Lecture notes for the 2023/24 lectures

Mathematical methods for Computer Science I & II and Discrete Mathematics I & II

University of Fribourg
Livio Liechti

Lecture notes adapted from Ivan Izmestiev's notes for his 2018/19 lectures

Contents

I Basic combinatorics **1**

1 How to count 1
- 1.1 Sum rule 1
- 1.2 Product rule 1
- 1.3 Difference rule 2
- 1.4 Quotient rule 3

2 Counting maps and subsets 3
- 2.1 Maps 3
- 2.2 Counting all maps 4
- 2.3 Counting injective maps and ordered choices 6
- 2.4 Unordered choices 7
- 2.5 Birthday problem 7

3 Many faces of $\binom{n}{k}$ 7
- 3.1 Subsets or unordered choices 7
- 3.2 Monotone paths 8
- 3.3 Pascal's triangle 9
- 3.4 Binomial theorem 11
- 3.5 Compositions 12
- 3.6 Multisets 13

4 Multinomial coefficients 13
- 4.1 Words with repeating letters 13
- 4.2 Multinomial theorem 15
- 4.3 Monotone paths in higher dimensions 15

5 Inclusion-exclusion formula 15
- 5.1 The formula 15
- 5.2 De Montmort problem, or counting the derangements 17
- 5.3 Euler's totient function 19

II Graph theory **21**

1 Basic notions 21
- 1.1 Types of graphs 21
- 1.2 Some graphs known by names 22
- 1.3 Isomorphic graphs and subgraphs 23

1.4 Incidence and adjacency 24
1.5 Connectivity and components 25
1.6 Eulerian graphs 26
2 Trees 28
2.1 Basics 28
2.2 Leaves 28
2.3 The number of edges in a tree 29
2.4 Spanning trees 30
2.5 The number of spanning trees 31
2.6 Minimum spanning tree: Kruskal's algorithm 32
3 Planar graphs 34
3.1 Basics 34
3.2 Euler's formula 36
3.3 Planarity criteria 38
3.4 Duality for embedded graphs 39
4 Matchings 41
4.1 Basics 41
4.2 Augmenting paths and maximum matchings 42
4.3 Matchings in bipartite graphs: Hall's theorem 44

III Propositional logic **47**
1 Syntax and semantics of propositional logic 48
1.1 Propositional formulas 48
1.2 Truth tables 49
1.3 Satisfiability, tautologies, logical equivalence 50
1.4 Boolean functions 53
1.5 Disjunctive and conjunctive normal forms 56
2 Proof theories 57
2.1 Deductive systems 57
2.2 A Hilbert system 58
2.3 Gentzen's sequent calculus: the idea 59
2.4 Sequents and inference rules 60
2.5 Axioms and proofs 62
2.6 Soundness of the sequent calculus 63
2.7 Closed deduction trees and completeness of the sequent calculus 63
2.8 A byproduct: CNF and DNF 64

IV Predicate logic **65**
1 Syntax and semantics of predicate logic 66
1.1 First-order languages 66
1.2 Free and bound variables 68
1.3 Closed formulas and universal closure 69
1.4 First-order structures 69

1.5 Propositional logic inside predicate logic 70
2 Proof theory 71
2.1 Substitutions 71
2.2 Inference rules 72
2.3 Completeness of the sequent calculus 74
3 Gödel's incompleteness theorems 77
3.1 Theories and models 77
3.2 Models and proofs: Gödel's completeness theorem . . 79
3.3 Peano arithmetic 80
3.4 Recursive functions and recursive sets 80
3.5 Gödel's incompleteness theorems 82

V Combinatorics II **85**
1 Linear recursive sequences 85
1.1 Fibonacci sequence and Binet formula 85
1.2 Linear recursive sequences of order 2 86
1.3 Linear recursive sequences of higher order 88
1.4 The case of complex roots 88
1.5 An application of the Binet formula 90
2 Generating functions 90
2.1 Fibonacci again 90
2.2 Operations with formal power series 92
2.3 Linear recursive sequences and partial fraction decomposition 94
2.4 Generalized binomial theorem 96
2.5 Summary of the generating function method 98
3 Partition of integers 99
3.1 Money changing problem 99
3.2 Compositions again 100
3.3 Fibonacci once again 101
3.4 Partitions and their generating function 101
3.5 Infinite products of power series 103
3.6 Algebraic and bijective proofs 104
3.7 Recursive formulas for the number of partitions 105
3.8 More about partitions 106
4 Catalan numbers 108
4.1 Rooted binary trees 108
4.2 Generating function and the formula for c_n 109
4.3 Bracket-variable expressions 110
4.4 Triangulations of polygons 111
4.5 Dyck paths 112
4.6 A combinatorial proof of the formula for c_n 114

VI Automata theory **117**
1 Finite automata 117
1.1 Alphabets, words, and languages 117
1.2 Deterministic finite automata 118
1.3 Nondeterministic finite automata 120
1.4 Finite automata with epsilon-transitions 123
2 Regular expressions 126
2.1 Definition and examples 126
2.2 Equivalence of regular expressions and regular languages 128
3 Properties of regular languages 132
3.1 Closure under boolean operations 132
3.2 The pumping lemma 133
3.3 Closure under homomorphisms 136
3.4 Closure under inverse homomorphism 137
4 The Myhill-Nerode theorem 138
4.1 Equivalence of words with respect to a language . . . 138
4.2 The theorem 139
4.3 Minimization of a DFA 140
5 Context-free grammars and languages 142
5.1 Generating a language by a grammar 142
5.2 Grammars in Chomsky form 144
6 Pushdown automata 148
6.1 Definition 148
6.2 Instantaneous description 150
6.3 Equivalence of acceptance by final state and empty stack 151
6.4 Equivalence of PDA's and CFL's 151
7 Properties of context-free languages 153
7.1 Closure properties of CFLs 153
7.2 The pumping lemma for CFLs 154
7.3 Non-closure properties of CFLs 156
8 Turing machines 158
8.1 Definition 158
8.2 Modifications of Turing machines 160
8.3 Problems and languages 161
8.4 The universal language 161
8.5 Undecidability of the halting problem 161

Chapter I

Basic combinatorics

1 How to count

All sets in this chapter are finite.

For a finite set X, by $|X|$ we denote the number of elements in X, also called the *cardinality* of X.

1.1 Sum rule

If $X \cap Y = \varnothing$, then $|X \cup Y| = |X| + |Y|$.

More generally,

If the sets $X_1, \dots, X_n$ are pairwise disjoint (that is $X_i \cap X_j = \varnothing$ for all $i \neq j$), then $|X_1 \cup X_2 \cup \dots \cup X_n| = |X_1| + |X_2| + \dots + |X_n|$.

Formally, this follows by induction on n from the sum rule for two sets.

Later we will learn how to proceed if the sets are not disjoint.

1.2 Product rule

First, let us state a special product rule:

$$|X \times Y| = |X| \cdot |Y|.$$

Here $X \times Y$, the *Cartesian product* of X and Y, denotes the set of ordered pairs (x, y) with $x \in X$, $y \in Y$.

The elements of $X \times Y$ can be written in a table whose rows correspond to the elements of X, and the columns correspond to the elements of Y. This justifies the product rule.

Again, there is an extension to several sets:

$$|X_1 \times \dots \times X_n| = |X_1| \cdots |X_n|.$$

Example 1.1. A restaurant offers a choice of 3 first courses, 4 main courses, and 5 desserts. How many different full course dinners are there?

A full course dinner is an element of the Cartesian product

$$\{\text{first courses}\} \times \{\text{main courses}\} \times \{\text{desserts}\}.$$

Multiplying the cardinalities of these sets we obtain the answer: $3 \cdot 4 \cdot 5 = 60$ different full course dinners are possible.

Up to now we have considered independent choices, when the result of one choice does not influence the sets of subsequent choices. However, it is not the **set** of choices what matters, but rather the **number** of choices. This leads us to the general product rule:

> *If two consecutive choices are made, with m possibilities for the first choice and n possibilities for the second choice, then the number of all possible outcomes is equal to mn.*

Of course, this can be generalized to several consecutive choices if the number of possibilities for each choice is independent of the results of all previous choices.

Example 1.2. A person wants to go to a swimming pool once a week, and play tennis once a week, but not both on the same day. How many different schedules are there?

Choose a swimming day. There are 7 possibilities for this. When the choice is made, there remain 6 possible tennis days. So, there are $7 \cdot 6 = 42$ different schedules.

1.3 Difference rule

The set difference: $X \setminus Y = \{x \in X \mid x \notin Y\}$.

$$\text{If } X \supset Y, \text{ then } |X \setminus Y| = |X| - |Y|.$$

Instead of counting the number of "good" outcomes, one can count the number of "bad" ones and subtract it from the total number of outcomes.

Example 1.3. Two dice are thrown. What is the probability that at least one of them shows six?

We give two solutions. The first one uses the sum and the product rules. Consider three cases:

- Both dice show six. This is one outcome.
- The first dice shows six, the second does not. Five outcomes.
- The first dice does not show six, the second does. Five outcomes as well.

Thus in total we have $1+5+5 = 11$ possibilities. To compute the probability, we have to divide by the total number of possibilities, which is $6 \cdot 6 = 36$.

The second solution uses the difference rule. A "bad" outcome is one where neither of the dice shows six. For each of the dice there are 5 possibilities, so that this number is $5 \cdot 5 = 25$. To count the "good" outcomes, we subtract 25 from the number of all possible outcomes: $36 - 25 = 11$.

1.4 Quotient rule

> *The number of sheep in a herd is equal to the number of legs divided by four.*

This is a highly inefficient way of counting sheep. But if we see only the legs and cannot see the heads, then this is the only available way.

Example 1.4. Draw a convex n-gon and all of its diagonals. How many segments (sides and diagonals) do we get?

Every point belongs to $n-1$ segments. If we multiply this by the number of points, we get $n(n-1)$. But every segment was counted twice, because it has two endpoints (two "legs"). Thus the total number of segments is $\frac{n(n-1)}{2}$.

2 Counting maps and subsets

2.1 Maps

A *map* $f\colon X \to Y$ is a rule that associates to every element $x \in X$ a unique element of Y. The element associated to x is denoted by $f(x)$.

A map can be pictured as a collection of arrows going from elements of X to elements of Y. At every element of X one and only one arrow must start. By contrast, at an element of Y several arrows or none at all may end.

A map $f\colon X \to Y$ is called

- *injective*, if no two different elements of X are sent to the same element of Y: for every $x_1 \neq x_2$ we have $f(x_1) \neq f(x_2)$;
- *surjective*, if to every element of Y some element of X is sent: for every $y \in Y$ there is $x \in X$ such that $f(x) = y$;
- *bijective*, if it is injective and surjective.

Example 2.1. Let X be the set of all countries in the world, and Y be the set of all cities in the world. Define a map $f\colon X \to Y$ by letting $f(x)$ be the capital of the country x. Then f is injective (no city can be the capital of two countries), but not surjective (some cities are not capitals). Define a

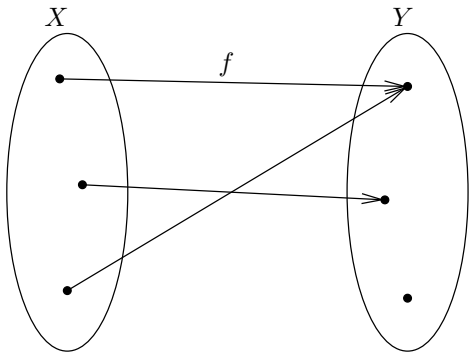

Figure 1: A map $f\colon X \to Y$.

map $g\colon Y \to X$ by letting $g(y)$ be the country in which the city y lies. Then g is surjective (in every country there is at least one city), but not injective (some countries have more than one city).

It is also interesting to look at the compositions $f \circ g$ and $g \circ f$...

A map is bijective iff at every element of Y ends exactly one arrow. By inverting the arrows we obtain the *inverse map* $f^{-1}\colon Y \to X$, which has the properties $f^{-1}(f(x)) = x$ for all $x \in X$ and $f(f^{-1}(y)) = y$ for all $y \in Y$.

If f is not bijective, then there is no inverse map f^{-1}. However, by abuse of notation one uses $f^{-1}(y)$ to denote the *preimage* of y:

$$f^{-1}(y) = \{x \in X \mid f(x) = y\}.$$

Similarly one can define the preimage $f^{-1}(B)$ of any subset $B \subset Y$.

Observe that

- f injective $\Leftrightarrow |f^{-1}(y)| \leq 1$ for all y;
- f surjective $\Leftrightarrow f^{-1}(y) \neq \varnothing$ for all y.

We can now formulate the quotient rule in the mathematical language.

> *If a map $f\colon X \to Y$ satisfies $|f^{-1}(y)| = k$ for all $y \in Y$, then $|Y| = \frac{|X|}{k}$.*

A special case of this is the bijection principle:

> *If a map $f\colon X \to Y$ is a bijection, then $|X| = |Y|$.*

2.2 Counting all maps

Theorem 2.2. *If $|X| = m$ and $|Y| = n$, then there are n^m different maps $X \to Y$.*

Proof. Without loss of generality, let $X = \{1, 2, \dots, m\}$. In order to define a map $f\colon X \to Y$ we need to make m choices, each time from n possibilities: $f(1)$ can take n different values, so can $f(2)$, and so on up to $f(m)$. Thus there are

$$\underbrace{n \cdots n}_{m} = n^m$$

different maps $X \to Y$. □

Theorem 2.3. *The number of different subsets of an n-element set is 2^n.*

Proof. Let $|X| = n$. With every subset $A \subset X$ we associate a map $\mathbf{1}_A\colon X \to \{0, 1\}$ (the *indicator function* of A) defined as

$$\mathbf{1}_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{if } x \notin A. \end{cases}$$

One can show that $A \mapsto \mathbf{1}_A$ is a bijection between the set of all subsets and the set of all maps $X \to \{0, 1\}$: different subsets define different maps and every map f is the indicator function of the subset $f^{-1}(1) \subset X$.

The number of all maps $X \to \{0, 1\}$ is $2^{|X|}$ by Theorem 2.2. By the bijection principle, the number of all subsets of X is the same. □

Remark 2.4. The set of all maps $X \to Y$ is sometimes denoted by Y^X, so that Theorem 2.2 can be formulated as $\left|Y^X\right| = |Y|^{|X|}$. This is not the only reason for the notation Y^X. One can show that $Z^{X \cup Y} = Z^X \times Z^Y$ for disjoint X and Y, and $Z^{X \times Y} = (Z^X)^Y$.

Also, the Cartesian power

$$X^n = \underbrace{X \times \cdots \times X}_{n} = \{(x_1, \dots, x_n) \mid x_i \in X \text{ for all } i\}$$

can be viewed as the set $X^{\{1,\dots,n\}}$: a sequence $(x_1, \dots, x_n)$ corresponds to a map $f\colon \{1, \dots, n\} \to X$, $f(i) = x_i$.

Theorem 2.5. *Tossing a coin n times can lead to 2^n different outcomes.*

Proof. Apply the product rule. The coin is making n choices, each time from two possibilities. Thus the number of outcomes is $\underbrace{2 \cdots 2}_{n} = 2^n$. □

Theorem 2.6. *The number of binary sequences of lenght n is 2^n.*

Proof. One can argue by the product rule again. Or, one can establish a bijection between the set of binary sequences and the set of outcomes when tossing a coin: one encodes an outcome by putting 1 for "heads" and 0 for "tails". It is easy to see that this is a bijection, thus by the previous theorem the number of binary sequences is 2^n. □

Remark 2.7. One might argue that the number of binary sequences of length n is 2^n because they represent all numbers from 0 to $2^n - 1$ in the binary system. But this result itself requires a proof, which is not easier than our combinatorial argument.

To summarize, we have found that the number of elements in each of the following sets is 2^n:

- Maps from an n-element set to $\{0, 1\}$.
- Subsets of an n-element set.
- Outcomes in tossing a coin n times.
- Binary sequences of length n.

Between any two of these sets there is a natural bijection. Some of these bijections were described above.

2.3 Counting injective maps and ordered choices

Theorem 2.8. *If $|X| = k$ and $|Y| = n$, then the number of injective maps from X to Y is $n(n-1)\cdots(n-k+1)$.*

Proof. Without loss of generality, $X = \{1, \ldots, k\}$. We have n possibilities to choose $f(1)$. After this, for $f(2)$ there remain $n-1$ possibilities, then $n-2$ possibilities for $f(3)$, and so on up to $f(k)$, for which $n-(k-1) = n-k+1$ possibilities remain.

The above argument works for $k \le n$, but the formula is true for $k > n$ as well: there are no injective maps in this case, and the product $n(n-1)\cdots(n-k+1)$ vanishes because it contains a zero factor. □

Corollary 2.9. *The number of bijective maps between two n-element sets is equal $n! = 1 \cdot 2 \cdots n$. The number of permutations of n elements is equal to $n!$.*

Indeed, a permutation of an n-element set X can be thought of as a bijection $\{1, 2, \ldots, n\} \to X$.

The same number appears as the answer to a different problem. Imagine that we have a bag with n balls. We take out k balls consecutively and lay them in a line in the order in which they were taken. How many different outcomes are there?

Theorem 2.10. *The number of ordered choices of k balls out of n is*

$$n(n-1)\cdot\ldots\cdot(n-k+1).$$

Proof. One can argue by the product rule, or one can interpret an ordered choice of k balls out of n as an injective map $f\colon \{1, \ldots, k\} \to \{1, \ldots, n\}$. Here $f(i)$ tells us which of the balls was taken on the i-th step. □

2.4 Unordered choices

Take the same bag with n balls. We take out k balls simultaneously. Equivalently, we take out k balls consecutively, but then forget the order in which they were taken. How many possibilities are there?

