_n, B_n of positive numbers we say that A_n is $O(B_n)$ if there exist positive constants C, N such that

$$A_n \leq C \cdot B_n \text{ whenever } n \geq N.$$

For instance, we can prove "by induction" (the proof method we will learn shortly) that

$$2^n \leq n! \text{ whenever } n \geq 4;$$

it follows that 2^n is $O(n!)$, because we can take $C = 1$ and $N = 4$ in the definition of big-O notation.

An example of a proof that a function is in big-O relation with another function The statement $n^2 > n$ holds, whenever $n \geq 2$. It follows that n is $O(n^2)$. It also follows that sequence $P_n = a_2n^2 + a_1n + a_0$, where a_i is a positive real number, is $O(n^2)$. Indeed,

$$\begin{aligned} &a_2n^2 + a_1n + a_0 \leq \\ &a_2n^2 + a_1n^2 + a_0n^2 = \\ &(a_2 + a_1 + a_0)n^2 \end{aligned}$$

Thus we can take $C = a_2 + a_1 + a_0$ and $N = 2$. It is trivial to see that n^2 is $O(P_n)$. Indeed, take $C = 1$ and $N = 1$.

An example of a proof that a function is not in big-O relation with another function We now show that n^2 is not $O(n)$, $(n \geq 1)$. Proof by contradiction. Assume that there are positive constants C and N such that

$$n^2 \leq C \cdot n \text{ whenever } n \geq N.$$

Then the following holds

$$n \leq C \text{ whenever } n \geq N.$$

Take n to be $C + N$.

About two sequences A_n, B_n of positive numbers we say that A_n is $\Theta(B_n)$, or A_n is *the same order as* B_n, if A_n is $O(B_n)$ and B_n is $O(A_n)$.

For instance, n^2 is $\Theta(P_n)$.

We now relate the concept of "growth" and Big-O Notation.

Theorem 1. *For two sequences A_n, B_n of positive numbers, when $\lim \frac{B_n}{A_n}$ exists, if B grows faster than A then A is $O(B)$.*

Theorem 2. *For two sequences A_n, B_n of positive numbers, when $\lim \frac{A_n}{B_n}$ exists, A and B grow at the same rate if and only if A is $\Theta(B)$.*

Exercise 41. *Use Theorem 1 to illustrate that $\frac{n^2}{n+1}$ is $O(n^2)$, $(n \geq 1)$.*

Exercise 42. *Use Theorem 2 to illustrate that* $\frac{n^2}{n+1}$ *is* $\Theta(n+2)$, $(n \geq 1)$.

Exercise 43. *Use Theorem 2 to illustrate that* $n + log\ n$ *is* $\Theta(n)$, $(n \geq 1)$.

These theorems provide us with additional means to illustrate whether two sequences are in big-O or big-Θ relation.

In this section we considered increasing sequences of positive numbers. The definitions we presented can be generalized to broader class of functions.

For instance, for functions $f(x)$ and $g(x)$ from real numbers to real numbers that are positive for x sufficiently large, we say that

$$f(x) \text{ grows faster than } g(x) \text{ if} \qquad \lim \frac{f(x)}{g(x)} = \infty,$$

$$f(x) \text{ and } g(x) \text{ grow at the same rate if} \qquad \lim \tfrac{f(x)}{g(x)} = L \neq 0,$$
$$\text{where } L \text{ is some finite number,}$$

$$g(x) \text{ grows faster than } f(x) \text{ if} \qquad \lim \frac{f(x)}{g(x)} = 0. \ .$$

For functions f and g from the set of integers or the set of real numbers to the set of real numbers, we say that $f(x)$ is $O(g(x))$ if there are positive constants C and N such that

$$|f(x)| \geq C|g(x)|$$

whenever $x > N$.

Theorem 3. *Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, *where* $a_0, a_1, \ldots, a_{n-1}, a_n$ *are real numbers with* $a_n \neq 0$.

$$f(x) = \Theta(x^n)$$

In other words, $f(x)$ *is the same order as* x^n.

Terminology

If a function of n is	we say that it is
$O(1)$	constant
$O(log\ \ n)$	log arithmic
$O(n)$	linear
$O(n\ log\ \ n)$	n log n
$O(n^2)$	quadratic
$O(n^c)$ where c is a constant	polynomial
$O(c^n)$ where c is a constant greater than 1	exponential
$O(n!)$	factorial

Note how earlier we illustrated that function n^2 $(n \geq 1)$ is not $O(n)$. It is the same as saying that function n^2 is not linear.

Exercise 44. *Let sequence* A_n *be* $3n^3 + (log\ n)^4$, $(n \geq 1)$. *Find the least integer* x *so that sequence* B_n *of the form* n^x $(n \geq 1)$ *is such that* A *is* $O(B)$. *Prove your claim.*

Proof. The answer is 3.

One part of the argument: We illustrate that A and B grow at the same rate. From this fact by Theorem 2 it follows that A is $\Theta(B)$. The definition of big-Θ assumes that A is $O(B)$. We start by computing

$$\lim \frac{3n^3+(log\ n)^4}{n^3} = \lim \frac{(3n^3+(log\ n)^4)'}{(n^3)'} = \lim \frac{9n^2+\frac{4(log\ n)^3}{n}}{3n^2} =$$
$$\lim \frac{(\frac{9n^3+4(log\ n)^3}{n})}{3n^2} = \lim \frac{9n^3+4(log\ n)^3}{3n^3} = \lim \frac{(9n^3+4(log\ n)^3)'}{(3n^3)'} =$$
$$\lim \frac{(27n^3+12(log\ n)^2)}{9n^3} = \lim \frac{(27n^3+12(log\ n)^2)'}{(9n^3)'} = \lim \frac{(81n^3+24(log\ n))}{27n^3} =$$
$$\lim \frac{(81n^3+24(log\ n))'}{(27n^3)'} = \lim \frac{(243n^3+24)}{81n^3} = \lim (\frac{243n^3}{81n^3} + \frac{24}{81n^3}) = 3$$

By definition, A and B grow at the same rate.

Another part of the argument: : we show that A is not $O(n^2)$ (or in other word it is not a quadratic function).

By contradiction. Assume that $3n^3 + (log\ n)^4$ is $O(n^2)$. Then, by the definition of big-O relation there are positive constants C and N such that

$$3n^3 + (log\ n)^4 \leq C \cdot n \text{ whenever } \quad n \geq N.$$

Note how

$$n^3 \leq 3n^3 \leq 3n^3 + (log\ n)^4.$$

Combining the last two claims we derive that

$$n^3 \leq C \cdot n \text{ whenever } \quad n \geq N.$$

Consequently,

$$n^2 \leq C \text{ whenever } \quad n \geq N.$$

Take n to be $C + N$; this value contradicts the last statement. Thus we derived a contradiction. □

4 Elements on Algorithms and their Complexity

4.1 Algorithms and their Properties

An *algorithm* is a finite sequence of precise instructions for performing a computation or for solving a problem. There are several properties that algorithms generally share:

- problem – a computational problem in question, i.e., a problem for which this algorithm is designed to find solutions
- input – an algorithm has input values from a specified set
- output – from each set of input values an algorithm produces output values from a specified set that correspond to solutions of the problem in question
- definiteness – steps of the algorithm must be defined precisely
- correctness – it needs to to what it claims
- finiteness – an algorithm should produce the desired output after a finite number of steps for any input
- effectiveness – it must be possible to perform each step of an algorithm exactly and in finite amount of time
- generality – the procedure should be applicable to all problems of the desired form, not just for a particular set of nput values.

Pseudocode[2] is often used to describe algorithms for humans. Programming languages are used to specify algorithms for machine reading.

A Toy Programming Language In the programming language TPL that we'll be using, all variables are integer variables, so that there is no need to declare them. *Integer expressions* are formed from integer variables and integer constants using the operators $+$, $-$, $\times$ and *div*, and finally ( ) which specifies operator precedence. *Boolean expressions* are formed from equalities and inequalities between integer expressions using the propositional connectives $\wedge$, $\vee$, $\neg$. *Assignments* are expressions of the form $v \leftarrow e$ where v is an integer variable and e is an integer expression. *Statements* are formed from assignments and the statement **skip** using three constructs:

- sequential composition
$$P_1 \; ; \ldots \; ; P_n$$
where $P_1, \ldots, P_n$ are statements,
- conditional composition
$$\textbf{if } B \textbf{ then } P_1 \textbf{ else } P_2 \textbf{ endif}$$
where B is a Boolean expression and P_1, P_2 are statements,
- loop
$$\textbf{while } B \textbf{ do } P \textbf{ enddo}$$
where B is a Boolean expression and P is a statement.

[2]Pseudocode is an informal high-level description of the operating principle of an algorithm that uses the structural conventions of a programming language, but is intended for human reading rather than machine reading.

Algorithms for Computing Triangular Numbers in TPL Recall that the triangular number T_n is the sum of all integers from 1 to n:

$$T_n = \sum_{i=1}^{n} i = 1 + 2 + \cdots + n.$$

The following algorithm in TPL computes the triangular number T_n

$$\begin{array}{l} T \leftarrow 0; \\ i \leftarrow 1; \\ \textbf{while } \ i \leq n \ \textbf{ do} \\ \quad T \leftarrow T + i; \\ \quad i \leftarrow i + 1; \\ \textbf{enddo} \end{array} \tag{39}$$

Later in the course we will prove this claim formally. Integer variable n is intuitively serves the role of an input for this algorithm whereas T is output that stand for the triangular number T_n.

Also an algorithm below

$$T \leftarrow (n \times (n + 1)) \ div \ 2 \tag{40}$$

computes the triangular number T_n. (Recall that $T_n = \frac{n(n+1)}{2}$. It is left as an exercise to prove that $\frac{n(n+1)}{2} = (n \times (n + 1)) \ div \ 2$ when n is a nonnegative integer.)

4.2 Time Complexity of Algorithms

We will now study one possible metric of evaluating the effectiveness of an algorithm. This metric relies on evaluating the number of operations used by the algorithm when the input has a particular size. In case of the TPL the operations include, for example, assignment $\leftarrow$ and addition $+$. Thus, when evaluating an effectiveness of an algorithm we are first interested in finding a function from the size of an input to a number of operations required by an algorithm to compute output for this input. Sometimes only some operation are considered. It is obvious that number of operations required by an algorithm correlates with the time spent by a computer executing the algorithm.

We say that function f (from a nonnegative integer to a nonnegative integer) *captures* algorithm A with respect to set S of operations if given an integer that represents the size of an input of A the function returns an integer that stands for the total number of operations from S used during the execution of A. We omit "with respect to set S of operations" if set S comprises all operations occurring in A.

For instance, let A_1 denote algorithm (39). The size of the input of A_1 is determined by the value of its input variable n. Function $f_1(n) = n + 1$ captures A_1 w.r.t. set $\{\leq\}$ of operations. Indeed, the comparison operation $\leq$ is used in a boolean expression of a while-loop in (39). This comparison will be performed $n+1$ times given the usual semantics of a while-loop. Function $f_2(n) = f_1(n) + 4n + 2 = 5n + 3$ captures A_1 w.r.t. $\{\leq, +, \leftarrow\}$. In other words, $f_2(n)$ captures A_1. Indeed, $f_1(n)$ stands for the number of comparisons performed by the algorithm. The while-loop of (39) executes n times. Thus, the statements in the scope of this loop

$$\begin{array}{l} T \leftarrow T + i; \\ i \leftarrow i + 1; \end{array}$$

will be performed n times. These statements comprise 4 operations occurring in $\{\leq, +, \leftarrow\}$, two additions and two assignments. This results in $4n$ operations due to addition and assignment within the while loop. The first two lines of (39) contribute two more operations.

For a function f from a nonnegative integer to a nonnegative integer, and a set S of operations we say that an algorithm A has *time complexity* $\Theta(f)$ measured with respect to S when there is a function g such that

- g captures A with respect to S,

- g is $\Theta(f)$.

We omit "measured with respect to S" if set S comprises all operations occurring in A.

It is easy to see that $f_1(n)$ and $f_2(n)$ are both $\Theta(n)$. Thus it follows that algorithm (39) has time complexity $\Theta(n)$ measured with respect to set $\{\leq\}$ of operations as well as with respect to $\{\leq, +, \leftarrow\}$. Or in other words, algorithm (39) has time complexity $\Theta(n)$.

Terminology

If an algorithm is captured by a function of n (w.r.t. set S of operations) that is	we say that its (time) complexity (measured w.r.t. S) is
$\Theta(1)$	constant
$\Theta(log\ n)$	logarithmic
$\Theta(n)$	linear
$\Theta(n\ log\ n)$	n log n
$\Theta(n^2)$	quadratic
$\Theta(n^c)$ where c is a constant	polynomial
$\Theta(c^n)$ where c is a constant greater than 1	exponential
$\Theta(n!)$	factorial

For instance, algorithm (39) is linear w.r.t. all operations used in this algorithm (or any set composed of some operations occurring in the algorithm). It is easy to see that algorithm (40) has complexity $\Theta(1)$ (or, in other words, its complexity is constant) w.r.t. all operations used in this algorithm.

5 Induction and Recursive Definitions

5.1 Proofs by Induction

Induction is a useful proof method in mathematics and computer science. When we want to prove by induction that some statement containing a variable n is true for all nonnegative values of n, we do two things. First we prove the statement when $n = 0$; this part of the proof is called the *basis*. Then we prove the statement for $n + 1$ assuming that it is true for n; this part of the proof is called the *induction step.* (The assumption that the statement is true for n, which is used in the induction step, is called the *induction hypothesis*.)

Once we have completed both the basis and the induction step, we can conclude that the statement holds for all nonnegative values of n. Indeed, according to the basis, it holds for $n = 0$. From this fact, according to the induction step, we can conclude that it holds for $n = 1$. From this fact, according to the induction step, we can conclude that it holds for $n = 2$. And so on.

As an example, we will give yet another proof of the formula for triangular numbers.

Problem. Prove that for all nonnegative integers n

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Solution. *Basis.* When $n = 0$, the formula turns into

$$0 = \frac{0(0+1)}{2},$$

which is correct. *Induction step.* Assume that

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

We need to derive from this assumption that

$$1 + 2 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2}.$$

Using the induction hypothesis, we calculate:

$$\begin{aligned} 1 + 2 + \cdots + n + (n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{n(n+1) + 2(n+1)}{2} \\ &= \frac{(n+1)(n+2)}{2}. \end{aligned}$$

Proofs by induction can be symbolically represented by this inference rule:

$$\frac{P(0) \qquad \forall n(P(n) \to P(n+1))}{\forall n P(n)}.$$

Here n is a variable for nonnegative integers, and the expression $P(n)$ means that n has the property that we want to prove holds for n. The first premise represents the basis, and the second premise represents the induction step.

We will give now two more examples of the use of induction.

Problem. Prove that for all nonnegative integers n, $2^n > n$.

Solution. *Basis.* When $n = 0$, the formula turns into $1 > 0$, which is correct. *Induction step.* Assume that $2^n > n$ for some $n > 0$. We need to derive from this assumption that $2^{n+1} > n + 1$. This can be done as follows, using the induction hypothesis and then the fact that $2^n \geq 2 > 1$ for $n > 0$:

$$2^{n+1} = 2^n + 2^n > n + 2^n > n + 1.$$

Problem. Prove that for all nonnegative integers n, $n^3 - n$ is a multiple of 3.

Solution. *Basis.* When $n = 0$, we need to check that $0^3 - 0$ is a multiple of 3, which is correct. *Induction step.* Assume that $n^3 - n$ is a multiple of 3. We need to derive from this assumption that $(n+1)^3 - (n+1)$ is a multiple of 3. This expression can be rewritten as follows:

$$\begin{aligned}(n+1)^3 - (n+1) = n^3 + 3n^2 + 3n + 1 - (n+1) &= n^3 + 3n^2 + 2n \\ &= (n^3 - n) + 3n^2 + 3n.\end{aligned}$$

Consider the three summands $n^3 - n$, $3n^2$, $3n$. By the induction hypothesis, the first of them is a multiple of 3. It is clear that the other two are multiples of 3 also. Consequently, the sum is a multiple of 3.

We have used induction to prove statements about *nonnegative* integers. Statements about *positive* integers can be proved by induction in a similar way, except that the basis corresponds to $n = 1$; also, in the induction step we may assume that $n \geq 1$. Similarly, if we want to prove a statement about all integers beginning with 2 then the basis corresponds to $n = 2$, and so on.

Problem. Prove that for all integers n such that $n \geq 10$, $2^n > n + 1000$.