Theorem 2.11. *The number of unordered choices of k balls out of n is*

$$\frac{n(n-1)\cdot\ldots\cdot(n-k+1)}{k!}.$$

Proof. Let X be the set of all possible ordered choices of k balls out of n, and Y be the set of all unordered choices of k balls. There is a map $f\colon X \to Y$ (the "forgetful map") that associates to an ordered collection of k balls the same set of balls, but unordered. (The balls lying in a line are put into another bag.)

For $y \in Y$, what is the cardinality of its preimage $f^{-1}(y)$? This is the number of ways to order an unordered set of k balls. An ordering is a bijection to the set $\{1, 2, \ldots, k\}$, and from Corollary 2.9 we know that there are $k!$ of them. Therefore by the quotient rule we have

$$|Y| = \frac{|X|}{k!} = \frac{n(n-1)\cdot\ldots\cdot(n-k+1)}{k!}.$$

□

As we already said, an unordered choice of k balls out of n is also called a k-combination. Yet another name of this is a *k-element subset* of a given n-element set. (By definition, a set is an unordered collection of elements.)

Notation. The number of k-element subsets of an n-element set is denoted by $\binom{n}{k}$ (pronounced "n choose k").

2.5 Birthday problem

Given k people, what is the probability that some two of them have the same birthday? How big should k be for this probability to exceed $\frac{1}{2}$?

3 Many faces of $\binom{n}{k}$

The number $\binom{n}{k}$ that we have introduced in the previous lecture has several interpretations.

3.1 Subsets or unordered choices

This is our original definition: $\binom{n}{k}$ is the number of unordered choices of k elements out of n. In a more abstract language, this is the number of k-element subsets of an n-element set.

We have proved that

$$\binom{n}{k} = \frac{n(n-1)\cdot\ldots\cdot(n-k+1)}{k!} = \frac{n!}{k!(n-k)!}. \tag{1}$$

Theorem 3.1.

$$\binom{n}{k} = \binom{n}{n-k}$$

First proof. This is immediate from (1): if we replace k by $n-k$, then the factors in the denominator simply interchange. □

Second proof. The theorem can be proved even without knowing a formula for $\binom{n}{k}$, by a bijective argument. Associate to every subset of an n-element set its complement: $A \mapsto X \setminus A$. This is a bijection, and it sends k-element subsets to $(n-k)$-element subsets. Thus there are as many k-element subsets as there are $(n-k)$-element subsets. □

Theorem 3.2. *For every n we have*

$$\sum_{k=1}^{n} \binom{n}{k} = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n. \tag{2}$$

Proof. By Theorem 2.3, an n-element set has 2^n subsets. By definition, $\binom{n}{k}$ is the number of k-element subsets. (One may say that we are using the sum rule here: the set of all subsets is a disjoint union of sets of k-element subsets over all k from 0 to n.) □

Theorem 3.3. *The number $\binom{n}{k}$ is equal to the number of binary words of length n containing exactly k digits* 1*.*

Proof. A subset of $\{1, 2, \ldots, n\}$ can be encoded with a binary word: the i-th digit of this word is 1 if and only if the number i belongs to the subset. The words with k digits 1 correspond to the k-element subsets, hence there are $\binom{n}{k}$ such words. □

3.2 Monotone paths

Take the square lattice in the plane. The nodes of the lattice can be identified with pairs of integers (k, l) (Cartesian coordinates). Take two non-negative integers k, l and mark the points $(0, 0)$ and (k, l). A *lattice path* from $(0, 0)$ to (k, l) is a sequence of segments of the square lattice that leads from $(0, 0)$ to (k, l). A lattice path is called *monotone* if it runs only upwards and to the right. See an example on Figure 2.

Theorem 3.4. *The number of monotone paths from $(0, 0)$ to (k, l) is $\binom{k+l}{k}$.*

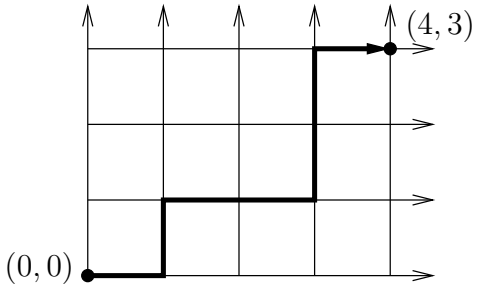

Figure 2: A monotone path.

Proof. A monotone path can be encoded by a binary word, where 1 stands for "a step to the right" and 0 stands for "a step upwards". In order to attain the point (k, l), we need to make k steps to the right and l steps upwards. Therefore the monotone paths from $(0, 0)$ to (k, l) correspond to words of length $k + l$ containing exactly k digits 1. From Theorem 3.3 we know that the number of such words is $\binom{k+l}{k}$. □

3.3 Pascal's triangle

Write two rows of 1 starting at the same place and going one downwards to the left and the other downwards to the right. Then fill in this frame row after row according to the rule that every number is equal to the sum of its top-left and top-right neighbors.

					1					
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1

One adopts the convention that the rows of the Pascal triangle as well as the numbers in each row are numbered starting from 0. Thus the row number n contains $n + 1$ entries numbered with 0 up to n.

Theorem 3.5. *The k-th number in the n-th row of the Pascal triangle is equal to $\binom{n}{k}$.*

We will need a lemma.

Lemma 3.6. *For every $0 \le k \le n$ the following identity holds:*

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}. \tag{3}$$

Proof. We know that $\binom{n}{k}$ is the number of binary words of length n with exactly k digits 1. There are two kinds of words like that: those that start with 1 and those that start with 0. How many words of each kind are there?

When we delete the first digit, we are left with a word of length $n-1$. For the words of the first kind, this word of length $n-1$ must contain $k-1$ digits 1. Thus there are $\binom{n-1}{k-1}$ words of the first kind.

Similarly, for a word of second kind we are left with a word of length $n-1$ that contains k digits 1. Thus there are $\binom{n-1}{k}$ words of the second kind.

Since every word is either of the first kind or of the second kind but not both, identity (3) holds. □

Proof of Theorem 3.5. Let us write the numbers $\binom{n}{k}$ in a triangle similar to the Pascal triangle:

$$\begin{array}{ccccccccccc}
 & & & & \binom{0}{0} & & & & & & \\
 & & & \binom{1}{0} & & \binom{1}{1} & & & & & \\
 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & & & \\
 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & & & \\
\binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4} & & \\
\binom{5}{0} & & \binom{5}{1} & & \binom{5}{2} & & \binom{5}{3} & & \binom{5}{4} & & \binom{5}{5}
\end{array}$$

The top-left neighbor of the number $\binom{n}{k}$ is $\binom{n-1}{k-1}$, the top-right neighbor is $\binom{n-1}{k}$. By Lemma 3.6, the numbers in the $\binom{n}{k}$-triangle satisfy the same rule that the numbers in the Pascal triangle: each number is the sum of its top-left and top-right neighbors. The outermost numbers $\binom{n}{0}$ and $\binom{n}{n}$ are also the same as in the Pascal triangle:

$$\binom{n}{0} = \binom{n}{n} = 1.$$

It follows that the $\binom{n}{k}$-triangle coincides with the Pascal triangle. (The formal argument here is proof by induction: if the n-th line of the $\binom{n}{k}$-triangle coincides with the n-th line of the Pascal triangle, then their $(n+1)$-st lines also coincide.) □

3.4 Binomial theorem

The binomial theorem is the generalization of the well-known formulas

$$(a+b)^2 = a^2 + 2ab + b^2, \quad (a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3.$$

Theorem 3.7. *For any $n \in \mathbb{N}$ we have*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}b^k$$
$$= \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \cdots + \binom{n}{n}b^n$$

The coefficients in the above formula come from the n-th row of the Pascal triangle. For example, by looking at the Pascal triangle we can conclude

$$(a+b)^5 = a^5 + 5a^4b + 10a^3b^2 + 10a^2b^3 + 5ab^4 + b^5.$$

Proof. How do we prove $(a+b)^2 = a^2 + 2ab + b^2$? For this, we write $(a+b)^2 = (a+b)(a+b)$, multiply every term inside the first pair of brackets with every term inside the second pair of brackets, and finally collect the like terms:

$$(a+b)^2 = (a+b)(a+b) = a^2 + ab + ba + b^2 = a^2 + 2ab + b^2.$$

What happens when we multiply out n pairs of brackets $(a+b)$?

$$(a+b)^n = \underbrace{(a+b)(a+b)\cdots(a+b)}_{n \text{ pairs of brackets}}$$

Before collecting the like terms, we obtain a sum of products of n factors, every factor being a or b. That is to say, we are writing down all words of length n consisting of letters a and b. When collecting the like terms, we ignore the order of letters in each word, counting only the number of a's and b's. That is to say, the term $a^{n-k}b^k$ occurs in our sum as often as there are (a,b)-words of length n with $n-k$ letters a and k letters b. But we know that there are $\binom{n}{k}$ such words, and the theorem follows. □

Instead of a and b we can substitute any numbers or expressions. For example, we have

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

Note that by substituting $x = 1$ we obtain a new (but a quite intricate) proof of Theorem 3.2.

Theorem 3.8. *For any n we have*

$$\binom{n}{0} - \binom{n}{1} + \cdots + (-1)^n \binom{n}{n} = 0$$

This is obvious for odd n: because of $\binom{n}{k} = \binom{n}{n-k}$ the summands can be split into pairs that cancel each other. For n even there is no cancellation.

Proof. Substitute $a = 1$ and $b = -1$ in the binomial theorem. □

3.5 Compositions

Definition 3.9. *A* composition *of a positive integer n is a representation of n as the sum of positive integers. The order of summands plays the role, for example $5 = 3 + 1 + 1$ and $5 = 1 + 3 + 1$ are different compositions of 5.*

Theorem 3.10. *The number of compositions of n from k parts is $\binom{n-1}{k-1}$.*

Proof. Lay n stones in a row. To form a composition of the number n we must separate these stones by $k - 1$ sticks. The numbers of stones between consecutive sticks sum up to n, thus form a composition of n. For example,

$$\bullet \,|\, \bullet \bullet \bullet \,|\, \bullet \quad \longleftrightarrow \quad 5 = 1 + 3 + 1$$

In how many ways can we put $k - 1$ sticks between n stones? There are $n - 1$ gaps, and we must choose $k - 1$ of them. There are $\binom{n-1}{k-1}$ ways to do this. The theorem is proved. □

Corollary 3.11. *For every positive integers $k \le n$ the equation*

$$x_1 + x_2 + \cdots + x_k = n$$

has $\binom{n-1}{k-1}$ solutions in positive integers.

Proof. This is nothing else but a formal description of a composition of the number n. □

Definition 3.12. *A* weak composition *of a positive integer n is a representation of n as the sum of non-negative integers.*

Theorem 3.13. *The number of weak compositions of n from k parts is $\binom{n+k-1}{k-1}$.*

First proof. Let us establish a bijection between the weak compositions of n from k parts and the compositions of $n + k$ from k parts. Indeed, adding 1 to every summand in a weak composition of n transforms it into a (strong) composition of $n + k$. In the opposite direction, subtracting 1 from every summand of a composition of $n + k$ transforms it into a weak composition of n. This is a one-to-one correspondence (a bijection). By Theorem 3.10, the number of (strong) compositions of $n + k$ from k parts is $\binom{n+k-1}{k-1}$, and we are done. □

Second proof. You have n stones and $k-1$ sticks. Mark $n+k-1$ spots on the ground. You have to choose $k-1$ among them where you lay sticks, then you will lay your stones on the remaining spots. There are $\binom{n+k-1}{k-1}$ different arrangements, and they correspond to weak compositions of n from k parts. For example, if the stones are on the first $k-1$ spots, then the first $k-1$ summands are equal to zero, and the k-th summand equals n. □

3.6 Multisets

Imagine a bag with k balls numbered by $1, 2, \ldots, k$. We are taking out a ball, writing down its number, and then putting the ball back into the bag. This is done n times. From the product rule we know that k^n different sequences of numbers are possible. (One can see such a sequence as a map $f\colon \{1, \ldots, n\} \to \{1, \ldots, k\}$: here $f(i)$ is the number at the i-th place.) But what if we don't care for the order of the results, but only count how often each ball was taken? For example, the sequences $(2, 1, 4, 4)$ and $(4, 1, 4, 2)$ are considered as the same. How many different combinations of balls are possible?

One cannot proceed by the quotient rule, because the number of different orderings of a sequence with repetitions depends on how often the repetitions occur.

Definition 3.14. *A* multiset *is an unordered collection of elements with possible repetitions.*

Our question can be formulated as "how many n-multisets, with elements taken from $\{1, 2, \ldots, k\}$ are there?"

Theorem 3.15. *The number of different n-multisets with elements taken from $\{1, 2, \ldots, k\}$ is $\binom{n+k-1}{k-1}$.*

Proof. There is a bijection between n-multisets and weak compositions of n from k parts. Namely, to a weak composition $n = x_1 + \cdots + x_k$ of n we associate the multiset where 1 occurs x_1 times, 2 occurs x_2 times etc. The number of weak compositions was computed in Theorem 3.13. □

4 Multinomial coefficients

4.1 Words with repeating letters

Consider the following problem:

> *How many different words of length $n = k+l+m$ can be written with k letters a, l letters b, and m letters c?*

We have n places for the letters. First choose k places where to put the letters a. This can be done in $\binom{n}{k}$ different ways. Then from $n-k$ remaining places choose l places where to put the letters b. This can be done in $\binom{n-k}{l}$ different ways. Thus by the (general) product rule the number of different words is

$$\binom{n}{k}\binom{n-k}{l} = \frac{n!}{k!(n-k)!}\frac{(n-k)!}{l!(n-k-l)!} = \frac{n!}{k!l!m!}.$$

We might have started by choosing l places for the letters b, and then, say, choose m places for the letters c. Then we would compute the product

$$\binom{n}{l}\binom{n-l}{m}$$

which is the same.

The following notation is used:

$$\frac{n!}{k!l!m!} =: \binom{n}{k,l,m}.$$

Let us now consider a more general problem and solve it in a different way.

Theorem 4.1. *Let n balls of m different colors be given, with k_i balls of i-th color for all i (so that $k_1+\cdots+k_m = n$). Considering balls of the same color undistinguishable, the whole set of n balls can be arranged in a row in*

$$\binom{n}{k_1,\ldots,k_m} = \frac{n!}{k_1!\cdots k_m!}$$

different ways.

Proof. Make the balls of the same color distinguishable (by writing on them numbers, for example). Then all n balls can be arranged in $n!$ different ways. Now apply the quotient rule. There is a forgetful map from the set X of arrangements with balls of the same color distinguishable to the set Y of arrangements where balls of the same color are undistinguishable. What is the multiplicity of this map, that is the cardinalities of the preimages $|f^{-1}(y)|$? The balls of the i-th color can be permuted amongst themselves in $k_i!$ different ways. Permutations within every color can be performed independently, which gives us $k_1!\cdots k_m!$ elements in Y all of which correspond to the same element in X. Thus we have $|f^{-1}(y)| = k_1!\cdots k_m!$, and hence

$$|Y| = \frac{|X|}{k_1!\cdots k_m!} = \frac{n!}{k_1!\cdots k_m!}.$$

□

4.2 Multinomial theorem

The numbers

$$\binom{n}{k_1,\ldots,k_r}$$

are called *multinomial coefficients*. (It would be just logical to call $\binom{n}{k,l,m}$ a trinomial coefficient. Unfortunately, this term is sometimes used for the entries in an analog of Pascal triangle, where every number is equal to the sum of three numbers from the previous line, so a confusion may arise...) Similarly to the binomial coefficients, the multinomial coefficients appear in the expansion of the n-th power of a sum. Now the sum has not two, but m terms.

Theorem 4.2 (Multinomial theorem)**.**

$$(a_1+\cdots+a_m)^n=\sum_{\substack{k_1+\cdots+k_m=n\\ k_1,\ldots,k_m\geq 0}}\binom{n}{k_1,\ldots,k_m}a_1^{k_1}\cdots a_m^{k_m}$$

Proof. Multiply the brackets respecting the order of the factors. We obtain the sum of all words of length n from the letters $a_1,\ldots,a_m$. When collecting the like terms, we disregard the order of the letters, looking only at the number of occurences of every letter. The words with k_i letters a_i (for all i from 1 to m) become the terms $a_1^{k_1}\cdots a_m^{k_m}$. The coefficient at this term is the number of the words made of k_1 letters a_1,... k_m letters a_m, that is $\binom{n}{k_1,\ldots,k_m}$. □

4.3 Monotone paths in higher dimensions

Consider the cubical grid in $\mathbb{R}^m$. It is possible to visualize for $m=3$; for larger m one uses an abstract description. Mark the points $(0,\ldots,0)$ and $(k_1,\ldots,k_m)$ in this grid. A monotone path is one that moves along the grid lines and in the positive directions only.

Theorem 4.3. *The number of monotone paths from* $(0,\ldots,0)$ *to* $(k_1,\ldots,k_m)$ *is* $\binom{n}{k_1,\ldots,k_m}$.