Solution. *Basis.* When $n = 10$, the formula turns into $1024 > 1010$, which is correct. *Induction step.* Assume that $2^n > n + 1000$ for an integer n such that $n \geq 10$. We need to derive that $2^{n+1} > n + 1001$. This can be done as follows:

$$2^{n+1} = 2^n + 2^n > n + 1000 + 2^n \geq n + 1000 + 2^{10} > n + 1001.$$

Where does Induction come from? Consider Peano's Axioms[3] – axioms for the natural numbers:

1. Zero – 0 – is a number.
2. If n is a number, the successor of n, denoted by n', is a number.
3. Zero is not the successor of a number.
4. Two numbers of which the successors are equal are themselves equal.
5. (induction axiom.) If zero has the property P [Basis], and if the successor of every number with this property P has this property P also [Induction Step], then all numbers have this property.

Induction axiom is the basis for Proofs by Induction.
Let us prove some "trivial" property about natural numbers. First we define addition:

- m+0=m,
- m+n'=(m+n)'.

where m and n are numbers.

Problem. Use Peano axioms to prove that $0 + n = n$.

Solution. *Basis.* By the definition of addition $m + 0 = m$. Thus $0 + 0 = 0$. In other words property in question holds about 0. *Induction step.* Assume that $0 + n = n$ holds for n. We now show that under this assumption also the following claim holds: $0 + n' = n'$. Indeed, $0 + n' = (0 + n)'$ by the second clause in the definition of addition. In turn, $(0 + n)' = n'$ by the inductive assumption. In other words we just illustrated that if property in question holds about n then it also holds for its successor n'. By induction axiom we derive that all numbers have this property that $0 + n = n$.

[3] `http://mathworld.wolfram.com/PeanosAxioms.html`

More on Induction The sum S_n of the squares of numbers from 1 to n, i.e., $S_n = 1^2 + \cdots + n^2$: can be written also as

$$S_n = \frac{(2n+1)(n+1)n}{6}. \tag{41}$$

Exercise 45. *Prove the formula* (41) *for* S_n *by induction.*

Exercise 46. *Prove the formula for the sum* C_n *of the cubes of numbers from 1 to* n*:*

$$C_n = \frac{n^2(n+1)^2}{4}.$$

Exercise 47. *(a) Calculate the values of the expression*

$$\frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \cdots + \frac{1}{n\cdot(n+1)}$$

for $n = 1, \ldots, 4$*, and guess what a simple formula for those values may look like. (b) Use your conjecture to predict the value of this expression for* $n = 999$*. (c) Prove your conjecture by induction.*

Exercise 48. *Prove that for all nonnegative integers* n*,* $4^n - 1$ *is a multiple of 3.*

Exercise 49. *Guess which nonegative values of* n *satisfy the inequality*

$$2^n > n + 5.$$

Prove that your answer is correct.

5.2 Strong and Structural Induction

The following modification of the induction method is called *strong induction*. We want to prove some statement for all nonnegative values of n. To this end, we assume that the statement is true for all numbers that are less than n. From this *induction hypothesis* we need to derive that the statement is true for n; this is the *induction step* of strong induction.

Here is a proof by strong induction.

Recall that the sequence of *Fibonacci numbers* $F_0, F_1, F_2, \ldots$ is defined by the equations

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_{n+2} &= F_n + F_{n+1}. \end{aligned}$$

Problem. Let F be the sequence of Fibonacci numbers. Prove that, for all nonnegative integers n, $F_n < 2^n$.

Solution. Assume that, for every $k < n$, $F_k < 2^k$. We need to prove that $F_n < 2^n$.
Case 1: $n = 0$. Then the inequality to be proved turns into $0 < 1$.
Case 2: $n = 1$. Then the inequality to be proved turns into $1 < 2$.
Case 3: $n > 1$. Then

$$\begin{aligned} F_n = F_{n-2} + F_{n-1} &< 2^{n-2} + 2^{n-1} \\ &< 2^{n-2} + 2\cdot 2^{n-2} \\ &< 3\cdot 2^{n-2} < 4\cdot 2^{n-2} \\ &2^n. \end{aligned}$$

Exercise 50. *Let F be the sequence of Fibonacci numbers. Prove that, for all nonnegative integers n, $F_n < (\frac{5}{3})^n$.*

Proof by strong induction. Assume that, for every $k < n$, $F_k < (\frac{5}{3})^k$. We need to prove that $F_n < (\frac{5}{3})^n$.
Case 1: $n = 0$. Then the inequality to be proved turns into $0 < 1$.
Case 2: $n = 1$. Then the inequality to be proved turns into $1 < \frac{5}{3}$.
Case 3: $n > 1$. Then

$$\begin{aligned} F_n = F_{n-2} + F_{n-1} &< (\tfrac{5}{3})^{n-2} + (\tfrac{5}{3})^{n-1} \\ &< (\tfrac{5}{3})^{n-2} + (\tfrac{5}{3})^{n-2} \cdot (\tfrac{5}{3}) \\ &< (\tfrac{8}{3})(\tfrac{5}{3})^{n-2} < (\tfrac{25}{9}) \cdot (\tfrac{5}{3})^{n-2} \\ &< (\tfrac{5}{3})^2 \cdot (\tfrac{5}{3})^{n-2} \\ &< (\tfrac{5}{3})^n \end{aligned}$$

Recall the recursive definition of a (propositional) formula:

- every atom is a formula,
- both 0-place ($\top$ and $\bot$) connectives are formulas,
- if F is a formula then $\neg F$ is a formula,
- for any binary connective $\odot$, if F and G are formulas then $(F \odot G)$ is a formula.

Properties of formulas can be often proved by induction. For example, by strong induction on length (number of symbols) of a formula. In such a proof, the induction hypothesis is that every formula which is shorter than F has the property P that we want to prove. From this assumption we need to derive that F has property P also. Then it follows that all formulas have property P.

In another useful form of induction, we check that all atoms and 0-place connectives have property P, and that the property is preserved when a new formula is formed using a unary or binary connective. More precisely, we show that

- every atom has property P,
- both 0-place connectives have property P,
- if a formula F has property P then so does $\neg F$,
- for any binary connective $\odot$, if formulas F and G have property P then so does $(F \odot G)$.

Then we can conclude that property P holds for all formulas. This is called "structural induction."

Problem. Prove that the number of left parentheses in any formula is equal to the number of right parentheses.

Solution 1. By structural induction. We call the property *"the number of left parentheses is equal to the number of right parentheses"* – P. Consider any formula F.

- Every atom trivially has property P. (Indeed, there are no parentheses.)
- Both 0-place connectives trivially have property P.
- Let a formula F have property P. Then $\neg F$ also has this property. It is easy to see that formula $\neg F$ and F share the same parentheses. Hence property P holds about F.

- Consider any binary connective $\odot$. Let formulas G and H have property P. We now show that formula $F = (G \odot H)$ also has this property. Let l_F and r_F denote numbers of left and right parentheses of F respectively. Similarly, letters l_G, r_G, l_H, and r_H denote numbers of left and right parentheses of G and H. We assumed that $l_G = r_G$ and $l_H = r_H$. From the construction of F (i.e., the fact that F has the form $(G \odot H)$), it follows that $l_F = l_G + l_H + 1$. Similarly, $r_F = r_G + r_H + 1$. Consequently, $l_F = r_F$.

Solution 2. By strong induction. We call the property *"the number of left parentheses is equal to the number of right parentheses"* – P. Consider any formula F. Assume that, for every formula that is shorter than F the number of left parentheses is equal to the number of right parentheses. We need to prove that this property P holds for F. From the definition of a formula there are 4 cases or in other words formula F can take 4 different forms. We now show that for every possible form of F the property P holds.
Case 1. F is an atom. P trivially holds.
Case 2. F is some 0-place connective. P trivially holds.
Case 3. F has the form $\neg F'$. F' is the shorter formula than F. Hence P holds about F'. It is easy to see that formula $\neg F'$ and F' share the same parentheses. Hence property P holds about F.
Case 4. F has the form $(G \odot H)$, where G and H are formulas which are trivially of shorter length than F. Let l_F and r_F denote numbers of left and right parentheses of F respectively. Similarly, letters l_G, r_G, l_H, and r_H denote numbers of left and right parentheses of G and H. From the inductive hypothesis it follows that $l_G = r_G$ and $l_H = r_H$. From the construction of F, it follows that $l_F = l_G + l_H + 1$. Similarly, $r_F = r_G + r_H + 1$. Consequently, $l_F = r_F$.

Exercise 51. *A* prefix *of a string $a_1 \cdots a_n$ is any string of the form $a_1 \cdots a_m$ where $0 \leq m \leq n$. In any prefix of a formula, the number of left parentheses is greater than or equal to the number of right parentheses.*

We call the property *"In any prefix of a formula, the number of left parentheses is greater than or equal to the number of right parentheses"* – P. We call the property *"The number of left parentheses is greater than or equal to the number of right parentheses"* – P'. Note that to illustrate that P holds for formula F we have to show that property P' holds for every prefix of F. On the other hand, if we know that property P holds for formula F, we then know that property P' holds for every prefix of F. By structural induction.

- For an atom a, it has only two prefixes: empty string and atom a itself. The property P' trivially holds for both of these prefixes. Thus property P holds for atom a.
- Case of 0-place connectives is similar to that of an atom.
- Let a formula F have property P. Consider formula $\neg F$. Take any prefix p of $\neg F$. It is easy to see that it is either an empty string or has the form $\neg p'$ where p' is a prefix of F. The empty string case is trivial. For the later case, we know that the number of left parentheses is greater than or equal to the number of right parentheses in p'. Hence the same holds about $\neg p'$.
- Consider any binary connective $\odot$. Let formulas G and H have property P. We now show that formula $F = (G \odot H)$ also has this property P by illustrating that property P' holds about every possible prefix p of F.
  Case 1. Prefix p of F is an empty string. Trivial.
  Case 2. Prefix p of F is (p' where p' is some prefix of G. We know that the number of left parentheses is greater than or equal to the number of right parentheses in p'. The number of left parentheses in (p' is greater by one than the number of left parentheses in p'. Hence the property P' holds of p.
  Case 3. Prefix p of F is ($G \odot p'$ where p' is some prefix of H. We know that (i) the number of left parentheses is greater than or equal to the number of right parentheses in G, (ii) the number of left parentheses is greater than or equal to the number of right parentheses in p'. The number of left parentheses in ($G \odot p'$ is greater by one than the number of left parentheses in G plus the number of left parentheses in p'. The number of right parentheses ($G \odot p'$ is the same as the sum of right parentheses in G and p'. It follows that property P' holds of p.
  Case 4. Prefix p of F is F itself. Hence p has the form $(G \odot H)$. Let l_F and r_F denote numbers of left and right parentheses of F respectively. Similarly, letters l_G, r_G, l_H, and r_H denote numbers of left

and right parentheses of G and H. We know that (i) the number of left parentheses is greater than or equal to the number of right parentheses in G, (ii) the number of left parentheses is greater than or equal to the number of right parentheses in H. Or, in other words, $l_G \leq r_G$ and $l_H \leq r_H$. By the construction of F, $l_F = l_G + l_H + 1$ and $r_F = r_G + r_H + 1$. Consequently, $l_F \leq r_F$.

6 Recursive Definitions

A *recursive definition* of a sequence of numbers expresses some members of that sequence in terms of its other members. For instance, here is a recursive definition of triangular numbers:

$$\begin{aligned} &T_0 = 0, \\ &T_{n+1} = T_n + n + 1 \end{aligned}$$

(n is a variable for nonnegative integers.) The first formula gives the first triangular number explicitly; the second formula shows how to calculate any other triangular number if we already know the previous triangular number.

There are two ways to find T_4 using this definition. One is to find first T_1, then T_2, then T_3, and then T_4:

$$\begin{aligned} T_1 &= T_0 + 1 = 0 + 1 = 1, \\ T_2 &= T_1 + 2 = 1 + 2 = 3, \\ T_3 &= T_2 + 3 = 3 + 3 = 6, \\ T_4 &= T_3 + 4 = 6 + 4 = 10. \end{aligned}$$

The other possibility is to form a chain of equalities that begins with T_4 and ends with a number:

$$\begin{aligned} T_4 &= T_3 + 4 \\ &= T_2 + 3 + 4 = T_2 + 7 \\ &= T_1 + 2 + 7 = T_1 + 9 \\ &= T_0 + 1 + 9 = T_0 + 10 \\ &= 0 + 10 = 10. \end{aligned}$$

This is an example of "lazy evaluation": we don't calculate the members of the sequence other than our goal T_4 until they are needed. The strategy used in the first calculation is "eager," or "strict."

Consider the recursive definition of factorials

$$\begin{aligned} &0! = 1, \\ &(n+1)! = n! \cdot (n+1). \end{aligned} \tag{42}$$

Exercise 52. *Calculate 4! using (a) eager evaluation, and (b) lazy evaluation.*

Exercise 53. *Function f is defined by the formulas*

$$\begin{aligned} &f(0) = 10, \\ &f(n+1) = f(n)(n^2 + n - 90) + 1. \end{aligned}$$

Find $f(11)$ without a calculator.

If a sequence of numbers is defined using Sigma-notation then we can always rewrite its definition using recursion. For instance, the formula

$$X_n = \sum_{i=1}^{n} \frac{1}{i^2 + 1}$$

can be rewritten as

$$\begin{aligned} &X_0 = 0, \\ &X_{n+1} = X_n + \frac{1}{(n+1)^2 + 1}. \end{aligned}$$

The definition of factorials

$$n! = \prod_{i=1}^{n} i$$

can be rewritten as (42).

Exercise 54. *The numbers $U_0, U_1, U_2, \ldots$ are defined by the formula*

$$U_n = \sum_{i=1}^{n} (2i-1)^2.$$

Define this sequence using recursion, instead of sigma-notation.

To prove properties of recursively defined sequences, we often use induction. Consider, for instance, the numbers $Y_0, Y_1, Y_2, \ldots$ defined by the formulas

$$\begin{aligned} Y_0 &= 0, \\ Y_{n+1} &= 2Y_n + n + 1. \end{aligned}$$

We will prove by induction that $Y_n \geq 2^n$ whenever $n \geq 2$. *Basis:* $n = 2$. Since $Y_2 = 4$ and $2^2 = 4$, the inequality $Y_2 \geq 2^2$ holds. *Induction step.* Assume that $Y_n \geq 2^n$ for an integer n such that $n \geq 2$. We need to derive that $Y_{n+1} \geq 2^{n+1}$. This can be done as follows:

$$Y_{n+1} = 2Y_n + n + 1 \geq 2 \cdot 2^n + n + 1 = 2^{n+1} + n + 1 > 2^{n+1}.$$

Recall one of the axioms about integers:

> (i) *The set of integers Z is closed under the operations of addition and multiplication, that is, the sum and product of any two integers is an integer.*

We will now prove by induction that Y_n is an integer for $n \geq 0$ (recall that n is an integer). *Basis:* $n = 0$. Since $Y_0 = 0$, it immediately follows that Y_0 is an integer. *Induction step.* Assume that Y_n is an integer for some $n > 0$. We need to derive that Y_{n+1} is also an integer. Per the definition of this sequence, $Y_{n+1} = 2Y_n + n + 1$. Since the set of integers is closed under the operations of addition and multiplication and we are given that Y_n and n are integers it follows that Y_{n+1} is also an integer.

Exercise 55. *Is it true that for every even n, Y_n is even? Prove that your answer is correct. Is induction required in the proof?*

Recursive definitions can be written in "case notation" by showing which formula should be used for calculating the n-th member of the sequence depending on the value of n. For instance, the definition of triangular numbers, rewritten in case notation, will look like this:

$$T_n = \begin{cases} 0, & \text{if } n = 0, \\ T_{n-1} + n, & \text{otherwise.} \end{cases}$$

The sequence of *Fibonacci numbers* $F_0, F_1, F_2, \ldots$ is defined by the equations

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_{n+2} &= F_n + F_{n+1}. \end{aligned}$$

It is different from the examples that we've seen before in that two members of the sequence are given explicitly, not one; but to calculate any other Fibonacci number we need to know two previous Fibonacci numbers, one is not enough. Here is the definition of Fibonacci numbers in case notation:

$$F_n = \begin{cases} 0, & \text{if } n = 0, \\ 1, & \text{if } n = 1, \\ F_{n-2} + F_{n-1}, & \text{if } n \geq 2. \end{cases}$$

Exercise 56. *Rewrite the recursive definition of the factorial function in the case format.*

In all examples of recursive definitions so far, the larger n is, the more work is needed to calculate the n-th member of the sequence. The recursive definition of the numbers $M(0), M(1), M(2), \ldots$ shown below is different: it's easy to calculate $M(n)$ when n is large, and difficult when n is small.

$$M(n) = \begin{cases} n - 10, & \text{if } n > 100, \\ M(M(n+11)), & \text{otherwise.} \end{cases}$$

Recursive Definitions with Two Variables A *function of two variables* is a function whose domain consists of ordered pairs (x, y). For example, multiplication is a function of two variables: $f(x, y) = x \cdot y$.