Proof. Monotone paths can be encoded by words: write the letter a_i for a step along the i-th axis. Then a monotone path from $(0,\ldots,0)$ to $(k_1,\ldots,k_m)$ corresponds to a word consisting of k_i letters a_i for all i from 1 to m. □

5 Inclusion-exclusion formula

5.1 The formula

If $A\cap B\neq\varnothing$, then

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

Indeed, in $|A|+|B|$ we have counted all elements that belong to both A and B twice, see Figure 3, left. Subtracting $|A\cap B|$ makes our count correct, see Figure 3, right.

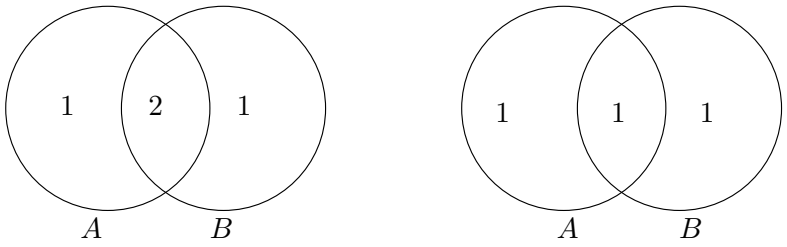

Figure 3: Counting the elements in the union of two sets.

Let us now count the elements in the union of three sets $A\cup B\cup C$. In the sum

$$|A|+|B|+|C|$$

every element is counted as many times as to how many sets it belongs, see Figure 4, left. Let us subtract the numbers of the elements in the pairwise intersections:

$$|A|+|B|+|C|-|A\cap B|-|B\cap C|-|A\cap C|.$$

Now every element that belongs to one or two sets is counted exactly once, but the elements in $A\cap B\cap C$ are not counted at all, see Figure 4, middle. So it remains to add the number of these elements to obtain the final formula:

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

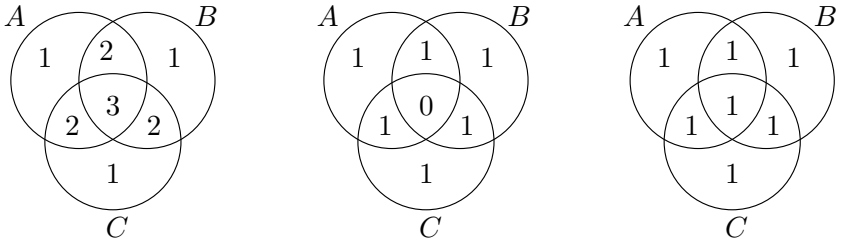

Figure 4: Counting the elements in the union of three sets.

What will the formula for the number of elements in the union of n sets look like? The formulas for $n=2$ and $n=3$ suggest that this will be the sum of the cardinalities of all sets minus the sum of the cardinalities of pairwise intersections plus all the triple intersections minus all the quadruple intersections, and so on. This conjecture is true, and we will prove it now.

Theorem 5.1 (Inclusion-exclusion formula)**.**

$$|A_1 \cup \cdots \cup A_n| = \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \cdots \\ \cdots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|$$

Proof. We will show that every element of the union is counted exactly once on the right hand side of the formula.

(Can make a warm-up for the elements in the intersection of all sets.) Take an element that belongs to exactly k of the sets $A_1, \ldots, A_n$. In the first sum on the right hand side it is counted k times, in the second sum $\binom{k}{2}$ times, and so on. In total, this element is counted with the multiplicity

$$\binom{k}{1} - \binom{k}{2} + \cdots + (-1)^{k-1} \binom{k}{k}.$$

Due to Theorem 3.8, this sum is equal to $\binom{k}{0} = 1$. □

Remark 5.2. One can give an exact meaning to the words "how many times was an element counted". Instead of the intersections $A_i \cap A_j$ etc. consider their indicator functions $\mathbf{1}_{A_i \cap A_j}$. Instead of summing the cardinalities, sum these functions. The diagrams on Figures 3 and 4 show the values of certain sums of indicator functions. The argument in the proof of Theorem 5.1 shows that

$$\mathbf{1}_{A_1 \cup \cdots \cup A_n} = \sum_{i=1}^{n} \mathbf{1}_{A_i} - \sum_{1 \le i < j \le n} \mathbf{1}_{A_i \cap A_j} + \sum_{1 \le i < j < k \le n} \mathbf{1}_{A_i \cap A_j \cap A_k} - \cdots \\ \cdots + (-1)^{n-1} \mathbf{1}_{A_1 \cap \cdots \cap A_n}$$

by comparing the values of the functions on the left and on the right at every point. The formula for the number of elements is obtained by taking the "integrals" of both sides, that is replacing each function f by the number $\sum_{x \in A_1 \cup \cdots \cup A_n} f(x)$.

5.2 De Montmort problem, or counting the derangements

The problem was originally posed by Pierre Rémond de Montmort in 1708, and was solved by him and, independently, Nicholas Bernoulli.

De Montmort stated it in terms of a game of cards, later it became popular in the following formulation:

> *The guests leaving a party are taking their hats in the garderobe. In the darkness they cannot tell the hats one from the other, so everybody takes a hat by chance. What is the probability that nobody will get his own hat?*

Here is a formal description of the problem. Consider a bijection

$$f\colon \{1,2,\dots,n\} \to \{1,2,\dots,n\}.$$

Such a bijection is called a permutation. An element $x \in \{1,2,\dots,n\}$ is called a *fixed point* of f if $f(x) = x$. A permutation without fixed points is sometimes called a *derangement*. The probability to be computed is equal to

$$\frac{\#\text{derangements}}{\#\text{permutations}}.$$

The number of permutations is known: it is $n!$. Thus we have to count the number of derangements.

Theorem 5.3. *The number of all derangements of n elements is equal to*

$$\sum_{k=0}^{n}(-1)^k\binom{n}{k}(n-k)! = \sum_{k=0}^{n}(-1)^k\frac{n!}{k!}.$$

Proof. Let A_i be the set of permutations f such that $f(i) = i$. Then $A_1 \cup \dots \cup A_n$ is the set of all non-derangements, permutations with at least one fixed points. Calculate its cardinality by the inclusion-exclusion formula. The intersection $|A_{i_1} \cap \dots \cap A_{i_k}|$ consists of all permutations satisfying the conditions

$$f(i_1) = i_1, \dots, f(i_k) = i_k.$$

The number of such permutations is $(n-k)!$ (to determine f, it remains to map bijectively an $(n-k)$-element set to an $(n-k)$-element set). Thus we have

$$|A_{i_1} \cap \dots \cap A_{i_k}| = (n-k)!$$

for any choice of k sets out of $A_1, \dots, A_n$. Since the number of such choices is $\binom{n}{k}$, we have

$$|A_1 \cup \dots \cup A_n| = n(n-1)! - \binom{n}{2}(n-2)! + \dots + (-1)^{n-1}\binom{n}{n}0!.$$

To count the derangements, we subtract the number of non-derangements from the number of all permutations, which leads to the formula in the theorem. □

Now we can compute the probability that no guest gets his hat:

$$\frac{\#\text{derangements}}{\#\text{permutations}} = \frac{\sum_{k=0}^{n}(-1)^k\frac{n!}{k!}}{n!} = \sum_{k=0}^{n}\frac{(-1)^k}{k!}.$$

Recall that

$$\sum_{k=0}^{\infty}\frac{x^k}{k!} = e^x.$$

It follows that the above probability converges to $\frac{1}{e} \approx 0.37$.

5.3 Euler's totient function

Two positive integers whose greatest common divisor equals 1 are called relatively prime. Denote by $\varphi(n)$ the number of positive integers $\leq n$ which are relatively prime to n:

$$\varphi(n) = \#\{k \in \{1, 2, \ldots, n\} \mid \gcd(k, n) = 1\}.$$

For example, for $n = 6$ only the numbers 1 and 5 among $1, 2, 3, 4, 5, 6$ are relatively prime to 6, so that we have $\varphi(6) = 2$. In $n = p$ is prime, then all numbers $1, \ldots, p-1$ are relatively prime to p, so that $\varphi(p) = p - 1$.

Theorem 5.4. *Let $n = p_1^{\alpha_1} \cdots p_m^{\alpha_m}$ be a prime factorization of n, that is $p_1, \ldots, p_m$ are distinct prime numbers, and $\alpha_1, \ldots, \alpha_m$ are positive integers. Then*

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_m}\right) = p_1^{\alpha_1 - 1}(p_1 - 1) \cdots p_m^{\alpha_m - 1}(p_m - 1).$$

Proof. Let A_i be the set of numbers among $1, 2, \ldots, n$ divisible by p_i. Then $A_1 \cup \cdots \cup A_m = \{k \in \{1, 2, \ldots, n\} \mid \gcd(k, n) > 1\}$ and we have

$$\varphi(n) = n - |A_1 \cup \cdots \cup A_m|.$$

To compute $|A_1 \cup \cdots \cup A_m|$, use the inclusion-exclusion formula. We have

$$A_i = \{p_i, 2p_i, \ldots, \frac{n}{p_i} p_i\},$$

thus $|A_i| = \frac{n}{p_i}$. Similarly, we have

$$|A_{i_1} \cap \cdots \cap A_{i_k}| = \frac{n}{p_{i_1} \cdots p_{i_k}}.$$

By the inclusion-exclusion formula,

$$|A_1 \cup \cdots \cup A_m| = \sum_{i=1}^{m} \frac{n}{p_i} - \sum_{i<j} \frac{n}{p_i p_j} + \cdots + (-1)^{m-1} \frac{n}{p_1 \cdots p_m}.$$

Thus we have

$$\begin{aligned} \varphi(n) &= n - |A_1 \cup \cdots \cup A_m| \\ &= n\left(1 - \sum_{i=1}^{m} \frac{1}{p_i} + \cdots + (-1)^m \frac{1}{p_1 \cdots p_m}\right) \\ &= n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_m}\right). \end{aligned}$$

□

Chapter II

Graph theory

1 Basic notions

1.1 Types of graphs

Intuitively, a graph is a set of points, some of which are joined by lines. The points are called *vertices* of the graph, the lines are called *edges* of the graph. Quite often, we represent a graph by drawing it in the plane. For some graphs, one can draw the edges in such a way that they do not intersect. But for other graphs self-intersections are inavoidable. In order not to confuse the intersection points with the "true" vertices, we draw the vertices as small disks. See Figure 1 for two drawings of the same graph: one with, the other without self-intersections.

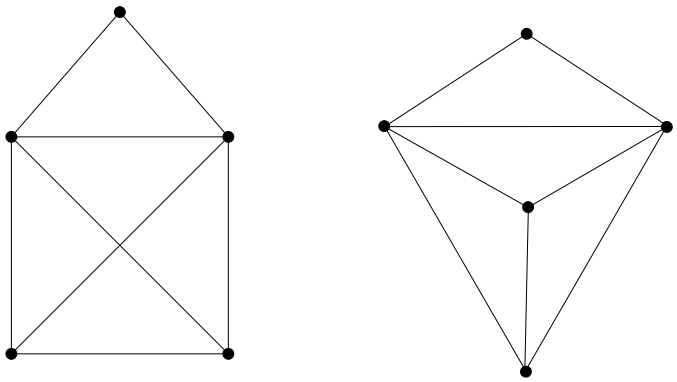

Figure 1: Two drawings of the same graph.

Let us give a formal definition.

Definition 1.1. *A graph G is a pair (V, E), where V is some set and E is a collection of two-element subsets (unordered pairs of elements) of V. The set V is called the vertex set of G, the set E is called the edge set of G.*

Examples of graphs: transport networks, neural networks, "friendship" graphs (social networks). Usually the set V of vertices is assumed to be finite. However it can be quite large (and it is in some of the above examples).

In some situations one is lead to consider graphs with *loops* (lines joining a vertex to itself) and *multiple edges* (several lines between the same pair of vertices), see Figure 2, left. In some other situations one wants to draw arrows instead of lines, see Figure 2, right. Graphs with oriented edges are called *directed graphs*. Another type of graphs are *weighted graphs*: here to every edge a number is assigned. When we say "a graph", we mean it in the sense of definition 1.1: an undirected graph without loops and multiple edges and without assignment of weights.

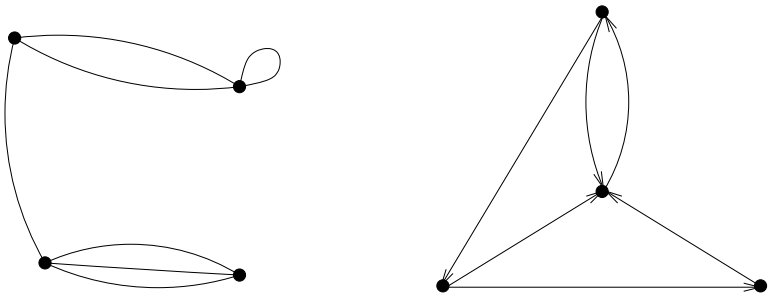

Figure 2: A graph with loops and multiple edges; a directed graph.

1.2 Some graphs known by names

Below are some important families of graphs.

- The *complete graph* K_n is a graph with n vertices, with each pair of vertices joined by an edge.
- The *cycle graph* C_n is a graph with n vertices $v_1, \cdots, v_n$ and the edges $\{v_i, v_{i+1}\}$ for $i = 1, \ldots, n-1$ and $\{v_1, v_n\}$.
- The *path graph* P_n is a graph with n vertices $v_1, \cdots, v_n$ and the edges $\{v_i, v_{i+1}\}$ for $i = 1, \ldots, n-1$.

Definition 1.2. *A graph $G = (V, E)$ is called* bipartite *if its vertex set can be partitioned in two sets V_1 and V_2 such that no two vertices from V_1 are joined by an edge and no two vertices from V_2 are joined by an edge.*

- The *complete bipartite graph* $K_{m,n}$ is a bipartite graph with the vertex set $V_1 \cup V_2$, where $|V_1| = m$, $|V_2| = n$, and every vertex from V_1 is joined with every vertex from V_2. The graph $K_{3,3}$ is shown in Figure 3, left.

1.3 Isomorphic graphs and subgraphs

Definition 1.3. *Two graphs* (V, E) *and* (V', E') *are called* isomorphic *if there is a bijection* $f\colon V \to V'$ *between their vertex sets such that*

$$\{x, y\} \in E \text{ if and only if } \{f(x), f(y)\} \in E'.$$

Example 1.4. The graphs in Figure 3 are isomorphic (check this!). Thus the graph on the right is also $K_{3,3}$.

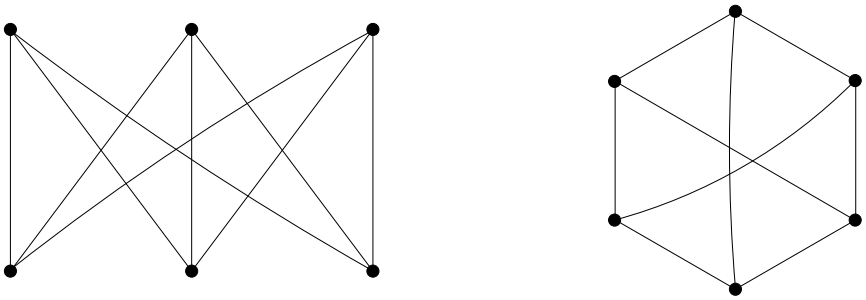

Figure 3: The complete bipartite graph $K_{3,3}$.

Remark 1.5. Isomorphic graphs obviously have the same number of vertices and the same number of edges. The converse is not true: one can find two graphs with the same number of vertices and the same number of edges which are not isomorphic.

Exercise 1.1. The graph shown in Figure 4 is called *Petersen graph*. Show that it is isomorphic to the following graph:

$$V = \{\text{all two-element subsets of } \{1, 2, 3, 4, 5\}\},$$
$$E = \{\{A, B\} \mid A \cap B = \varnothing\}$$

Definition 1.6. *A graph* $G' = (V', E')$ *is called a* subgraph *of graph* $G = (V, E)$ *if* $V' \subset V$ *and* $E' \subset E$.

Note that we cannot take any pair of subsets $V' \subset V, E' \subset E$: if $\{v, w\} \in E'$, then we must have $v, w \in V'$.

An *n-cycle* in a graph is a subgraph isomorphic to C_n. Bipartite graphs contain no 3-cycles, and more generally no cycles of odd length.

Exercise 1.2. Prove the converse: if a graph contains no cycles of odd length, then it is bipartite.

A *Hamiltonian cycle* in a graph is a cycle that contains all of its vertices. It provides a closed path visiting all of its vertices exactly once. A graph is called *Hamiltonian* if it has a Hamiltonian cycle. The graph $K_{3,3}$ is Hamiltonian, the Petersen graph is not.

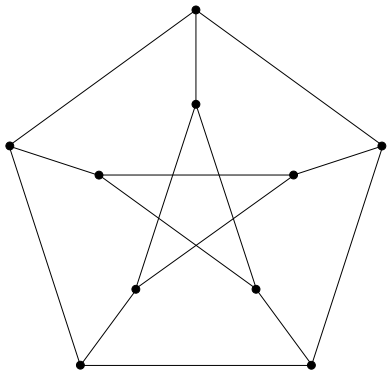

Figure 4: The Petersen graph.

1.4 Incidence and adjacency

An edge is said to be *incident* with a vertex if it contains this vertex. In other words, the edge $\{v, w\}$ is incident with the vertices v and w.

The *degree* of a vertex v, denoted $\deg v$, is the number of edges incident with v. A vertex of degree zero, that is without incident edges, is called an *isolated* vertex.

In the complete graph K_n all vertices have degree $n - 1$. In the cycle C_n, all vertices have degree 2.