Some functions of two variables can be defined using recursion. For example, the multiplication of nonnegative integers can be defined in terms of addition as follows:

$$f(m, 0) = 0,$$
$$f(m, n + 1) = f(m, n) + m.$$

Exercise 57. *Use the definition above to find* $f(2, 3)$.

Exercise 58. *Give a recursive definition for the function* $f(m, n) = m^n$ *using no algebraic operations other than addition and multiplication. Use it to find* $f(2, 3)$.

Exercise 59. *Consider the function* f *defined by*

$$f(m, 0) = 1,$$
$$f(0, n + 1) = 1,$$
$$f(m + 1, n + 1) = f(m, n) + f(m, n + 1).$$

Make the table of values of $f(m, n)$ *for all* $m, n \leq 5$.

7 Counting

7.1 The Product Rule

Theorem 4. The Product Rule. *Suppose a sequence $T_1, \ldots, T_n$ of tasks so that t_i $(1 \leq i \leq n)$ stands for the number of possible ways to perform task T_i, regardless in which ways the other tasks in the sequence are done. The number of possible ways to perform the sequence $T_1, \ldots, T_n$ of tasks is $t_1 \times \cdots \times t_n$.*

The product rule is often phrased in terms of sets as follows: If $A_1, \ldots A_m$ are finite sets, then the number of elements in the Cartesian product of these sets is the product of the number of the elements in each of the sets:

$$|A_1 \times A_2 \times \cdots \times A_m| = |A_1| \times |A_2| \times \cdots \times |A_m|$$

Problem. A local telephone number is given by a sequence of six digits. How many different telephone numbers are there if the first digit cannot be 0?

Solution.

$$9 \times 10 \times 10 \times 10 \times 10 \times 10 = 900,000$$

Problem. A local telephone number is given by a sequence of six digits. How many different telephone numbers are there if the first digit cannot be 0; and the situation where all of the remaining 5 digits are 0 is forbidden.

Solution. The number of different telephone numbers whose first digit is not 0 follows: $900,000$. The situation where the first digit is non 0 and all of the remaining 5 digits are 0 accounts for $9 \times 1 \times 1 \times 1 \times 1 \times 1 = 9$ by product rule. So the answer is $900,000 - 9 = 899,991$.

Problem. A new company with just two employees, Sanchez and Patel, rents a floor of a building with 12 offices. How many ways are there to assign different offices to these two employees?

Solution.

$$12 \times 11 = 132$$

Problem. The chairs in an auditorium are to be labeled with a letter and a positive integer not exceeding 100. What is the largest number of chairs that can be labeled differently?

Solution.

$$26 \times 100 = 2600$$

Problem. How many different license plates are available if each plate contains a sequence of three letters followed by three digits?

Solution.

$$26 \times 26 \times 26 \times 10 \times 10 \times 10 = 17,576,000$$

Exercise 60. *What is the number of subsets of set composed of n elements.*

Problem. How many functions are there from a set with m elements to a set with n elements?

Solution. Suppose the elements in the domain are $a_1, \ldots, a_m$:

$f(a_1) =$	n *possibilities*
$f(a_2) =$	n *possibilities*
$\vdots$	$\vdots \quad \vdots$
$f(a_m) =$	n *possibilities*
Total	n^m

Exercise 61. *How many different functions there are from a set with* 3 *elements to a set with* 5 *elements.*

Problem. How many one-to-one functions are there from a set with m elements to a set with n elements ($m \leq n$).

Solution. Suppose the elements in the domain are $a_1, \ldots, a_m$:

$f(a_1) =$	n *possibilities*
$f(a_2) =$	$(n-1)$ *possibilities, because the function is one to one*
$\vdots$	$\vdots$ $\vdots$
$f(a_m) =$	$(n-(m-1))$ *possibilities*
$Total$	$n \times (n-1) \times \cdots \times (n-m+1) = \frac{n!}{(n-m)!}$

Exercise 62. *How many different one-to-one functions there are from a set with* 3 *elements to a set with* 5 *elements.*

7.2 The Sum Rule

Theorem 5. The Sum Rule. *Suppose that a task T can be done in one of t_1 ways, or in one of t_2 ways, ... or in one of t_n ways, where none of the t_i ways are the same as any of the t_j ways for any i, j so that $1 \leq i < j \leq n$. Then the number of possible ways to perform T is $t_1 + \cdots + t_n$.*

The sum rule is often phrased in terms of sets as follows: If $A_1, \ldots A_m$ are pairwise disjoint finite sets, then the number of elements in the union of these sets is the sum of the numbers of elements in each of these sets:

$$|A_1 \cup A_2 \cup \cdots \cup A_m| = |A_1| + |A_2| + \cdots + |A_m|$$

Problem. A student can choose a computer project from one of the three lists. No project is on more than one list. The three lists contain 23, 15 and 19 possible projects respectively. How many possible projects are there to choose from?

Solution. $23 + 15 + 19 = 57$.

Combining the Product Rule and the Sum Rule Many counting problems cannot be solved using just the sum or just the product rule. Yet, they can be solved using both of these rules in combination.

Problem. Each user on a computer system has a password, which is six to eight characters long, where each character is an uppercase letter or a digit. How many possible passwords are there?

Solution. Let P_6, P_7 and P_8 stand for numbers of possible ways to create passwords of length 6, 7, and 8 respectively. The total number of passwords of the described system is $P_6 + P_7 + P_8$ due to the sum rule. On the other hand by the product rule $P^6 = 36^6$, $P^7 = 36^7$, $P^8 = 36^8$ (indeed, there are 26 uppercase letter and 10 digits). The answer follows: $36^6 + 36^7 + 36^8$.

7.3 The Pigeonhole Principle

Theorem 6. *If k is a positive integer and $k+1$ or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.*

Proof. By contradiction. Suppose none of the boxes contain more than one object. It follows that there are at most k objects which contradicts to our assumption that there are at least $k+1$ objects. □

Problem. Among any group of 367 people, there must be at least two people with the same birthday, because there are only 366 possible birthdays.

Problem. How many students must be in a class to guarantee that at least two students receive the same score on the final exam – assuming the exam is graded on a scale from 0 to 100 points? By pigeon hole principle: at least 102.

Theorem 7 (Generalized Pigeonhole Principle)**.** *If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects, where $\lceil x \rceil$ denotes a ceileing function that maps a real number x to the smallest integer not less than x.*

Problem. Among 100 people there are at least $\lceil 100/12 \rceil = 9$ who were born in the same month.

Problem. What is the minimum number of students required in a discrete math class to be sure that at least six will receive the same grade – assuming five possible grades: A, B, C, D, and F?

The smallest number N that satisfies the following condition

$$\lceil N/5 \rceil = 6$$

is 26. This number represents the solution to the problem by generalized Pigeonhole principle.

Exercise 63. *At least how many cards must be selected from a standard deck of 52 cards to guarantee that at least three cards of the same suit are chosen? The number of distinct suits is* 4*.*

7.4 Permutations and Combinations

Permutations A *permutation* is an ordered arrangement of the objects in a given set. For example, for a set $\{a \;\; b \;\; c\}$ of three elements there are 6 different permutations:

$$a \;\; b \;\; c,\; a \;\; c \;\; b,\; b \;\; a \;\; c,\; b \;\; c \;\; a,\; c \;\; a \;\; b,\; c \;\; b \;\; a$$

For a set of n elements there are $n!$ permutations.

An *r-permutation* is an ordered arrangement of r elements of a given set. For example, for a set $\{a \;\; b \;\; c\}$ there are 6 different 2-permutations:

$$a \;\; b,\; a \;\; c,\; b \;\; a,\; b \;\; c,\; c \;\; a,\; c \;\; b.$$

For a set of n elements and an integer r where $1 \leq r \leq n$, there are

$$n \times (n-1) \times (n-2) \times \cdots \times (n-(r-1))$$

r-permutations. We denote this number by $P(n, r)$. It is easy to show that

$$P(n, r) = \frac{n!}{(n-r)!}.$$

Problem. How many ways are there to select a first-prize winner, a second-prize winner, and a third-prize winner from 100 different people who have entered a contest?