Definition 1.7. *A graph is called* k-regular *if all of its vertices have degree* k*.*

Theorem 1.8 (Handshake lemma)**.** *In every graph, the sum of all vertex degrees is twice the number of edges:*

$$\sum_{v \in V} \deg v = 2|E|.$$

Proof. Count the vertex-edge incidences in two ways. From the vertices viewpoint, every vertex v is incident to $\deg v$ many edges. Thus the number of incidences is the sum of the degrees of all vertices. From the edges viewpoint, every edge is incident to two vertices. Thus the number of incidences is twice the number of edges. The theorem follows. □

The name "handshake lemma" suggests a reformulation of the above argument. In a group of people, several handshakes take place. How to count the number of handshakes? One way to do it is to ask every person how many handshakes it made and to add all these numbers. Since every handshake is counted twice, we have to divide the result by two.

Corollary 1.9. *In every graph, the number of vertices of odd degree is even.*

Indeed, the sum of all vertex degrees must be even by Theorem 1.8.

Two vertices joined by an edge are called *adjacent*.

Definition 1.10. *Let $G = (V, E)$ be a graph on n vertices. Denote the vertices by $v_1, \dots, v_n$ (in an arbitrary order). The* adjacency matrix *of G (with respect to a given ordering of vertices) is an $n \times n$ matrix A with the following entries*

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{if } \{v_i, v_j\} \notin E. \end{cases}$$

Recall the matrix multiplication: for $n \times n$ matrices A and B we define their product $C = AB$ as

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}.$$

Theorem 1.11. *Let G be a graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and let A be its adjacency matrix. Let $a_{ij}^{(k)}$ denote the element of the matrix A^k at the position (i, j). Then $a_{ij}^{(k)}$ is the number of walks of length exactly k from the vertex v_i to the vertex v_j in the graph G.*

Proof. Exercise. □

The reader is invited to define analogs of the adjacency matrix for directed and weighted graphs.

1.5 Connectivity and components

Definition 1.12. *A* walk *in a graph $G = (V, E)$ is a sequence*

$$(v_0, e_1, v_1, e_2, \dots, e_k, v_k),$$

where $v_i \in V$, and for each $i = 1, \dots, k$ we have $e_i = \{v_{i-1}, v_i\} \in E$.

Remark 1.13. A walk can revisit vertices and go along an edge several times. That is, one can have $v_i = v_j$ or $e_i = e_j$ for $i \neq j$.

Definition 1.14. *A* path *in a graph is a walk with distinct vertices: $v_i \neq v_j$ for $i \neq j$.*

In other words, a *path* in a graph G is a subgraph of G isomorphic to P_n for some n.

Definition 1.15. *A graph G is called* connected *if for any two vertices $x, y \in V(G)$ there is a path between x and y.*

This is the same as to say that there is a walk between x and y: every path is a walk, and from every walk one can remove cycles so that it becomes a path.

Definition 1.16. *A* component *of a graph G is a maximal connected subgraph of G.*

1.6 Eulerian graphs

Euler's "Seven bridges of Königsberg" problem.

Informally speaking, we are asking what graphs can be drawn without lifting the pencil from paper (and drawing every edge only once). You can try to draw the graph on Figure 1 in this way.

Definition 1.17. *A* Euler walk *on a graph G is a walk that takes each edge of G exactly once. A* Eulerian circuit *is a closed walk that takes each edge exactly once.*

In other words, a Eulerian circuit or Eulerian walk is an ordering of the edges of G such that two consecutive edges share a vertex.

A graph is called *Eulerian* if it has a Eulerian circuit and *semi-Eulerian* if it has a Eulerian walk.

Theorem 1.18. *A graph is Eulerian if and only if all of its vertex degrees are even and all vertices of positive degree belong to the same connected component. A graph is semi-Eulerian if and only if it has exactly two vertices of odd degree and all vertices of positive degree belong to the same connected component.*

The somewhat awkward condition "all vertices of positive degree belong to the same connected component" is equivalent to "the graph becomes connected after deleting all isolated vertices". Another way to deal with the isolated vertices is to define the notion of a Eulerian circuit/walk differently: it must visit all vertices. Then a graph is (semi-)Eulerian if and only if it is connected and the degree evenness condition is satisfied.

Proof. The "only if" direction. Assume that the graph has a Eulerian walk or a Eulerian circuit. Then there is a walk between any two vertices of positive degree: one can use a piece of the walk or circuit to get from one to the other. In order to prove that the degrees of all vertices (except two in the semi-Eulerian case) are even, orient the edges in the direction of the walk: orient e_i from v_{i-1} to v_i. We obtain a directed graph. In a directed graph, every vertex v has the *in-degree* $\deg_+ v$ and the *out-degree* $\deg_- v$: the number of edges entering v and the number of edges leaving v. Clearly, $\deg v = \deg_+ v + \deg_- v$. On the other hand, a Eulerian circuit enters and leaves every vertex equal number of times: $\deg_+ v = \deg_- v$. This implies that $\deg v$ is even for every vertex of a Eulerian graph.

In a semi-Eulerian graph we have $\deg_+ v = \deg_- v$ for every intermediate vertex of the walk, but

$$\deg_+ v_0 = \deg_- v_0 - 1, \quad \deg_+ v_m = \deg_- v_m + 1$$

for the initial and the final vertices of the walk, respectively. It follows that the degrees of all vertices except v_0 and v_m are even.

The "if" direction. First, consider the case when all vertex degrees are even. Start to walk from any vertex without going along any edge twice. At some point we must stop because all edges incident to the current vertex are used. This can only happen at the initial vertex of our walk because if you stop at a different vertex, then its in-degree will be one bigger than the out-degree, which contradicts the assumption that all vertex degrees are even. Thus we obtain a circuit (which does not necessarily cover all edges). Remove this circuit from the graph. We obtain a possibly disconnected graph where all vertex degrees are even. Repeat the procedure until the edge set of our graph will be partitioned into circuits. Then start to merge the circuits: if two circuits have a common vertex, then they can be replaced by a single circuit. If some circuit has no common vertices with the other circuits, then its vertices and edges form a connected component of the graph, and we will have at least two non-trivial connected components. Thus all circuits can be merged to a circuit covering all edges of the graph.

If the graph has two vertices of odd degree, then start our first walk from one of these vertices. This walk will necessarily stop at the other odd vertex (again, by consideration of the in- and out-degrees). Removing this walk from the graph yields a graph with all vertices of even degree, and we proceed as in the previous case. □

Try this algorithm on the graph on Figure 5.

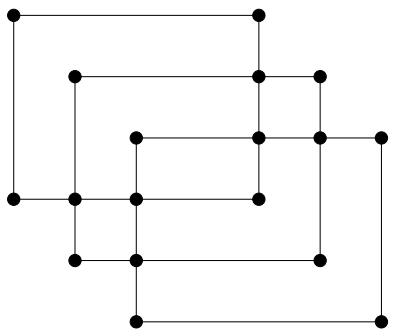

Figure 5: Find a Eulerian circuit in this graph.

Note that for graphs with two vertices of odd degree we have proved a bit more: every Eulerian circuit starts in one of the odd vertices and ends in the other one.

2 Trees

2.1 Basics

Recall that a cycle in a graph is a subgraph isomorphic to C_n for some n. A graph without cycles is called *acyclic*.

Definition 2.1. *A* tree *is a connected acyclic graph.*

Examples are shown in Figure 6.

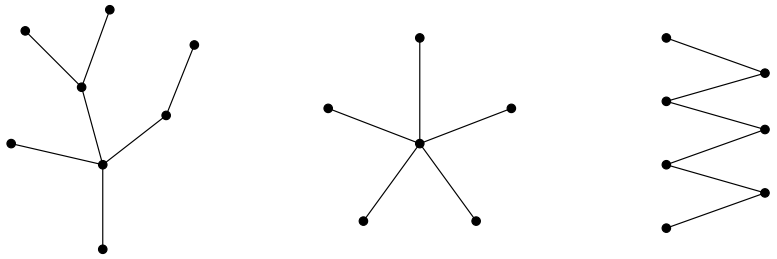

Figure 6: Some trees.

A disconnected acyclic graph is called a *forest*. Every connected component of a forest is a tree. Indeed, a component is connected by definition; it has no cycles because, clearly, every subgraph of an acyclic graph is acyclic.

Theorem 2.2. *In a tree, any two vertices are connected by exactly one path.*

Proof. Take any two vertices of a tree. Since a tree is connected, there is at least one path between these two vertices. If there is more than one path, then this implies the existence of a cycle. Namely, take the first vertex where the two paths diverge and the first vertex where they meet again; the union of the segments of our paths between these vertices will be a cycle. (We are not working out the details here.) This contradicts the assumption that our graph is a tree, thus there cannot be more than one path between two vertices. □

A *rooted* tree is a tree T with a specified vertex x, called the *root* of T. The edges of a tree can be equipped with orientation so that for every vertex v the (unique) path from x to v always follows the directions of edges. (Again, this looks intuitively clear, but requires a formal proof.) See Figure 7 for an example.

2.2 Leaves

Definition 2.3. *A vertex of degree* 1 *is called a* leaf.

Find leaves of trees in Figure 6.

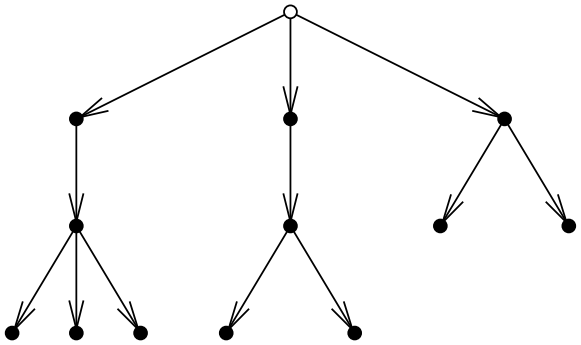

Figure 7: A canonically oriented rooted tree.

Lemma 2.4. *Every tree has at least two leaves.*

Proof. Take a path $P \subset T$ of maximum length. (If there are several paths of maximum length, take one of them.) I claim that its endpoints v and w are leaves. Indeed, assume $\deg v > 1$. Then there is an edge of T incident with v and not belonging to P. If the other end of this edge is also a vertex of P, then T contains a cycle, which contradicts to it being a tree. If the other end is not in P, then adding it to P we obtain a longer path. This contradicts the choice of P. Thus the assumption $\deg v > 1$ was false, v is a leaf, and w is a leaf as well. □

Definition 2.5. *Let G be a graph and v a vertex of G. Denote by $G - v$ the graph obtained by deleting the vertex v and all edges incident to it. This operation is called* vertex deletion.

Lemma 2.6. *Let T be a tree and v a leaf of T. Then $T - v$ is also a tree.*

Proof. The graph $T - v$ is acyclic, because it is a subgraph of an acyclic graph. It remains to prove that $T - v$ is connected. Let x and y be two vertices of T different from v. There is a path in T connecting x and y. This path does not contain v, because otherwise the degree of v would be at least 2. Thus it can be considered as a path in $T - v$, so x and y are connected within $T - v$. □

2.3 The number of edges in a tree

Lemma 2.6 allows to use induction when proving theorems about trees.

Theorem 2.7. *If a graph $T = (V, E)$ is a tree, then $|E| = |V| - 1$.*

Proof. Let $|V| = n$. Use induction on n. For $n = 1$, there is only one tree with one vertex. It has no edges, which proves the induction base.

Now let us prove the induction step: if every tree with n vertices has $n-1$ edges, then every tree with $n+1$ vertices has n edges. Take any tree T with $n+1$ vertices. By Lemma 2.4 it has a leaf v. By Lemma 2.6 the graph $T-v$ is also a tree. The tree $T-v$ has n vertices, therefore by the induction assumption it has $n-1$ edges. But then T has n edges, and the induction step is proved. □

Corollary 2.8. *A forest with n vertices and k components has $n-k$ edges.*

Proof. Let the components have $n_1, \ldots, n_k$ vertices. By the above theorem, they have $n_1-1, \ldots, n_k-1$ edges, respectively. By summing up the number of edges we obtain the desired result. □

2.4 Spanning trees

Definition 2.9. *Let $G = (V, E)$ be a graph. A subgraph of a graph G is called a* spanning tree *if it is a tree with the same vertex set as G.*

Figure 8 shows examples of spanning trees.

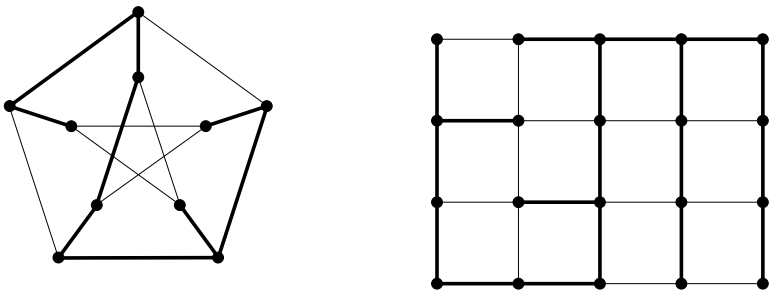

Figure 8: Examples of spanning trees.

Theorem 2.10. *Every connected graph has a spanning tree.*

We will find a spanning tree by deleting edges from the graph one by one while taking care that the graph remains connected. For any graph $G = (V, E)$ and any its edge $e \in E$ denote by $G - e$ the graph $(V, E \setminus \{e\})$. (Note that we are not removing any vertices, even if after deletion of e an isolated vertex appears.) This is the operation of *edge deletion*.

Proof. Let G be a connected graph. If G contains a cycle C, then let e be any edge of C. I claim that the graph $G - e$ is connected. Indeed, let $v, w \in V$ be any two vertices of G. Since G is connected, there is a path in G between v and w. If this path never uses the edge e, then this is also a path in $G - e$. If it does use e, then instead going on e, take a detour via

the path $C - e$. This produces a walk in $G - e$ from v to w. A walk can be transformed into a path by removing cycles.

Thus $G - e$ is connected. If it is acyclic, then it is a spanning tree. Otherwise repeat the operation: take another cycle and remove an edge from it etc. until we arrive at an acyclic connected subgraph with the same vertex set as G. □

The following theorem is a strengthening of Theorem 2.7.

Theorem 2.11. *Let $G = (V, E)$ be a graph. If $|E| > |V| - 1$, then G contains a cycle. If $|E| < |V| - 1$, then G is not connected.*

Proof. Both statements are proved by contraposition. That is, we prove the following: if G contains no cycle, then $|E| \leq |V| - 1$; if G is connected, then $|E| \geq |V| - 1$.

Let G be acyclic. Then by Corollary 2.8 it has $|V| - k \leq |V| - 1$ edges, where k is the number of components of G.

Let G be connected. By Theorem 2.10, G has a spanning tree which, by Theorem 2.7, has $|V| - 1$ edges. Thus G has at least $|V| - 1$ edges. □

2.5 The number of spanning trees

The number of spanning trees of a given graph is an interesting combinatorial problem.

Theorem 2.12 (Borchardt, Cayley)**.** *The complete graph on n vertices has n^{n-2} spanning trees.*

For example, K_3 has 3 spanning trees (obtained by deleting one arbitrary edge), K_4 has 16, and K_5 already 125 different spanning trees.

In order to count the spanning trees in an arbitrary graph, the following matrix is needed.

Definition 2.13. *The* Laplacian matrix *of a graph G is*

$$L = D - A,$$

where D is the degree matrix, that is a diagonal matrix with $d_{ii} = \deg v_i$, and A is the adjacency matrix of G.

The matrix L has zero determinant (because the vector $(1, 1, \ldots, 1)$ belongs to its kernel). Moreover, its rank equals $n - k$, where $n = |V|$ and k is the number of connected components of G. In particular, if G is connected, then L contains an $(n-1) \times (n-1)$ minor with non-zero determinant. In fact, $\det L = 0$ implies that all cofactors $(-1)^{i+j} \det L_{ij}$ are equal (you can try to prove this). This common value is the number of the spanning trees.

Theorem 2.14 (Kirchhoff)**.** *Let G be a connected graph, and L be its Laplacian matrix. Then the number of spanning trees of G is equal to each of the following numbers.*

- $(-1)^{i+j} \det L_{ij}$*, where L_{ij} is the matrix obtained by removing the i-th row and the j-th column from L;*
- $\frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$*, where λ_i are the non-zero eigenvalues of L.*

In particular, for the complete graph we have the determinant of the following $(n-1)\times(n-1)$ matrix:

$$\det \begin{pmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & n-1 \end{pmatrix}.$$

Cayley's theorem indicates that it is equal to n^{n-2}. Of course, it is easier (but still not straightforward) to prove this equality without invoking two difficult theorems above.

2.6 Minimum spanning tree: Kruskal's algorithm

Every spanning tree in a connected graph on n vertices has $n-1$ edges. However, if edges are equipped with weights, then we can speak about the spanning tree of minimum total weight. For example, if the weights of edges are the costs of building railroads between towns, then the minimum spanning tree is the cheapest railroad network connecting all towns.

Definition 2.15. *A* weighted graph *is a pair (G,ω), where G is a usual graph, and*

$$\omega\colon E(G)\to\mathbb{R}$$

is a map. The number $\omega(e)$ is called the weight *of the edge e.*

Definition 2.16. *A* minimum spanning tree *of a weighted connected graph (G,ω) is a spanning tree (V,E') of G such that the sum $\sum_{e\in E'}\omega(e)$ has the minimum possible value among all spanning trees of G.*

A minimum spanning tree is not necessarily unique: if all weights are the same, then all spanning trees have the same total weight.

Theorem 2.17 (Kruskal's algorithm)**.** *Order the edges of a weighted connected graph according to their weights:*

$$(e_1,\dots,e_m) \quad \textit{such that} \quad \omega(e_1)\le\cdots\le\omega(e_m).$$

Going through this list, mark an edge if it does not create a cycle together with the previously marked edges. More exactly, put $G_0 = (V, \varnothing)$: the graph with isolated vertices only. If G_i is already defined, then put

$$G_{i+1} = \begin{cases} G_i + e_{i+1} & \text{if } G_i + e_{i+1} \text{ has no cycles} \\ G_i & \text{otherwise.} \end{cases}$$

The output of the algorithm is the graph G_m.

This is a greedy algorithm: at each step we do what seems to us the best, we take the lightest edge. Of course, a "locally optimal" procedure does not always lead to a "globally optimal" result. We must prove that Kruskal's algorithm outputs a minimum spanning tree. But first of all, we must show that the output is a tree at all.

Proof that the output is a tree. By construction, every graph G_i, and in particular G_m, is acyclic. Let us show that G_m is connected. Assume the converse. Take any two connected components of G_m. Since (V, E) is connected, there is an edge $e_i \in E$ joining two vertices v and w from these components. This e_i does not belong to G_m. Thus at the i-th step of the algorithm, when we were deciding to take e_i or not, this edge created a cycle with edges of G_{i-1}. This means that in G_{i-1} (and hence in G_m) there is a path from v to w. But then v and w are in the same connected component of G_m, which is a contradiction. □

Proof that the tree is minimal. We prove by induction on i that every graph G_i is contained in some minimum spanning tree. For $i = m$ this will tell us that G_m is a minimum spanning tree.

As the induction base take $i = 0$. Here the assertion is trivially true, because there exists at least one minimum spanning tree.

For the induction step, assume that G_i is a subgraph of a minimum spanning tree T_i. Consider the next edge e_{i+1}. If $G_{i+1} = G_i$ (which happens if e_{i+1} creates a cycle), then G_{i+1} is a subgraph of T_i, and we are good. If $G_{i+1} = G_i + e_{i+1}$ and e_{i+1} is an edge of T_i, then G_{i+1} is a subgraph of T_i as well.

The only non-trivial case is when $G_{i+1} = G_i + e_{i+1}$ and e_{i+1} is not an edge of T_i. Then the graph $T_i + e_{i+1}$ contains a cycle, which is formed by the edge e_{i+1} and the path P in T_i connecting the endpoints of e_{i+1}. There is an edge f of P that does not belong to G_i: if this is not the case, then G_{i+1} contains a cycle. The graph $T_i + e_{i+1} - f$ is a tree: it is connected and has $|V| - 1$ edges. We claim that this tree is also a minimum tree. For this one has to show that $\omega(f) = \omega(e_{i+1})$.

Assume that $\omega(f) < \omega(e_{i+1})$, then the edge f appears on the list of all edges earlier than e_{i+1}. But why did not we add f to the graph that we are constructing? This can only be because f would create a cycle. But then f

also creates a cycle when added to G_i. Since $G_i + f \subset T_i$, it follows that T_i contains a cycle, which is a contradiction.

Thus $\omega(f) \geq \omega(e_{i+1})$. It is not possible that $\omega(f) > \omega(e_{i+1})$, because then the tree $T_i + e_{i+1} - f$ has smaller weight than T_i, which by assumption is a minimum spanning tree. Thus we have $\omega(f) = \omega(e_{i+1})$. But then G_{i+1} is a subgraph of the minimum spanning tree $T_i + e_{i+1} - f$, so the induction step is proved. □

Remark 2.18. There are other algorithms that find a minimum spanning tree, for example Prim's algorithm. Closely related to Kruskal's algorithm is the *reverse-delete algorithm*. Here we order the edges in the nonincreasing order of weights, go through the list and delete an edge if this does not disconnect the graph.

Remark 2.19. Another problem for a connected weighted graph is to find a minimum path between two given vertices. This can be done with Dijkstra's algorithm. Note that a minimum spanning tree contains a path between any pair of vertices, but this path is not necessarily minimal.

3 Planar graphs

3.1 Basics

Definition 3.1. *A graph is called* planar *if it can be drawn in the plane in such a way that its edges do not cross each other. A drawing with pairwise non-crossing edges is called a* planar embedding *of the graph or, for short, a* plane graph.

A graph can have "different" embeddings in the plane, see Figure 9.

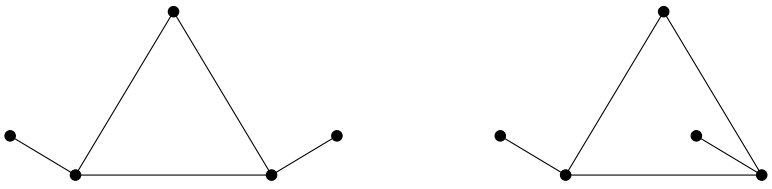

Figure 9: Two different embeddings of the same graph.

Two embeddings are called *non-isotopic* if one cannot be deformed into the other continuously while remaining an embedding during the deformation.

Remark 3.2. Let us stress the difference between "planar graph" and "plane graph". A planar graph is an abstract graph that *can be* drawn in the plane. A plane graph is a graph that *is* already drawn in the plane.

The reason for distinguishing between plane and planar is that different drawings can be possible. As the above example shows there are different (non-isotopic) plane graphs which represent the same (isomorphic) planar graphs.

Theorem 3.3. *The graph K_5 is not planar.*

Sketch of proof. Let $v_1, \dots, v_5$ be the vertices of K_5. We denote by the same letters their images in the plane. The vertices v_1, v_2, v_3, v_4 and the edges between them form a planar embedding of K_4. Up to relabeling of vertices and isotopy there is a unique embedding of K_4 in the plane, see Figure 10, left. The graph K_4 separates the plane into four regions. The fifth vertex v_5 must lie in one of these regions. In whatever region it lies, it will be separated from one of the first four vertices. For example, if it lies in the outer region, then the edge v_5v_1 must intersect the contour $v_2v_3v_4$ at least once, see Figure 10, right. □

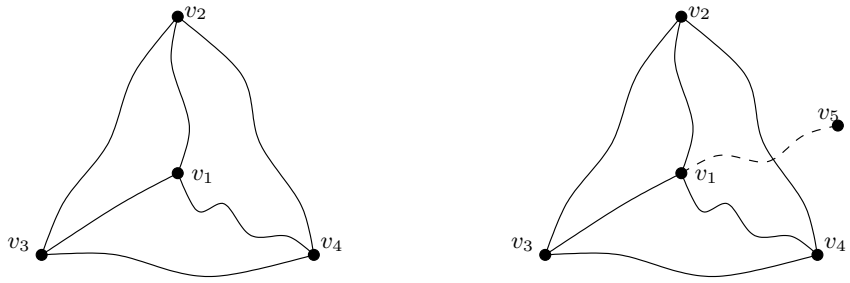

Figure 10: Proof of the non-planarity of K_5.

The above is only a sketch of the proof, because the assertions "there is a unique embedding of K_4" and "the edge must intersect the contour" require formal proofs. They follow from the Jordan curve theorem: any embedded closed curve in the plane separates the plane in two connected components. This might seem trivial but you should take into account that there are curves which are neither smooth nor polygonal (maybe you have heard about fractals).

On the other hand, there is the following theorem (whose proof relies on the Jordan curve theorem).

Theorem 3.4 (Fáry)**.** *If a graph can be drawn in the plane, then it can also be drawn with straight edges.*

Thus we may always imagine our graphs being drawn with straight edges and rely on our geometric intuition.

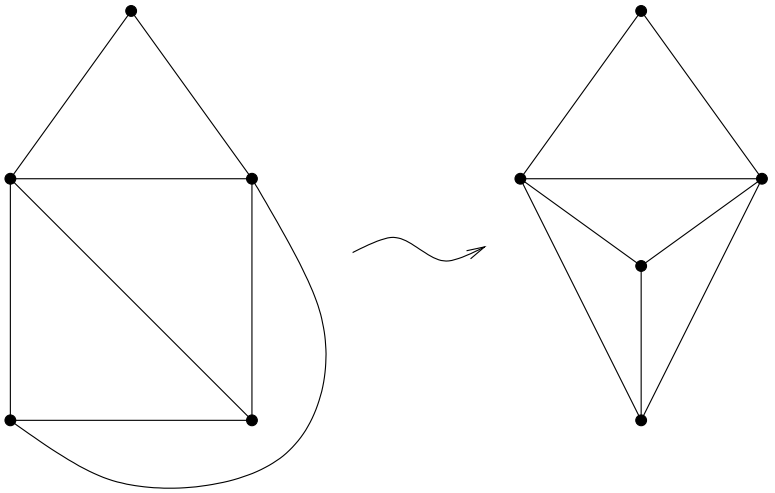

Figure 11: Any planar graph can be drawn with straight edges.

3.2 Euler's formula

A plane graph separates the plane into regions, called *faces*. Every embedding has one unbounded face, called the *outer face*.

The graphs on Figure 9 have two faces each. If the graph has n vertices and no edges, then its embedding has only one face: the plane punctured at n points.

A face is called incident with an edge if the edge belongs to the boundary of the face.

Definition 3.5. *The* degree *of a face of a plane graph is the number of edges incident with it. If on both sides of an edge lies the same face, then this edge is counted twice when calculating the degree of the face.*

For example, on Figure 9, left, the outer face has degree 7 and the inner face has degree 3. On Figure 9, right, both the inner and the outer face have degrees 5.

Theorem 3.6. *For every plane graph the sum of the degrees of its faces is twice the number of edges:*

$$\sum_{f \in F} \deg f = 2|E|.$$

(Here F denotes the set of all faces.)

Proof. Count the face-edge incidences in two ways: from the viewpoint of the faces and from the viewpoint of the edges. □

Observe that in the previous theorem the graph may be disconnected; it may have no edges at all.

Theorem 3.7 (Euler)**.** *Let $G = (V, E)$ be a connected plane graph. Denote by F the set of its faces. Then we have*

$$|V| - |E| + |F| = 2.$$

In particular, the number of faces does not depend on the choice of an embedding.

Example 3.8. The graphs of platonic solids (and more generally, of all convex polytopes) are planar. Thus the Euler formula holds for the numbers of vertices, edges, and faces of any convex polytope. For example, the cube has 8 vertices, 12 edges, and 6 faces, and we have indeed $8 - 12 + 6 = 2$.

Proof. Induction on the number of edges.

Let $|E| = 0$. The only connected graph without edges is the graph with one vertex. It has one face, and we have $1 - 0 + 1 = 2$. The induction base is proved.

Now take a graph with n edges, $n \geq 1$. Consider two cases.
1) The graph is acyclic. Then it is a tree. A tree does not separate the plane, so we have $|F| = 1$ in this case. By Theorem 2.7, $|E| = |V| - 1$. Thus we have

$$|V| - |E| + |F| = |V| - (|V| - 1) + 1 = 2.$$

2) The graph contains a cycle. Let $C \subset G$ be any cycle and let e be any edge of C. The graph $G - e$ is still connected and has one edge less. Let us show that it also has one face less. Indeed, the points on different sides of e belong to different faces of G: any arc connecting them must intersect the cycle C. These two faces are merged to one face in $G - e$; all other faces are unchanged. Thus G has the same number of vertices, one edge less, and one face less than $G - e$. By the induction assumption, Euler's formula holds for $G - e$. Thus it also holds for G. □

Theorem 3.9. *Let G be a connected plane graph with $n \geq 3$ vertices. Then $|E| \leq 3|V| - 6$. Moreover, equality holds if and only if all faces of G are triangles.*

Proof. Observe that in a plane graph with ≥ 3 vertices the degree of every face is at least 3. Indeed, the only way for a face of a connected graph to have degree 2 is to enclose an edge, in which case the graph has two vertices and one edge. A face cannot have degree 1. And if a face of a connected graph has degree 0, then the graph consists of a single vertex.

Then from Theorem 3.6 and Euler's formula we get the inequality

$$2|E| = \sum_{f \in F} \deg f \geq 3|F| = 3(2 - |V| + |E|),$$

which implies $|E| \leq 3|V| - 6$. The equality takes place only if the $\deg f = 3$ for all f, that is if all faces are triangles. □

Theorem 3.9 implies that the graph K_5 is not planar. Indeed, it has 5 vertices and $10 > 3 \cdot 5 - 6$ edges. This proof of non-planarity of K_5 looks very nice and seems to avoid the intricacies of Jordan's curve theorem. The simplicity is deceiving: Jordan's curve theorem is needed in the proof of Euler's formula (when we say that the cycle separates the plane).

An attempt to prove the non-planarity of $K_{3,3}$ in the same way fails: this graph has 6 vertices and $9 \leq 3 \cdot 6 - 6$ edges. Note however that a planar embedding of $K_{3,3}$ (if it exists) has no faces of degree 3 (a bipartite graph contains no odd cycles). For any plane graph without triangles we have

$$2|E| = \sum_{f \in F} \deg f \geq 4|F| = 4(2 - |V| + |E|),$$

which implies $|E| \leq 2|V| - 4$. Since $K_{3,3}$ does not satisfy this inequality: $9 > 2 \cdot 6 - 4$, it is not planar.

3.3 Planarity criteria

Theorem 3.10 (Kuratowski)**.** *A graph is planar if and only if it does not contain a subgraph isomorphic to a subdivision of K_5 or $K_{3,3}$.*

A graph G' is a *subdivision* of a graph G if G' is obtained from G by repeated *edge subdivisions*. To subdivide an edge $e = \{v, w\}$ of a graph G means to introduce a new vertex x, delete e, and introduce two new edges $\{x, v\}$ and $\{x, w\}$. Figure 12 shows a subdivision of K_5 and a subdivision of $K_{3,3}$.

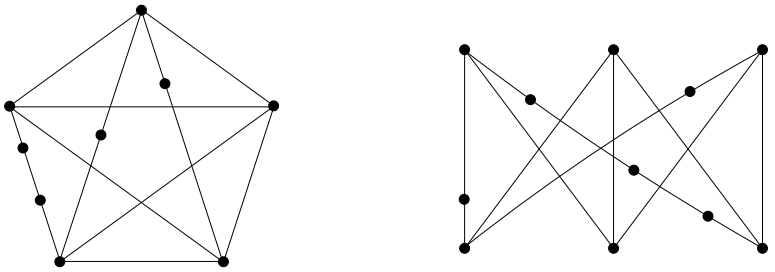

Figure 12: Some subdivisions of K_5 and $K_{3,3}$.

One direction of the Kuratowski theorem is easy to prove: If a graph contains a subdivision of K_5 or $K_{3,3}$, then it cannot be planar. Indeed, an embedding of the graph would contain an embedding of a subdivision of K_5 or $K_{3,3}$, and hence an embedding of K_5 or $K_{3,3}$. It is the opposite

direction which is the most interesting and non-obvious: the only obstacles to existence of a planar embedding of G are graphs K_5 or $K_{3,3}$ contained in G (in the form of subdivisions).

There is a similar planarity criterion that uses the notion of a *minor*.

Theorem 3.11 (Wagner)**.** *A graph is planar if and only if it does not have a minor isomorphic to K_5 or $K_{3,3}$.*

A minor of a graph G is any graph obtained by repeated vertex deletions, edge deletions and edge contractions. See Figure 13 for an example of an edge contraction. If an edge contraction results in a multiple edge, then we replace a multiple edge by a simple edge. If it results in a loop, then we remove a loop.

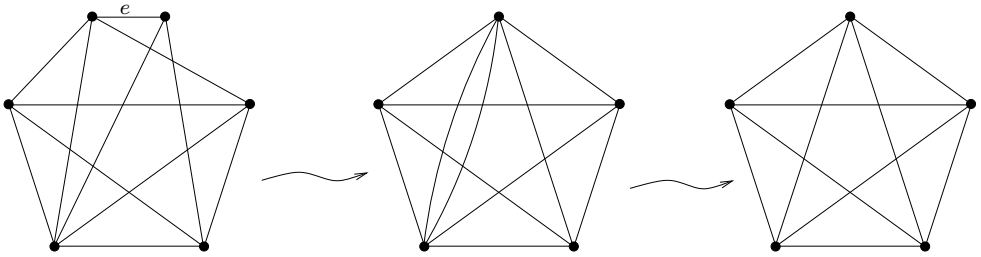

Figure 13: An example of edge contraction.

Note that a minor of G is not necessarily isomorphic to a subgraph of G. For example, the cycle C_3 is a minor of C_4 but not its subgraph.

Similarly to the Kuratowski theorem, one direction of the Wagner theorem is easy to prove, but not the other.

3.4 Duality for embedded graphs

Let G be a connected plane graph. Define a new plane graph G^* as follows. Inside every face f of G choose a point f^*. For every edge e of G draw an arc e^* crossing the edge e and joining the points f_1^* and f_2^* inside the faces incident with e. (If e is incident to one face only, then the arc e^* is a loop.)

It is possible to draw all arcs e^* so that they do not intersect each other. (Mark a point in the interior of every edge; inside every face f, join the point f^* to the points marked on the incident edges in a non-self-intersecting way.)

See Figure 14 for an example.

Different planar embeddings of the same graph may have different duals. For example, consider the duals of the graphs on Figure 9.

Let us describe those graphs whose duals have no loops and multiple edges.