Solution.

$$P(100, 3) = n \times (n-1) \times (n-2) = 100 \times 99 \times 98.$$

Problem. Suppose there are eight runners in a race. The winner receives a gold medal, the second-place finisher receives a silver medal, and the third-place finisher receives a bronze medal. How many different ways are there to award these medals, if all possible outcomes of the race can occur and there are no ties?

Solution.

$$P(8, 3) = 8 \times 7 \times 6 = 336$$

Problem. How many permutations of the letters ABCDEFGH contain the string ABC?

Solution. The string ABC can occur in 6 positions.

$$\begin{array}{ll} ABCp_4p_5p_6p_7p_8 \\ p_1ABCp_5p_6p_7p_8 \\ \vdots \quad\quad \vdots \\ p_1p_2p_3p_4p_5ABC \end{array}$$

For each of these 6 situations we have 5! permutations of 5 remaining elements $\{D \; E \; F \; G \; H\}$. Thus the answer is $6 \times 5!$.

Combinations An *r-combination* of elements of a set A is an unordered selection of r elements from A. Thus an r-combination is a subset of A whose cardinality is r.

The number of r-combinations of a set with cardinality n is denoted by $C(n, r)$, which is also sometimes written as $\binom{n}{r}$. We also say that "n choses r". The number $\binom{n}{r}$ is called *binominal coefficient*.

Theorem 8. *For a set whose cardinality is n, where n is a nonnegative integer an r is an integer with $0 \leq r \leq n$:*

$$C(n, r) = \frac{n!}{r!(n-r)!}$$

Corollary 1. *Let n and r be nonnegative integers with $r \leq n$. Then*

$$C(n, r) = C(n, n-r)$$

Problem. How many different committees of three students can be formed from a group of four students?

Solution.

$$C(4, 3) = \frac{4!}{3!1!} = \frac{4}{1} = 4$$

Problem. How many poker hands of five cards can be dealt from a standard deck of 52 cards?

Solution.

$$C(52, 5) = \frac{52!}{5!47!} = \frac{52 \times 51 \times 50 \times 49 \times \cdots \times 48}{5 \times 4 \times 3 \times 2 \times 1} = 26 \times 17 \times 10 \times 49 \times 12 = 2,598,960$$

Problem. How many bit strings of length n contain exactly r 1's?

Solution. $C(n, r)$. (Recall how we used binary strings to model subsets of a set.)

Problem. Suppose there are 9 faculty members in the mathematics department and 11 in the computer science department. How many ways are there to select a committee to develop a discrete math course at a school if the committee is to consist of three faculty members from the mathematics department and four from the computer science department.

Solution.

$$C(9, 3) \times C(11, 4)$$

8 Relations

8.1 Relations and their Kinds: Reflexive, Symmetric, Transitive

Many examples in this section refer to comparison operations $=, <, >, \leq$, and $\geq$. Their properties are captured by axioms of real numbers (see for instance, Appendix I in K. Rosen textbook.)

Any condition on a pair of elements of a set A defines a *binary relation*, or simply *relation*, on A. For instance, the condition $x < y$ defines a relation on the set $\mathbf{R}$ of real numbers (or on any other set of numbers). If R is a relation, the formula xRy expresses that R holds for the pair x, y.

A relation R can be characterized by the set of all ordered pairs (x, y) such that xRy. In mathematics, it is customary to talk about a relation as it were the same thing as the corresponding set of ordered pairs. For instance, we can say that the relation $<$ on the set $\{1, 2, 3, 4\}$ is the set

$$\{(1,2),\ (1,3),\ (1,4),\ (2,3),\ (2,4),\ (3,4)\}.$$

A relation R on a set A is said to be *reflexive* if, for all $x \in A$, xRx. For instance, the relations $=$ and $\leq$ on the set $\mathbf{R}$ (or on any set of numbers) are reflexive, and the relations $\neq$ and $<$ are not.

A relation R on a set A is said to be *symmetric* if, for all $x, y \in A$, xRy implies yRx. For instance, the relations $=$ and $\neq$ on $\mathbf{R}$ are symmetric, and the relations $<$ and $\leq$ are not.

A relation R on a set A is said to be *transitive* if, for all $x, y, z \in A$, xRy and yRz imply xRz. For instance, the relations $=$, $<$ and $\leq$ on the set $\mathbf{R}$ are transitive, and the relation $\neq$ is not.

Problem. Consider a relation on the set $\{a, b, c\}$ defined as follows:

$$R = \{(a,a), (a,b), (b,a), (b,b), (c,c)\}.$$

Illustrate that this relation is reflexive, symmetric, and transitive.

Solution. To illustrate that R is reflexive, we have to verify that for every member x of the set $\{a, b, c\}$, pair (x, x) is in R. Indeed, pairs (a, a), (b, b), and (c, c) are in R.

To show that R is symmetric, we have to verify that for any pair of elements x, y from $\{a, b, c\}$, the statement

$$\textit{if } (x,y) \textit{ is in } R \textit{ then } (y,x) \textit{ is in } R \tag{43}$$

holds. The table below lists all possible assignments for x and y and provides details on evaluation of the statement in question given each assignment.

x	y	(x, y) is in R	(y, x) is in R	statement (43)
a	a	t	t	t
a	b	t	t	t
a	c	f		t
b	a	t	t	t
b	b	t	t	t
b	c	f		t
c	a	f		t
c	b	f		t
c	c	t	t	t

(44)

Recall that t stands for true (holds) and f stands for false (does not hold). The last column illustrates that under any assignment statement (43) holds. Thus, relation R is symmetric. The fourth column of the table is missing some of the entries, this is due to the fact that when the antecedent of the *if-then* statement is evaluated to f the value of the consequent is immaterial as the given *if-then* statement as a whole will be evaluated to t. (Remember the truth table of the implication symbol in propositional logic).

To show that R is transitive, we have to verify that for any triple of elements x, y, z from $\{a, b, c\}$, the statement

$$\textit{if pairs } (x,y) \textit{ and } (y,z) \textit{ are in } R \textit{ then } (x,z) \textit{ is in } R \tag{45}$$

holds. To do so it is is sufficient to construct a table in style of (44) that lists all possible assignments for triple x, y, z and verify that the column that presents evaluation of proposition (45) under each assignment contains value t. This table must contain 27 rows each representing a dstinct possible assignment. Yet, we know that when the antecedent of (45), namely, statement *pairs* (x, y) *and* (y, z) *are in* R does not hold (in other words, evaluates to f) then the statement (45) evaluates to t. The table that we present below does not list the assignments of x, y, z that result in evaluating the antecedent of (45) to f (such as, for example, an assignment $x = a$, $y = b$, $z = c$ or an assignment $x = c$, $y = a$, $z = a$).

x	y	z	pairs (x, y) and (y, z) are in R	(x, z) is in R	statement (45)
a	a	a	t	t	t
a	a	b	t	t	t
a	b	a	t	t	t
a	b	b	t	t	t
b	a	a	t	t	t
b	a	b	t	t	t
b	b	a	t	t	t
b	b	b	t	t	t
c	c	c	t	t	t

Exercise 64. *Argue that any relation on any singleton set is transitive.*

Problem. Consider a relation on the set of all people defined as follows:

$$R = \{(a, b) \mid a \text{ is shorter than } b\}.$$

Is R transitive?

Solution. We say that a person a is shorter than a person b, when the height of a person a is less than the height of a person b, where height is understood as a real number in, say, metric scale. In other words, this question can be reformulated as a question on whether the comparison relation less than, i.e., $<$, (on the subset of real numbers composed of the values of heights of all the people) is transitive. As discussed earlier $<$ relation is transitive.

8.2 Equivalence Relations and Partitions

An equivalence relation is a relation that is reflexive, symmetric, and transitive.

A *partition* of a set A is a collection P of non-empty subsets of A such that every element of A belongs to exactly one of these subsets. For instance, here are some partitions of $\mathbf{R}$:

$$\begin{aligned} P_1 &= \{\{0, 2, 4, \dots\}, \{1, 3, 5, \dots\}\}, \\ P_2 &= \{\{0, 1\}, \{2, 3\}, \{4, 5\}, \dots\}, \\ P_3 &= \{\{0\}, \{1\}, \{2\}, \{3\}, \dots\}. \end{aligned}$$

If P is a partition of a set A then the relation "x and y belong to the same element of P" is an equivalence relation.