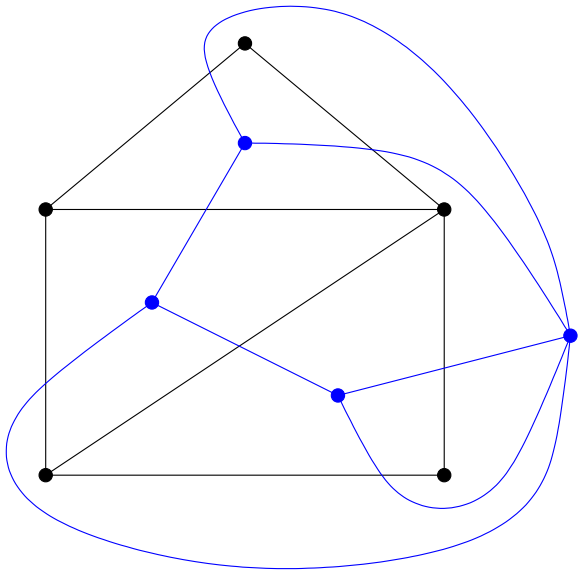

Figure 14: A graph and its dual.

Definition 3.12. *A graph is called* k-edge connected *if one needs to delete at least* k *edges in order to disconnect the graph.*

Thus a graph is 1-edge connected graphs if and only if it is connected. A connected graph is not 2-edge connected if it has a *bridge* or *cut edge*: an edge whose deletion disconnects the graph.

Lemma 3.13. *The dual of a plane graph has no loops if and only if the graph is* 2*-connected. The dual of a plane graph has no multiple edges if and only if the graph is* 3*-connected.*

The faces of the dual graph G^* correspond to the vertices of G: the duals of the edges incident to $v \in V(G)$ form a cycle around v. Thus we have bijections

$$V(G) \mapsto F(G^*), \quad E(G) \mapsto E(G^*), \quad F(G) \mapsto V(G^*).$$

Observe that Theorem 3.6 is equivalent to the handshake lemma for the dual graph G^*.

Finally, note that the dual of G^* is G again:

$$(G^*)^* = G.$$

For more information on planar graphs, see [3, Chapter 10].

4 Matchings

4.1 Basics

Definition 4.1. *Let $G = (V, E)$ be a graph. A set of its edges $M \subset E$ is called a* matching *if no two edges in M have a common vertex. If a vertex v belongs to an edge of M, then v is called* matched*, otherwise* unmatched*. A matching is called* perfect *if all vertices of G are matched.*

A graph that has at least one perfect matching is called *matchable*. A matchable graph must have an even number of vertices. (More generally, every connected component of a matchable graph must have an even number of vertices.) But there are also connected unmatchable graphs with an even number of vertices, see Figure 15, right.

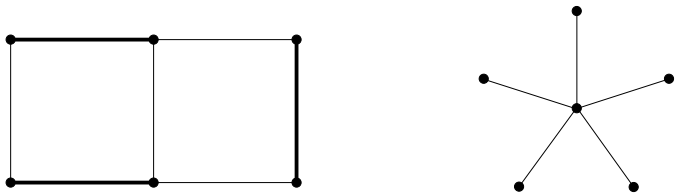

Figure 15: Two connected graphs on six vertices: one matchable (matching is shown by thick edges) and one non-matchable.

If it is not possible to match all vertices, then one can ask for a maximum possible matching. There are two notions of maximality.

Definition 4.2. *A matching M in a graph G is called a* maximum matching *if it has the maximum possible number of edges among all matchings in G (in other words if it covers a maximum possible number of vertices).*

A matching is called inclusion-maximal *if it is not contained in any larger matching.*

A maximum matching is obviously inclusion-maximal, but not vice versa. See Figure 16 for two examples.

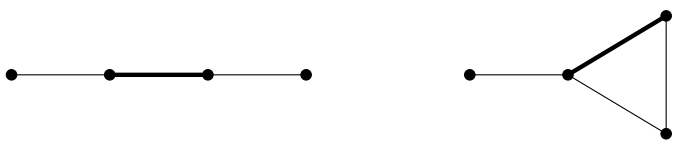

Figure 16: Two inclusion-maximal but not maximum matchings.

It follows that the greedy algorithm (add edges as long as it is possible) does not always find a maximum matching.

4.2 Augmenting paths and maximum matchings

Assume that M is a non-maximum matching in a graph. We want to modify it so that to obtain a matching with more edges. By the previous section, it is not always possible just to add edges to M, we also have to remove some.

Definition 4.3. *Let M be a matching in G. An* M-alternating path *in G is a path whose edges are alternately in M and not in M.*

Figure 17 shows possible types of alternating paths.

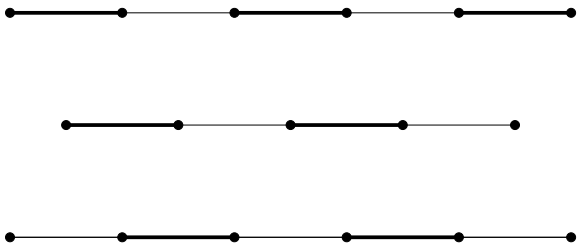

Figure 17: Alternating paths.

Definition 4.4. *An* M-augmenting path *is an M-alternating path that starts and ends with unmatched vertices.*

It follows that an M-augmenting path can look only as the bottom path on Figure 17. In addition, there must be no edge of M incident to the initial and terminal vertices of the path. See Figure 18.

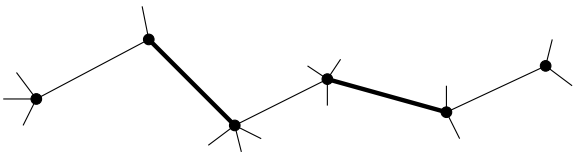

Figure 18: An augmenting path.

An M-augmenting path can be used to modify M to a bigger matching by "switching" the edges along the path. This is illustrated in Figure 19.

The following theorem provides a basis to an algorithm for finding a maximum matching.

Theorem 4.5 (Berge)**.** *A matching M in G is a maximum matching if and only if G contains no M-augmenting path.*

Proof. Let us prove that if M is a maximum matching, then there is no M-augmenting path. By contraposition we have to show that if G contains

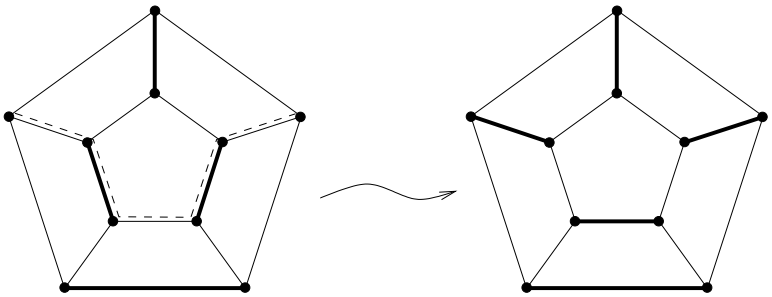

Figure 19: Modifying a matching with the help of an augmenting path.

an M-augmenting path, then M is not a maximum matching. This follows from the observation we made before stating the theorem: an augmenting path can be used to increase the number of edges in a matching.

For the opposite direction we have to show that if M is not a maximum matching, then there is an M-augmenting path. Let M' be a maximum matching in G. Then $|M'| > |M|$. Let H be the graph formed by those edges that belong to exactly one of M and M': the edge set of H is

$$(M \setminus M') \cup (M \setminus M').$$

(This is called symmetric difference of M and M'.) Every vertex of H has degree 1 or 2 because it is incident to at most one edge from M and at most one edge from M'. Therefore each connected component of H is either an even cycle with edges alternately in M and M' or a path with edges alternately in M and M', see Figure 20.

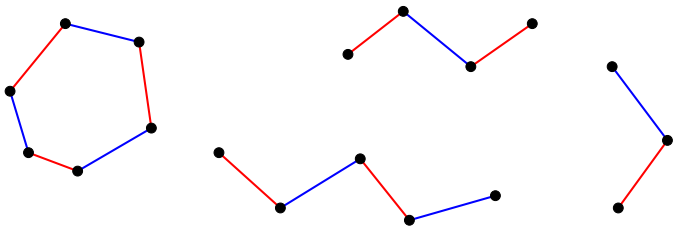

Figure 20: Symmetric difference of a non-maximum matching M (blue) and a maximum matching M' (red).

Due to $|M'| > |M|$, in H there are more edges from M' than from M. Therefore there is a path that starts and ends with an M'-edge. The endpoints of this path have no incident M-edges, otherwise such an edge would also belong to H, and the path would not stop here. This path is an M-augmenting path and the theorem is proved. □

In order to find a maximum matching in a graph, start with any matching (for example, an empty set of edges). Then, recursively, find an augmenting path and modify the current matching. If no augmenting path can be found, then the current matching is a maximum one. Algorithms for finding an augmenting path are described in [3, Section 16.5].

4.3 Matchings in bipartite graphs: Hall's theorem

Definition 4.6. *Let $G = (V, E)$ be a graph. For any subset $S \subset V$ of the vertex set denote by $N(S)$ the set of all vertices adjacent to vertices in S. The set $N(S)$ is called the* neighbor set *of S.*

In a bipartite graph $(X \cup Y, E)$, if $S \subset X$, then $N(S) \subset Y$.

Theorem 4.7 (Hall)**.** *Let $G = (X \cup Y, E)$ be a bipartite graph. Then G contains a matching that covers X if and only if*

$$|N(S)| \geq |S| \text{ for all } S \subset X. \tag{1}$$

In particular, if $|X| = |Y|$, then the above theorem provides a necessary and sufficient condition for the existence of a perfect matching.

Proof. Assume there is a matching that covers all of X. This defines an injective map $f\colon X \to Y$ associating to every $x \in X$ its matched vertex in Y. For every subset $S \subset X$ we have $f(S) \subset N(S)$, hence $|N(S)| \geq |f(S)| = |S|$. Thus condition (1) is necessary.

Let us show that it is sufficient. Let M be a maximum matching of G. Suppose that M does not cover X; we will prove that (1) is violated. Take a vertex $u \in X$ unmatched by M and consider all alternating paths starting from u. All these paths start with non-M-edges.

There are alternating paths of two sorts. Those made of an even number of edges end in X. Denote the set of their endpoints by S. (We have $u \in S$, because we also consider the path of zero length starting and ending at u.) Alternating paths made of an odd number of edges end in Y. Denote the set of their endpoints by T. See Figure 21.

We now prove a series of claims.

Claim 1. Every vertex $v \in S$ other than u is matched to a vertex in T. Indeed, v is the endpoint of a non-trival alternating path P of even length. The last edge of P belongs to M, thus v is matched to some vertex $w \in Y$. We have $w \in T$, because w is the endpoint of an alternating path $P - \{v, w\}$.

Claim 2. Every vertex $w \in T$ is matched to a vertex in S. Assume that some $w \in T$ is not matched. Then the alternating path P from u to w is an augmenting path, which contradicts the assumption of maximality of M. If w is matched to $v \in X$, then the path $P + \{v, w\}$ is also an augmenting path, thus $v \in S$.

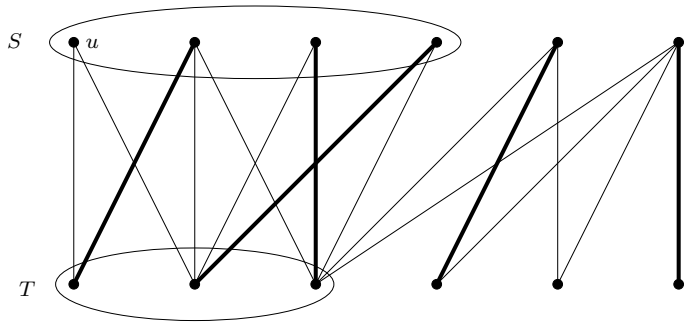

Figure 21: To the proof of Theorem 4.7.

Claim 3. $|T| = |S| - 1$. Indeed, by the previous two claims the matching M establishes a bijection between $S \setminus \{u\}$ and T.

Claim 4. $N(S) = T$. Every edge incident to u is a length 1 alternating path. Thus all neighbors of u belong to T. Let e be an edge incident to $v \in S$, $v \neq u$. If e belongs to M, then its other endpoint is in T by Claim 1. If e does not belong to M, then it extends an alternating path ending in v, thus again ends in T. This proves $N(S) \subset T$. We have $T \subset N(S)$ by construction of S and T: every vertex $w \in T$ is the endpoint of an alternating path starting at u. Just before coming to w, this path visited a vertex in S.

It follows that $|N(S)| = |T| = |S| - 1 < |S|$, which violates (1). Thus the assumption that a maximum matching does not cover all of X was false, and the theorem is proved. □

Corollary 4.8. *Every k-regular ($k > 0$) bipartite graph has a perfect matching.*

Proof. Let $G = (X \cup Y, E)$ be a k-regular bipartite graph. Since every edge is incident to exactly one vertex from X, and every vertex is incident to exactly k edges, we have $|E| = k|X|$. Similarly, $|E| = k|Y|$. Thus we have $|X| = |Y|$.

Let us show that k-regularity implies condition (1). Take any $S \subset X$ and consider the bipartite graph $G' = (S \cup N(S), E')$, where E' is the set of all edges incident to a vertex from S. By the above argument, $|E'| = k|S|$. On the other hand, for every $v \in N(S)$ we have $\deg_{G'} v \leq k$, which implies $|E'| \leq k|N(S)|$. It follows that

$$|N(S)| \geq \frac{|E'|}{k} = |S|,$$

and we are done. □

Chapter III

Propositional logic

Introduction

We will study two logical systems: *propositional logic* and *predicate logic*. The predicate logic is also know as first-order logic; there is also second-order logic and higher order logics.

Every logic has two aspects: *syntax* and *semantics*. On the syntactic side, logic consists of a *language*, which is a set of expressions built according to certain rules. The semantics interprets each of these expressions as true or false.

There are obvious parallels to human languages, but also important differences. Every human language has syntax rules, but firstly a real language is a living thing, so that the vocabulary and syntax are constantly changing, and secondly one may intentionally break the rules in order to create an artistic effect (you will easily find examples in literature or cinema). Also, a human language contains sentences without truth values, such as "What's for lunch?" or "Mind the gap".

Thus, if we apply logic to a human language, then we can deal only with declarative sentences. An example of a declarative sentence is "It will snow for Christmas". This is certainly either true or false, although we do not know it at the moment.

Here we come to an important point. Although we have said that every logical expression has a well-defined truth value, it may be not clear what this value is. A *proof theory* provides tools for determining this truth value. Every proof theory contains a set of *inference rules* or *deduction rules*, which allow to determine the truth value of an expression if the truth values of some other expressions are known. This is, of course, the way we are using logic in the everyday life, when we are trying to convince someone. A classical example of deduction is

> All men are mortal.
> Socrates is a man.

Therefore, Socrates is mortal.

(If the first two statements are true, then so is the third one.)

Formalization of human reasoning was at the origin of logic and stimulated its development over the centuries. Another important motivation was the search for foundations of mathematics (end of XIX – beginning of XX century) as one has tried to axiomatize mathematics and formalize the mathematical reasoning. Every mathematical proposition is either true or false, but it can be difficult to determine which way it is. The four-color theorem and Fermat's Last Theorem are famous examples.

The following logic textbooks are recommended: [6, 4, 5, 9].

1 Syntax and semantics of propositional logic

1.1 Propositional formulas

The language of propositional logic consists of strings of symbols, where each of the symbols is one of the following:

- A proposition symbol p, q, r, s, $p_1, p_2, \ldots$. (A countably infinite set.)
- A logical connective $\wedge$, $\vee$, $\rightarrow$, $\neg$.
- An auxiliary symbol (or).

Sometimes to the list of logical connectives one adds $\leftrightarrow$ and $\bot$. We will abstain from this.

The logical connectives have the following names.

$\wedge$	and	conjunction
$\vee$	or	disjunction
$\rightarrow$	if ..., then ...	implication
$\neg$	not	negation

Now there come the syntax rules describing what strings of symbols are allowed.

Definition 1.1. *The set of* propositions *or* propositional formulas PROP *is defined as follows.*

1. *Proposition symbols belong to* PROP*. They are called* atoms *or* atomic propositions*.*
2. *If* $A \in$ PROP*, then* $\neg A \in$ PROP*.*
3. *If* $A, B \in$ PROP*, then* $(A \wedge B)$*,* $(A \vee B)$*,* $(A \rightarrow B) \in$ PROP*.*
4. *A string of symbols belongs to* PROP *only if it can be obtained by applying the above rules.*

The last condition can be replaced by the requirement that PROP is the smallest (in the sense of inclusion) set satisfying the first three conditions.

A propositional formula has a recursive structure that can be represented with a *parse tree*. See Figure 1 for an example.

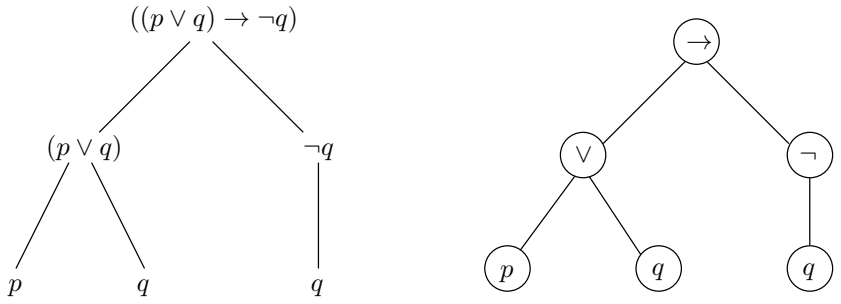

Figure 1: Parse tree for $((p \vee q) \to \neg q)$.

Parentheses in a propositional formula ensure that the parse tree is unique. If the root of the parsing tree is labeled with $\wedge, \vee$, or $\to$, then the formula starts with (and ends with). This pair of parentheses is not needed for parsing, and we will often omit it. Strictly speaking, this is syntactically wrong, but should not lead to confusion.

1.2 Truth tables

The set of *truth values* is a two-element set $\{\text{true}, \text{false}\}$. For brevity we will denote

$$1 = \text{true}, \quad 0 = \text{false}.$$

Various other abbreviations are used, for example $\mathbb{T}$ for "true" and $\mathbb{F}$ for "false".

Definition 1.2. *Let $S = \{p, q, r, s, p_1, p_2, \ldots\}$ be the set of all proposition symbols. A* valuation *is a map*

$$v \colon S \to \{0, 1\}$$

assigning a truth value to all propositional symbols.

Once the truth values of all proposition symbols are known, one can determine the truth value of any propositional formula. This is done with the help of the truth tables for logical connectives given below.

A	$\neg A$
0	1
1	0

A	B	$A \wedge B$	$A \vee B$	$A \to B$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

Here A and B are arbitrary propositions. The tables tell you the truth values of $\neg A$, $A \wedge B$, $A \vee B$, $A \to B$ once the truth values of A and B are known. They allow to write the truth table for any propositional formula recursively by "climbing" the parse tree.

Example 1.3. The following table gives the truth values for the formula $((p \vee q) \to \neg q)$ and all of its subformulas depending on a valuation $\{p, q\} \to \{0, 1\}$.

p	q	$(p \vee q)$	$\neg q$	$((p \vee q) \to \neg q)$
0	0	0	1	1
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0

Speaking more formally, any valuation v extends to a unique map

$$\hat{v} \colon \mathrm{PROP} \to \{0, 1\}$$

defined recursively by $\hat{v}(x) = v(x)$ for all $x \in S$ and by

$$\hat{v}(A \wedge B) = \hat{v}(A) \wedge \hat{v}(B) \qquad \hat{v}(A \vee B) = \hat{v}(A) \vee \hat{v}(B)$$
$$\hat{v}(A \to B) = \hat{v}(A) \to \hat{v}(B) \qquad \hat{v}(\neg A) = \neg \hat{v}(A),$$

where the values at the right hand sides are computed according to the truth tables of logical connectives.

Although by definition every valuation v assigns truth values to *all* proposition symbols, in order to determine $\hat{v}(A)$ we need to know only the values of symbols occuring in A.

1.3 Satisfiability, tautologies, logical equivalence

Definition 1.4. *Let A be a proposition and v be a valuation. If $\hat{v}(A) = 1$, then we say that v* satisfies *A and denote this by $v \vDash A$. If $\hat{v}(A) = 0$, then we say that v* falsifies *A and denote this by $v \nvDash A$.*

Definition 1.5. *A proposition A is* satisfiable *if it is satisfied by at least one valuation. A proposition is* unsatisfiable *if it is not satisfied by any valuation.*

A proposition A is valid *or a* tautology *if it is satisfied by all valuations: $\hat{v}(A) = 1$ for all v. We denote this by $\vDash A$.*

For example, looking at the truth table of Example 1.3 we see that the formula $((p \vee q) \to \neg q)$ is satisfiable but not a tautology.

Theorem 1.6. *1. Every tautology is satisfiable.*

2. A proposition A is a tautology if and only if $\neg A$ is unsatisfiable.

Proof. The set of all valuations is non-empty. Therefore if all valuations satisfy A, then there is at least one valuation that satisfies A.

From the truth table for $\neg$ it follows that

$$v \vDash A \text{ if and only if } v \nvDash \neg A.$$

It follows that A is satisfied by all valuations if and only if $\neg A$ is not satisfied by any. □

The problem of determining whether a given proposition is satisfiable is called *satisfiability problem*, abbreviated SAT. The problem of determining whether a given proposition is a tautology is called *tautology problem*, abbreviated $TAUT$. The satisfiability problem is NP-complete, that is every non-deterministically solvable in polynomial time problem can be reduced to it. Therefore if SAT can be solved in polynomial time, then every NP-problem can, that is $P = NP$. On the other hand, if $TAUT$ is not NP, then $P \neq NP$. For more details, see [6, Section 3.3.5].

Definition 1.7. *Two propositions A and B are called* logically equivalent*, which is denoted by $A \simeq B$ if they are satisfied by the same valuations:*

$$\hat{v}(A) = \hat{v}(B) \text{ for all } v.$$

In other words, $A \simeq B$ if and only if A and B have the same truth table.

Example 1.8. Looking at the truth table of Example 1.3 we see that

$$((p \vee q) \to \neg q) \simeq \neg q.$$

Remark 1.9. The symbol $\simeq$ of the logical equivalence does not belong to the alphabet of propositional logic, so that $A \simeq B$ is not a propositional formula. The symbols $\simeq$ and $\vDash$, and also symbols A and B used to denote arbitrary propositions, all belong to the *metalanguage*, a language that we use to describe propositional logic and prove its properties. Also the symbol $\Leftrightarrow$ that we will use in our arguments as an abbreviation for "if and only if" is a metasymbol. The definitions and statements in this section define notions of a metalanguage and formulate metatheorems.

When we prove some properties of propositional logic, we are implicitly using another, more complicated, logical system with its own semantics (what is true and what is not true). If, in turn, we want to discuss this more complicated system, then we need a metametalanguage and so on.

Imagine a computer program which is able to recognize propositional formulas and compute their truth values for any valuation. This program speaks the language of propositional logic, and it cannot analize its own actions. The programmer speaks a metalanguage and can predict the behavior of the program.

Theorem 1.10. *Propositions A and B are logically equivalent if and only if the proposition*

$$(A \to B) \wedge (B \to A)$$

is a tautology.

Proof. One proves that $\hat{v}((A \to B) \wedge (B \to A)) = 1$ if and only if $\hat{v}(A) = \hat{v}(B)$ by a case distinction, that is by considering all four possible combinations of values $\hat{v}(A)$ and $\hat{v}(B)$. Thus if $\hat{v}(A) = \hat{v}(B)$ for all v, then $(A \to B) \wedge (B \to A)$ is a tautology, and vice versa. □

There are several simple and useful equivalences between propositional formulas.

- Associativity laws

$$(A \wedge B) \wedge C \simeq A \wedge (B \wedge C) \qquad (A \vee B) \vee C \simeq A \vee (B \vee C)$$

- Commutativity laws

$$A \wedge B \simeq B \wedge A \qquad A \vee B \simeq B \vee A$$

- Distributivity laws

$$A \wedge (B \vee C) \simeq (A \wedge B) \vee (A \wedge C) \qquad A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$$

- De Morgan's laws

$$\neg(A \wedge B) \simeq \neg A \vee \neg B \qquad \neg(A \vee B) \simeq \neg A \wedge \neg B$$

- Idempotency laws

$$A \wedge A \simeq A \qquad A \vee A \simeq A$$

- Double negation law

$$\neg\neg A \simeq A$$

The associativity laws allow us to omit parentheses in conjunctions or disjunctions. Due to it we can allow abuse of notation and write strings like this one:

$$p \wedge q \wedge r \wedge s.$$

This is against the syntax rules. In order to conform the rules, we must put some parentheses. This leads to several different propositional formulas, like $((p \wedge q) \wedge (r \wedge s))$ or $((p \wedge (q \wedge r)) \wedge s)$. Although these formulas are different, they are logically equivalent. Thus, the string $p \wedge q \wedge r \wedge s$ stands for a class of equivalent formulas.

The distributivity laws allow us to operate with brackets as if $\wedge$ is the multiplication and $\vee$ is the addition or vice versa. For example,

$$\neg p \vee (p \wedge q) \simeq (\neg p \vee p) \wedge (\neg p \vee q).$$

Let us extend the language PROP by adding two new symbols $\top$ and $\bot$. We define a language $\widetilde{\text{PROP}}$ by adding to the point 1. in Definition 1.1 that $\top$ and $\bot$ belong to $\widetilde{\text{PROP}}$ and leaving the other conditions as they are. The symbols $\top$ and $\bot$ are *logical constants*: when computing the truth value of a proposition A we replace each $\top$ by 1 and each $\bot$ by 0. In $\widetilde{\text{PROP}}$ there are the following logical equivalences.

- Laws of zero and one.

$$\begin{aligned} (A\wedge \bot) &\simeq \bot & (A\vee \bot) &\simeq A \\ (A \wedge \top) &\simeq A & (A \vee \top) &\simeq \top \\ (A \wedge \neg A) &\simeq \bot & (A \vee \neg A) &\simeq \top \end{aligned}$$

These equivalences can be used in order to simplify certain propositions from PROP. As an example, let us simplify the formula we just obtained.

$$(\neg p \vee p) \wedge (\neg p \vee q) \simeq \top \wedge (\neg p \vee q) \simeq \neg p \vee q.$$

1.4 Boolean functions

Although the set of all proposition symbols is infinite, every proposition contains only a finite number of distinct proposition symbols. Assume that $A \in \text{PROP}$ contains only symbols from the set $\{p_1, p_2, \ldots, p_n\}$. Then A defines a map

$$f_A \colon \{0,1\}^n \to \{0,1\}$$

in the following way. Every element $(x_1, \ldots, x_n) \in \{0,1\}^n$ can be viewed as a partial valuation, giving p_i the truth value x_i for $i = 1, \ldots, n$. Then $f_A(x_1, \ldots, x_n)$ is the truth value of A corresponding to this valuation:

$$f_A(x_1, \ldots, x_n) = \hat{v}(A), \text{ where } v(p_i) = x_i.$$

The function f_A is described by the truth table of proposition A. As an immediate reformulation of Definition 1.7,

$$A \simeq B \Leftrightarrow f_A = f_B.$$

In fact, 0-1-valued functions of 0-1-valued arguments are a very basic object which can be studied irrespective of propositional logic.

Definition 1.11. *A map* $f \colon \{0,1\}^n \to \{0,1\}$ *is called a* boolean function *of* n *variables.*

We have shown that every propositional formula A determines a boolean function f_A. One can now ask whether every boolean function f can appear in this way. The answer is "yes".

Theorem 1.12. *Every boolean function* $\{0,1\}^n \to \{0,1\}$ *can be represented by a propositional formula.*

Proof. First let us find a formula that represents a very simple function: a function that takes value 1 at one point only. Take any vector $x \in \{0,1\}^n$ and put

$$f(x) = 1, \quad f(y) = 0 \text{ for all } y \neq x.$$

We need a propositional formula with symbols $p_1, \dots, p_n$ that evaluates to 1 only when $v(p_i) = x_i$ for all i. This is achieved by a conjunction

$$B = C_1 \wedge C_2 \wedge \cdots \wedge C_n,$$

where

$$C_i = \begin{cases} p_i, & \text{if } x_1 = 1, \\ \neg p_i, & \text{if } x_1 = 0. \end{cases}$$

Let us prove that $f_B = f$ in a formal way. Take any $y \in \{0,1\}^n$. By definition, $f_B(y) = \hat{v}(B)$ for the valuation $v(p_i) = y_i$. Apply the recursive definition of $\hat{v}$:

$$f_B(y) = \hat{v}(B) = \hat{v}(C_1) \wedge \hat{v}(C_2) \wedge \cdots \wedge \hat{v}(C_n).$$

The right hand side is equal to 1 if and only if $\hat{v}(C_i) = 1$ for all i. By definition of C_i we have

$$\begin{aligned} &\text{if } x_i = 1, \text{ then } \hat{v}(C_i) = \hat{v}(p_i) = v(p_i) = y_i \\ &\text{if } x_i = 0, \text{ then } \hat{v}(C_i) = \hat{v}(\neg p_i) = \neg \hat{v}(p_i) = \neg y_i, \end{aligned}$$

which implies that $\hat{v}(C_i) = 1$ if and only if $y_i = x_i$. It follows that $f_B(y) = 1$ if and only if $y_i = x_i$ for all i, that is $f_B = f$.

Now let f be an arbitrary Boolean function of n arguments. If $f(x) = 0$ for all x, then one can represent f by the formula $p_1 \wedge \neg p_1$. Assume that there is at least one x such that $f(x) = 1$. For every $x \in \{0,1\}^n$ such that $f(x) = 1$ construct a conjunction B_x as above and take the disjunction of all such B_x:

$$A = \bigvee_{f(x)=1} B_x.$$

We claim that $f_A = f$. Indeed, A evaluates to 1 if and only if at least one of B_x evaluates to 1 and, as we know, B_x evaluates to 1 only at x. Thus A evaluates to 1 exactly at those points where $f(x) = 1$.

Formally, for every y we have

$$f_A(y) = \hat{v}(A) = \bigvee_{f(x)=1} \hat{v}(B_x)$$

(where $v(p_i) = y_i$). The right hand side equals 1 if and only if $\hat{v}(B_x) = 1$ for some x. But $\hat{v}(B_x) = 1$ if and only if $y = x$. Thus $f_A(y) = 1$ if and only if $y = x$ for some x such that $f(x) = 1$, which is just a complicated way to say if and only if $f(y) = 1$. Hence $f_A = f$, and the theorem is proved. □

The above argument provides a procedure of writing a formula with a given truth table. We will illustrate it on the example of the function given by the table below.

p	q	?
0	0	0
0	1	1
1	0	1
1	1	0

Mark the rows with 1 at the end. For each of these rows write a conjunction of all proposition symbols, negated or not depending on whether the value of this symbol in this row is 0 or 1. In our case these are $\neg p \wedge q$ and $p \wedge \neg q$. Then write the disjunction of the obtained conjunctions:

$$(\neg p \wedge q) \vee (p \wedge \neg q).$$

Observe that the formula constructed in Theorem 1.12 does not use the connective $\rightarrow$.

Definition 1.13. *A system of logical connectives is called* functionally complete *if every Boolean function can be represented by a formula using only these connectives.*

Thus the system $\neg, \wedge, \vee$ is functionally complete. One can dispense of one of the connectives $\wedge$ or $\vee$ as well.

Lemma 1.14. *Each of the systems of logical connectives $\neg, \wedge$ and $\neg, \vee$ is functionally complete.*

Proof. By Theorem 1.12 every Boolean function can be represented by a formula using $\neg, \wedge, \vee$ only. Replace every occurrence of $\vee$ using a De Morgan law and the double negation:

$$A \vee B \simeq \neg(\neg A \wedge \neg B).$$

This gives an equivalent formula with connectives $\neg$ and $\wedge$ only. One removes conjunctions in a similar way with the help of the equivalence

$$A \wedge B \simeq \neg(\neg A \vee \neg B).$$

□

For example,

$$\begin{aligned}(\neg p \wedge q) \vee (p \wedge \neg q) &\simeq \neg(\neg(\neg p \wedge q) \wedge \neg(p \wedge \neg q)) \\ &\simeq \neg(p \vee \neg q) \vee \neg(\neg p \vee q)\end{aligned}$$

One can achieve an absolute minimalism by introducing a logical connective $\uparrow$ with the truth table

A	B	$A \uparrow B$
0	0	1
0	1	1
1	0	1
1	1	0

(also called $NAND$ for obvious reasons).

Lemma 1.15. *The system of a single logical connective $\uparrow$ is functionally complete.*

Proof. Exercise. □

1.5 Disjunctive and conjunctive normal forms

Propositional formulas that we obtained in Theorem 1.12 have a very special form: they are disjunction of conjunctions of (possibly negated) proposition symbols.

Let us fix some terminology. A *literal* is a proposition symbol or its negation. A *conjunctive clause* is a conjunction of one or several literals.

Definition 1.16. *A propositional formula is said to be in* disjunctive normal form *(for brevity,* DNF*) if it is a disjunction of conjunctive clauses.*

Corollary 1.17. *Every propositional formula is logically equivalent to a formula in DNF.*

Proof. This follows from Theorem 1.12. Every formula represents a boolean function. As we proved in Theorem 1.12, every boolean function is represented by a formula in DNF. Formulas representing the same function are logically equivalent. □

Similarly, a *disjunctive clause* is a disjunction of several literals.

Definition 1.18. *A propositional formula is said to be in* conjunctive normal form *(for brevity,* CNF*) if it is a conjunction of disjunctive clauses.*

Theorem 1.19. *Every propositional formula is logically equivalent to a formula in CNF.*

Proof. Let A be a propositional formula. By Corollary 1.17 the negation of A is equivalent to some formula in DNF:

$$\neg A \simeq \bigvee_{\alpha=1}^{N} B_\alpha, \quad B_\alpha = C_{\alpha,1} \wedge \cdots \wedge C_{\alpha,k_\alpha}, \quad C_{\alpha,i} \text{ literals.}$$

By De Morgan's law we have

$$A \simeq \neg\neg A \simeq \neg \bigvee_{\alpha=1}^{N} B_\alpha \simeq \bigwedge_{\alpha=1}^{N} \neg B_\alpha \simeq \bigwedge_{\alpha=1}^{N} (\neg C_{\alpha,1} \vee \cdots \vee \neg C_{\alpha,k_\alpha}).$$

The negation of a literal is a negated or a doubly negated symbol. Double negations can be removed, and we obtain a formula in CNF. □

An unsatisfiable formula is equivalent to $p \wedge \neg p$. This formula is at the same time in DNF (it is a single conjunctive clause) and in CNF (it is a conjunction of two disjunctive clauses). A tautology is equivalent to $p \vee \neg p$, which is also in DNF and in CNF.