Problem. Find the partition of the set $\{1, 2, \dots, 6\}$ so that each of its elements A corresponds to the equivalence relation $|x - 3| = |y - 3|$ on A.

Solution. The condition $|x - 3| = |y - 3|$ holds when $x = 1$ and $y = 5$. It holds also when $x = 2$ and $y = 4$. So the partition is $\{\{1, 5\}, \{2, 4\}, \{3\}, \{6\}\}$. Indeed, the relation $|x - 3| = |y - 3|$ is an equivalence relation on each of the members of the partition. Consider the set $\{1, 5\}$, the relation $|x - 3| = |y - 3|$ can be written as the following set of the ordered pairs:

$$\{(1, 1), (1, 5), (5, 1), (5, 5)\}.$$

For the set $\{2, 4\}$, the relation $|x - 3| = |y - 3|$ can be written as the following set of the ordered pairs:

$$\{(2, 2), (2, 4), (4, 2), (4, 4)\}.$$

For the sets $\{3\}$ and $\{6\}\}$, the relations $|x - 3| = |y - 3|$ can be written as the following sets of the ordered pairs $\{(3, 3)\}$ and $\{(6, 6)\}$ respectively. It is easy to see that all listed relations are reflexive, symmetric, and transitive. Hence they are also equivalence relations.

Order Relations A relation R on a set A is said to be *antisymmetric* if, for all $x, y \in A$, xRy and yRx imply $x = y$. For instance, the relation $\leq$ on $\mathbf{R}$ is anti-symmetric.

A *partial order* is a relation that is reflexive, anti-symmetric, and transitive. For instance, the relation $\leq$ on $\mathbf{R}$, and the relation $\subseteq$ on $\mathcal{P}(A)$ for any set A are partial orders.

A *total order* on a set A is a partial order such that for all $x, y \in A$, xRy or yRx. For instance, $\leq$ is total.

Problem. A bit string x is called a *substring* of a bit string y if there exist bit strings z_1, z_2 such that $z_1xz_2 = y$. For instance, the substrings of 0111 are

$$\epsilon,\ 0,\ 1,\ 01,\ 11,\ 011,\ 111, 0111.$$

Is the relation "x is a substring of y" an equivalence relation? a partial order? a total order?

Solution. The substring relation is reflexive and transitive. But it is not symmetric (for instance, 0 is a substring of 01, but not the other way around). Consequently it is not an equivalence relation. It is antisymmetric: if x is a substring of y and y is a substring of x then $x = y$. Consequently it is partial order. But it is not total (for instance, 0 is not a substring of 1, and 1 is not a substring of 0).

Table below summarizes properties on relations that we discussed earlier for comparison relations on the set $\mathbf{R}$ of real numbers (or on any set of numbers).

RELATION	reflexive	symmetric	transitive	equivalence	antisymmetric	partial order	total order
$=$	✓	✓	✓	✓	✓	✓	
$<$,$>$			✓		✓		
$\leq$,$\geq$	✓		✓		✓	✓	✓

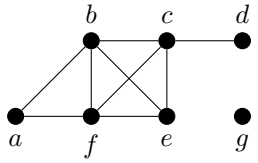
9 Graphs

9.1 Undirected Graphs

An *undirected graph* G is defined by two sets: a set V of objects called the *vertices* (or *nodes*) of G, and a collection E of two-element subsets $\{u, v\}$ of V, called the *edges* of the graph. The vertices u, v are called the *ends* of the edge $\{u, v\}$. When we draw a graph, we usually show every vertex as a small circle, and every edge as a line segment joining its two ends. For instance, the graph with the vertices $a,\ b,\ c,\ d,\ e,\ f,\ g$ and the edges

$$\{a,b\},\ \{b,c\},\ \{c,d\},\ \{a,f\},\ \{f,e\},\ \{b,e\},\ \{c,f\},\ \{b,f\},\ \{c,e\}$$

looks like this:



If a graph has an edge with the ends u, v, we say that u and v are *adjacent*. The adjacency relation is symmetric and irreflexive. The number of vertices adjacent to a vertex v is called the *degree* of v. For instance, the degree of a in the graph shown above is 2, and the degree of g is 0. Vertices of degree 0, such as g, are called *isolated*.

The *adjacency matrix* of a graph with n vertices is the $n \times n$ matrix such that its entry in row u and column v is 1 if u, v are adjacent, and 0 otherwise. For instance, here is the adjacency matrix of the graph shown above:

	a	b	c	d	e	f	g
a	0	1	0	0	0	1	0
b	1	0	1	0	1	1	0
c	0	1	0	1	1	0	0
d	0	0	1	0	0	0	0
e	0	1	1	0	0	1	0
f	1	1	0	0	1	0	0
g	0	0	0	0	0	0	0

The adjacency matrix of any graph is symmetric, and its main diagonal consists of zeroes.

A *complete graph on n vertices*, denoted by K_n, contains an edge between each pair of vertices.

A graph is called *bipartite* if its vertex set V can be partitioned into two disjoint sets V_1 and V_2 so that every edge has one vertex in V_1 and one in V_2.

Paths in a Graph A *path* in a graph is a list

$$v_1, v_2, \ldots, v_k \tag{46}$$

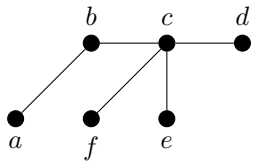
of vertices such that every two consecutive vertices v_i, v_{i+1} are adjacent. About this path we say that it is a path *from v_1 to v_k*. A path is *simple* if its vertices are distinct from one another. A path (46) is a *cycle* if $k > 2$, $v_k = v_1$, the vertices $v_1, v_2, \ldots, v_{k-1}$ are all distinct, and the edges $\{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}$ are all distinct. For instance, a, b, c, e is a simple path in the graph above. Path a, b, c, e, f, a is a cycle in that graph; note how b, c, e, f, a, b is also a cycle (graphically, we are not able to distinguish these cycles).

The relation "there is a path from u to v" is an equivalence relation. When the elements of some set A have a notion of equivalence, formalized as an equivalence relation, defined on them, then one may split the set A into *equivalence classes*. These equivalence classes are constructed so that elements a and b belong to the same equivalence class if and only if they are equivalent.

The equivalence classes of the relation "there is a path from u to v" are called the *connected components* of the graph. For instance, the connected components of the graph above are $\{a, b, c, d, e, f\}$ and $\{g\}$. If u and v belong to the same connected component then the distance from u to v is defined as the minimum number of edges in a path from u to v. For instance, the distance from a to c is 2.

If for every pair of vertices u, v there is a path from u to v then we say that the graph is *connected*.

Trees A *tree* is a connected graph that doesn't contain cycles. For instance, the graph



is a tree.

A *rooted tree* is a tree with one distinguished vertex, called the *root*. For every vertex v in a rooted tree, there is unique path from the root to v, The vertices that belong to that path are called the *ancestors* of x; if the last edge of that path is $\{y, x\}$ then we say that y is the *parent* of x, and x is a *child* of y. A vertex that doesn't have children is called a *leaf*.

A tree with n vertices has $n - 1$ edges: for every vertex v other than the root, the corresponding edge connects v with its parent.

9.2 Directed Graphs

In a *directed graph*, edges are ordered pairs of vertices, rather than two-element subsets. About an edge $\langle u, v\rangle$ we say that it *leaves* u and *enters* v. When we draw a directed graph, we show an edge $\langle u, v\rangle$ as an arrow from u to v.

The number of edges that enter a vertex v is called the *in-degree* of v. The number of edges that leave v is called the *out-degree* of v. The *adjacency matrix* of a directed graph with n vertices is the $n \times n$ matrix such that its entry in row u and column v is 1 if there graph has an edge that leaves u and enters v, and 0 otherwise.

A *path* in a directed graph is a list (46) of vertices such that for every two consecutive vertices v_i, v_{i+1} the graph has an edge that leaves v_i and enters v_{i+1}.

The relation "there is a path from u to v and a path from v to u" is an equivalence relation. Its equivalence classes are called the *strongly connected components* of the graph.

10 Elements on Proving Partial Correctness of Programs

A Toy Programming Language Let us recall a toy programming language (TPL) from earlier lecture notes. In the programming language TPL that we'll be using, all variables are integer variables, so that there is no need to declare them. *Integer expressions* are formed from integer variables and integer constants using the operators $+$, $-$ and $\times$. *Boolean expressions* are formed from equalities and inequalities between integer expressions using the propositional connectives $\wedge$, $\vee$, $\neg$. *Assignments* are expressions of the form $v \leftarrow e$ where v is an integer variable and e is an integer expression. *Statements* are formed from assignments and the statement **skip** using three constructs:

- sequential composition
$$P_1 \,; \ldots \,; P_n$$
where $P_1, \ldots, P_n$ are statements,
- conditional composition
$$\textbf{if } B \textbf{ then } P_1 \textbf{ else } P_2 \textbf{ endif}$$
where B is a Boolean expression and P_1, P_2 are statements,
- loop
$$\textbf{while } B \textbf{ do } P \textbf{ enddo}$$
where B is a Boolean expression and P is a statement.

Partial Correctness Let P be a statement, and let F and G be conditions (for instance, Boolean expressions). The expression
$$\{F\}\, P\, \{G\}$$
means that whenever F is true prior to the execution of P, and P terminates, G is true upon termination. This expression reads: P is *partially correct* with respect to *precondition* F and *postcondition* G.

For instance, the assignment $n \leftarrow n + 3$ is partially correct with respect to the precondition $n = 5$ and the postcondition $n = 8$:
$$\{n = 5\}\, n \leftarrow n + 3\, \{n = 8\}.$$

Proving Partial Correctness If v is an integer variable occurring in a condition F, and e is an integer expression, then F_e^v stands for the result of substituting e for v in F. For instance,
$$(n > m)_{n+3}^{n}$$
stands for
$$n + 3 > m.$$

The following rules can be used to prove partial correctness.

Rule of Assignment: $\{F_e^v\}\, v \leftarrow e\, \{F\}$.

Example: $\{n + 3 > m\}\, n \leftarrow n + 3\, \{n > m\}$.

Rules of Consequence:
$$\frac{\{F\}\, P\, \{G\} \qquad G \rightarrow H}{\{F\}\, P\, \{H\}}, \qquad \frac{F \rightarrow G \qquad \{G\}\, P\, \{H\}}{\{F\}\, P\, \{H\}}.$$

For instance, since $n > m$ implies $n > m - 5$, from the previous example we can conclude that
$$\{n + 3 > m\}\, n \leftarrow n + 3\, \{n > m - 5\}.$$

Since $n + 1 > m$ implies $n + 3 > m$, we can also conclude that
$$\{n + 1 > m\}\, n \leftarrow n + 3\, \{n > m\}.$$

Basic rules:

$$\frac{}{\{true\}}$$

$$\frac{\{F\} \qquad F \to G}{\{G\}}.$$

Rule of Composition:

$$\frac{\{F\}\, P_1\, \{G\} \qquad \{G\}\, P_2\, \{H\}}{\{F\}\, P_1; P_2\, \{H\}}.$$

If-then-else Rule:

$$\frac{\{F \wedge C\}\ P_1\ \{G\} \qquad \{F \wedge \neg C\}\ P_2\ \{G\}}{\{F\}\ \textbf{if}\ C\ \textbf{then}\ P_1\ \textbf{else}\ P_2\ \textbf{endif}\ \{G\}}.$$

Example: to prove

$$\{x = X \wedge y = Y\}\ \textbf{if}\ x < y\ \textbf{then}\ z \leftarrow x\ \textbf{else}\ z \leftarrow y\ \textbf{endif}\ \{z = \min(X, Y)\}$$

we need to verify that

$$\{x = X \wedge y = Y \wedge x < y\}\ z \leftarrow x\ \{z = \min(X, Y)\}$$

and

$$\{x = X \wedge y = Y \wedge \neg(x < y)\}\ z \leftarrow y\ \{z = \min(X, Y)\}.$$

We will use

$$\textbf{if}\ C\ \textbf{then}\ P\ \textbf{endif}$$

as shorthand for

$$\textbf{if}\ C\ \textbf{then}\ P\ \textbf{else skip endif}.$$

Skip Rule: $\{F\}$ **skip** $\{F\}$.

While-do Rule:

$$\frac{\{F \wedge C\}\ P\ \{F\}}{\{F\}\ \textbf{while}\ C\ \textbf{do}\ P\ \textbf{enddo}\ \{F \wedge \neg C\}}.$$

For instance, if we can prove

$$\{x = 2^i \wedge i < n\}\ x \leftarrow x \times 2; i \leftarrow i + 1\ \{x = 2^i\}$$

then we will be able to conclude that

$$\begin{array}{l}\{x = 2^i\} \\ \quad \textbf{while}\ i < n\ \textbf{do}\ x \leftarrow x \times 2; i \leftarrow i + 1\ \textbf{enddo} \\ \{x = 2^i \wedge \neg(i < n)\}.\end{array}$$

We note that a condition $x = 2^i$ above is called a loop invariant. A *loop invariant* is a property of a program loop that is true before (and after) each iteration. Knowing loops invariants adds to understanding the effect of a loop and allows us to reason of its correctness.

Proofs as Annotated Programs Proof of $\{n > 10\}\ m \leftarrow n + 2\ \{m > 10\}$:

$$\begin{array}{l}\{n > 10\}\\ \{n + 2 > 10\}\\ m \leftarrow n + 2\\ \{m > 10\}\end{array}$$

Proof of $\{m = n\}\ m \leftarrow n + 2\ \{m > n\}$:

$$\begin{array}{l}\{m = n\}\\ \{true\}\\ \{n + 2 > n\}\\ m \leftarrow n + 2\\ \{m > n\}\end{array}$$

Proof of $\{i = 1 \wedge j = 2\}\ m \leftarrow i;\ n \leftarrow j\ \{m = 1 \wedge n = 2\}$:

$$\begin{array}{l}\{i = 1 \wedge j = 2\}\\ m \leftarrow i;\\ \{m = 1 \wedge j = 2\}\\ n \leftarrow j\\ \{m = 1 \wedge n = 2\}\end{array}$$

Proof of $\{i = 1 \wedge j = 2\}\ m \leftarrow i; n \leftarrow j\ \{n > m\}$:

$$\begin{array}{l}\{i = 1 \wedge j = 2\}\\ \{j > i\}\\ m \leftarrow i;\\ \{j > m\}\\ n \leftarrow j\\ \{n > m\}\end{array}$$

Exercise 65. *For each of the following assertions, determine if it is true. If it is then present its proof as an annotated program. It not, give a counterexample.*

(a) $\{n > 3\}\ n \leftarrow n + 3\ \{n > 8\}$

(b) $\{n > 7\}\ n \leftarrow n + 3\ \{n > 8\}$

(c) $\{n > k\}\ n \leftarrow n + 3\ \{n > k\}$

(d) $\{n > k\}\ n \leftarrow n + 3\ \{n > m\}$

(e) $\{n > k\}\ n \leftarrow n + 3;\ k \leftarrow k + 2\ \{n > k\}$

(f) $\{n \neq 0\}\ n \leftarrow n + 3;\ n \leftarrow n \times n\ \{n \neq 9\}$

(g) $\{n \neq 0\}\ n \leftarrow n \times n;\ n \leftarrow n + 3\ \{n \neq 3\}$

(h) $\{x = 7 \wedge y = 10\}\, temp \leftarrow x; x \leftarrow y; y \leftarrow temp\, \{x = 10 \wedge y = 7\}$

Exercise 66. *For each of the following assertions, show the two premises from which it can be derived by the if-then-else-rule, and present proofs as annotated programs.*

(a) $\{x = 3\}$ **if** $x < 0$ **then** $y \leftarrow 0$ **else** $y \leftarrow 1$ **endif** $\{y = 1\}$

(b) $\{true\}$ **if** $x < 0$ **then** $x \leftarrow 0$ **endif** $\{x \geq 0\}$

Exercise 67. *Determine which of the following conditions are loop invariants for the loop*

$$\textbf{while } i < j \textbf{ do } i \leftarrow i + 1; j \leftarrow j - 1 \textbf{ enddo}.$$

(a) $i + 20 = j$

(b) $i + j = 20$

(c) $i^2 + j^2 = 20$

Justify your answers.

Acknowledgment The creation of this book was made possible with the assistance of *Affordable Content Grants* provided by the University of Nebraska Omaha in 2024.