Remark 1.20. If the logical symbols $\top$ and $\bot$ are present, then one can consider $\top$ as a disjunction of zero length (and thus a CNF and a DNF for a tautology), and $\bot$ as a conjunction of zero length (thus a CNF and a DNF for an unsatisfiable formula).

DNF is not unique: different formulas in DNF can be equivalent. The procedure described in Theorem 1.12 produces a disjunction of clauses of length n (where n is the number of variables in the input formula). For example, the formula $p \vee q$ (although it is already in DNF) will be represented by $(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q)$. The same applies to CNF.

2 Proof theories

2.1 Deductive systems

A proof theory is a method of establishing the validity of a proposition (that is whether a given proposition is a tautology). There is a type of proof theories called deductive systems. A deductive system consists of

- a set of propositional formulas called *axioms*;
- a set of inference rules.

An inference rule has the form $\Gamma \vdash A$, where A is a proposition and Γ is a set of propositions. It is convenient to write an axiom A in the form $\vdash A$. This allows to define provable propositions recursively.

Definition 2.1. *A proposition is called* provable *if and only if it is an axiom or can be obtained from provable propositions by applying rules of inference, that is there is a set of provable propositions* Γ *such that* $\Gamma \vdash A$ *is an inference rule.*

One can be more concrete by defining formal proofs as follows.

Definition 2.2. *A* formal proof *is a sequence of propositional formulas, where each formula is an axiom or is obtained from some of the previous formulas by an inference rule. That is, if* $\Gamma \vdash A$ *is an inference rule and the sequence already contains all the formulas from the list* Γ*, then you can add* A *at the end of the sequence.*

The final formula of a formal proof is called a theorem.

Thus, a formula is called provable if and only if it is a theorem. An axiom is also a theorem. Its proof is a sequence consisting of a single term.

Definition 2.3. *A proof system is called* sound *if every provable formula is a tautology. A proof system is called* complete *if every tautology is provable.*

In terms of the turnstile notation, a system is sound if $\vdash A \Rightarrow \vDash A$, and complete if $\vDash A \Rightarrow \vdash A$.

2.2 A Hilbert system

A Hilbert-style deductive system has a single inference rule, the so-called *modus ponens*:

$$A, A \to B \vdash B$$

(If A is provable and $A \to B$ is provable, then B is provable.) We will abbreviate modus ponens by MP.

There are many different axiom systems. Here is one of them, the third Łukasiewicz' system.

$$\vdash A \to (B \to A)$$
$$\vdash (A \to (B \to C)) \to ((A \to B) \to (A \to C))$$
$$\vdash (\neg A \to \neg B) \to (B \to A)$$

Here A, B, and C may be any propositional formulas. Thus, in fact, we have infinitely many axioms (axiom *instances*) that can be obtained from the above axiom *schemata* by substituting for A, B, and C some particular formulas.

Example 2.4. Let us prove the tautology $A \to A$. First, substitute in the second axiom $(A \to A)$ for B and A for C:

$$\vdash (A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$$

Now substitute $(A \to A)$ for B in the first axiom:

$$\vdash (A \to ((A \to A) \to A))$$

This coincides with the "left half" of the previous formula, so by modus ponens we infer the "right half" of that formula:

$$\vdash (A \to (A \to A)) \to (A \to A)$$

Now take the first axiom with B substituted for A:

$$\vdash (A \to (A \to A)).$$

This is the left half of the previous formula, so again by modus ponens we infer

$$\vdash (A \to A),$$

and the proof is finished.

Theorem 2.5. *A Hilbert system is sound if and only if all of its axioms are tautologies.*

Proof. Exercise. □

2.3 Gentzen's sequent calculus: the idea

Given a proposition A, we want to determine whether it is a tautology. Recall that A is *not* a tautology if and only if there is a valuation v that falsifies A. (We will also call v a *counterexample* to A.) So let us try to falsify A or to show that this is impossible. As an example, take

$$A = (p \to q) \to (p \vee q).$$

In order to falsify a proposition of the form $B \to C$ one has to satisfy B and falsify C at the same time. This reduces our problem to a combination of two simpler ones. Let us write our new task on the top of the old one:

$$\frac{p \to q \vdash p \vee q}{\vdash (p \to q) \to (p \vee q)}$$

The turnstile symbol is used as a separator: on the left we write the things we want to satisfy, on the right the things we want to falsify. (One can use any other separator. The choice of $\vdash$ looks awkward because the symbol was used earlier to denote provability. This choice is partially motivated by the subsequent sections.)

So, now we want to satisfy $p \to q$ and falsify $p \vee q$. In order to falsify $p \vee q$ we have to falsify both of them at the same time. We express this by $p \to q \vdash p, q$. Now on the right hand side we have a list of formulas that we want to falsify simultaneously. Our diagram takes the following form:

$$\dfrac{\dfrac{p \to q \vdash p, q}{p \to q \vdash p \vee q}}{\vdash (p \to q) \to (p \vee q)}$$

There are two ways to satisfy $p \to q$: one has either to satisfy q or to falsify p. This introduces branching in the diagram:

$$\dfrac{\dfrac{\dfrac{q \vdash p, q \qquad \vdash p, p, q}{p \to q \vdash p, q}}{p \to q \vdash p \vee q}}{\vdash (p \to q) \to (p \vee q)}$$

Now, any valuation that solves one of the problems on the top of the diagram also solves the problem in the bottom. The first problem sounds "satisfy p and falsify p and q". This is, of course, impossible. But the second problem says "falsify p and q". This is solved by setting $v(p) = 0$, $v(q) = 0$. From the construction principle of the diagram it follows that this valuation falsifies the initial proposition $A = (p \to q) \to (p \vee q)$. Thus we have shown that A is not a tautology.

A couple of remarks are in order. First, there is no need to repeat two equal propositions on the same side of $\vdash$ as we did with p on the top right. (We did it just to show that trying to satisfy $p \to q$ brings q to the left or p to the right.) Second, sometimes we have a choice which connective to eliminate. Here we had it at the second step. If we choose to eliminate $p \to q$ before $p \vee q$, then the branching happens earlier, and the final diagram looks as follows.

$$\dfrac{\dfrac{\dfrac{q \vdash p, q}{q \vdash p \vee q} \qquad \dfrac{\vdash p, q}{\vdash p, p \vee q}}{p \to q \vdash p \vee q}}{\vdash (p \to q) \to (p \vee q)}$$

2.4 Sequents and inference rules

In the above example we have operated with sets of propositions split into two subsets: those to satisy and those to falsify.

Definition 2.6. *A* sequent *is a pair* (Γ, Δ) *of finite (possibly empty) sets of propositions. The set* Γ *is called the* antecedent*, the set* Δ *is called the* succedent*. A sequent is written as* $\Gamma \vdash \Delta$ *with the elements of* Γ *and* Δ *listed in an arbitrary order.*

For any proposition in the sequent there is a rule that eliminates its outermost connective. This rule depends on whether the proposition occurs in the antecedent or in the succedent. Thus there are eight rules operating on sequents, two for each of the connectives $\wedge$, $\vee$, $\to$, $\neg$.

Definition 2.7. *The* inference rules *of the sequent calculus are as follows.*

$$(\wedge \text{ left}): \frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \qquad (\wedge \text{ right}): \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$$

$$(\vee \text{ left}): \frac{A, \Gamma \vdash \Delta \qquad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta} \qquad (\vee \text{ right}): \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}$$

$$(\to \text{ left}): \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{A \to B, \Gamma \vdash \Delta} \qquad (\to \text{ right}): \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}$$

$$(\neg \text{ left}): \frac{\Gamma \vdash A, \Delta}{\neg A, \Gamma \vdash \Delta} \qquad (\neg \text{ right}): \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$$

The sequents above the line are called premises*, the sequents below the line are called* conclusions*.*

Definition 2.8. *A valuation v* falsifies *a sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ if it satisfies all of A_i and falsifies all of B_j:*

$$v \vDash (A_1 \wedge \cdots \wedge A_m) \wedge (\neg B_1 \wedge \cdots \neg B_n).$$

A sequent is called falsifiable *if it is falsified by some valuation. A sequent is called* valid *if it is not falsifiable.*

Lemma 2.9. *For each of the rules given in Definition 2.7, a valuation v falsifies the conclusion if and only if it falsifies at least one of the premises.*

Proof. The proof is done by looking at the truth tables of the logical connectives. Let us prove, for example, the ($\wedge$ right) rule:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$$

A valuation v falsifies the conclusion if and only if it satisfies all propositions in Γ, falsifies $A \wedge B$, and falsifies all propositions in Δ. But v falsifies $A \wedge B$ if and only if it either falsifies A or falsifies B. Thus v falsifies the conclusion if and only if it

- satisfies Γ and falsifies A and Δ, or
- satisfies Γ and falsifies B and Δ.

These conditions are the premises of the ($\wedge$ right) rule, therefore the statement of the lemma holds for this rule. Similar arguments work for all of the other rules. □

For any proposition A one can form a sequent $\vdash A$, which is falsifiable, respectively valid, if and only if A is falsifiable, respectively valid (a tautology). Thus if we learn to prove all valid sequents, then we will be able to prove all tautologies.

2.5 Axioms and proofs

Definition 2.10. *A sequent* $\Gamma \vdash \Delta$ *is an* axiom *if and only if* $\Gamma \cap \Delta \neq \varnothing$*: there is a proposition listed both in* Γ *and* Δ*.*

Example 2.11. The following sequents are axioms:

- $q \vdash p, q$,
- $(p \to q), p \vdash (p \to q), q$.

Lemma 2.12. *All axioms are valid.*

Proof. In order to falsify a sequent $\Gamma \vdash \Delta$, we have to satisfy all propositions in Γ and falsify all propositions in Δ. If Γ and Δ contain a common proposition, then no valuation falsifies $\Gamma \vdash \Delta$. Thus, $\Gamma \vdash \Delta$ is valid. □

Definition 2.13. *A* deduction tree *is a rooted tree whose vertices are labeled with sequents so that every parent vertex forms an inference rule together with its children, where the children are the premises and the parent is the conclusion.*

A parent vertex in a rooted tree is a vertex with the out-degree ≥ 1 in the canonical orientation of the edges, see Figure 7. Vertices of out-degree 0 will be called *leaves* of a rooted tree. Since the in-degree of every non-root vertex is 1, non-root leaves are leaves in the usual sense. However, the root is a leaf if and only if the tree has only one vertex.

A standalone sequent is a simplest deduction tree (tree with one vertex). Next to it are the deduction trees copied from the inference rules as shown in Figure 2. Note that in a deduction tree, the root is at the bottom and the children of every vertex are situated above the vertex.

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$$

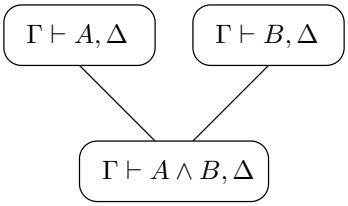

Figure 2: The (∧: right) inference rule as a deduction tree.

Definition 2.14. *A deduction tree is called a* proof tree *if all of its leaves are axioms. A sequent is called* provable *if there is a proof tree with this sequent at the root.*

2.6 Soundness of the sequent calculus

Theorem 2.15. *The sequent calculus is sound: every provable sequent is valid.*

Lemma 2.16. *A valuation falsifies the root of a non-trivial deduction tree if and only if it falsifies at least one of its leaves.*

Proof. Induction on the number of vertices.

For trees with one vertex the statement is trivial.

Assume that the statement is proved for all deduction trees with $< n$ vertices. Let T be a deduction tree with n vertices. By definition, if we delete the root of T, then we obtain one or two deduction trees "growing" from the children of the root, see Figure 3. Besides, the root of T is the conclusion and its children are the premises of an inference rule. Hence by Lemma 2.9 a valuation falsifies the root if and only if it falsifies one of its children. The subtrees growing from the children have $< n$ vertices, therefore by induction assumption a valuation falsifies a child if and only if it falsifies one of the leaves of the corresponding subtree. Such a vertex is also childless in the big tree. This proves the induction step. □

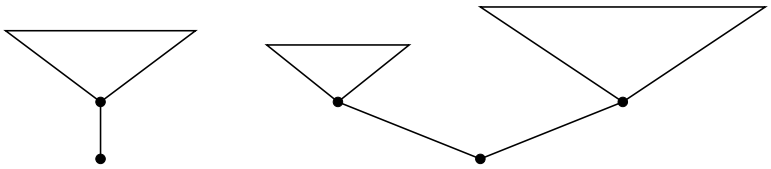

Figure 3: Deduction trees near their roots.

Proof of Theorem 2.15. We want to show that for every proof tree the sequent at its root is valid. By Lemma 2.16, if some valuation falsifies the root, then it falsifies one of the leaves. But all leaves of a proof tree are axioms, and by Lemma 2.12 they cannot be falsified. □

2.7 Closed deduction trees and completeness of the sequent calculus

Definition 2.17. *A deduction tree is called* closed *if the sequents labeling its leaves consist of proposition symbols only.*

Lemma 2.18. *Every sequent is the root of some closed deduction tree.*

Proof. Induction on the number of connectives.

A sequent without connectives consists of proposition symbols. Assume that every sequent with $< n$ connectives is the root of a closed tree. Take

any sequent S with n connectives, choose the outermost connective in any of its propositions and eliminate it. This produces a small tree with S at the root. In every inference rule, each of the premises contains less connectives than the conclusion. Thus by induction assumption the leaves of our small tree are roots of closed deduction trees. Together, this produces a closed deduction tree with S at the root. □

Theorem 2.19. *The sequent calculus is complete: every valid sequent is provable.*

Proof. For a given sequent, consider its closed deduction tree. It has leaves of two kinds:

- Leaves labeled with axioms $p_1, \ldots, p_k \vdash q_1, \ldots, q_l$ such that $p_i = q_j$ for some i, j.
- Leaves with labels $p_1, \ldots, p_k \vdash q_1, \ldots, q_l$ such that $p_i \neq q_j$ for all i, j. These are called counterexample leaves.

A counterexample leaf is falsifiable: it suffices to set $v(p_i) = 1$ for all i and $v(q_j) = 0$ for all j. Thus if the closed deduction tree has a counterexample leaf, then the sequent at the root is also falsifiable.

If our sequent is valid, then all of the leaves are of the first kind, and the tree is a proof tree. □

Note that we get more than just completeness: constructing a closed deduction tree provides a concrete counterexample to a falsifiable sequent.

2.8 A byproduct: CNF and DNF

Theorem 2.20. *Let A be a proposition. Take a closed deduction tree with $\vdash A$ at the root. For each counterexample leaf $p_1, \ldots, p_k \vdash q_1, \ldots, q_l$ of this tree write a disjunctive clause*

$$\neg p_1 \vee \cdots \vee \neg p_k \vee q_1 \vee \cdots \vee q_l.$$

Then the conjunction of all such clauses is a CNF for A.

Proof. By Lemma 2.16, a valuation v satisfies A if and only if it satisfies all leaves of the deduction tree. The leaf $p_1, \ldots, p_k \vdash q_1, \ldots, q_l$ is satisfied if and only if the disjunctive clause above is satisfied (see Definition 2.8). Finally, v satisfies all leaves if and only if it satisfies the conjunction of all these clauses. □

One constructs a DNF for A in a similar way starting from the sequent $A \vdash$.

Chapter IV

Predicate logic

Introduction

Predicate logic is a more complicated and powerful system than the propositional logic.

Before proceeding to formal definitions, let us have a glimpse at how it works. Consider the statement

> For every number x one has $x < x + 1$.

It can be expressed by a predicate formula

$$\forall x P(x, f(x)), \tag{1}$$

where $f(x) = x + 1$, and $P(x, y)$ means $x < y$. Functions of one or several arguments that take truth values (such as $x < y$ or "x is blue") are called *predicates*. Formula (1) contains all the main building blocks of the predicate logic: a variable x, a function f, and a predicate P.

Let us now forget the origin of formula (1). In order to make sense of it, its elements must be interpreted: what kind of objects are represented by the variable x, how is the function f defined, and what does $P(x, y)$ mean. This interpretation can be as above, but can also be different. For example, the same formula can be interpreted as

> It gets colder every day.

Now x is a day, $f(x)$ is the day after day x, and $P(x, y)$ means "y is colder than x".

Our first interpretation evaluates the formula (1) to true, while the second evaluates it to false. A formula of the predicate logic is *valid* if it evaluates to true in all interpretations. Similarly to the propositional logic, one aims at finding a method (a proof theory) to establish the validity of a formula.

1 Syntax and semantics of predicate logic

1.1 First-order languages

The alphabet of the predicate logic consists of

- variables;
- function symbols;
- predicate symbols;
- logical connectives $\wedge, \vee, \neg, \to, \forall, \exists$;
- auxiliary symbols (and);
- equality symbol $=$.

Informally speaking (and as indicated in the introduction), a variable is an object, a function is an operation with objects whose result is also an object, and a predicate is a statement about one or several object (in other words, an operation with objects whose result is a truth value).

Compared to the propositional logic, we have two new logical connectives: the *universal quantifier* $\forall$ and the *existential quantifier* $\exists$.

The equality symbol is not always included in the alphabet. Accordingly, there are two slightly different versions of predicate logic: logic with equality and logic without equality.

Before stating the rules according to which the alphabet symbols can be combined one has to fix a *signature*. This is a list of function symbols and predicate symbols together with the number of arguments for each of them.

Definition 1.1. *A* signature *is a triple of sets* $(\mathcal{V}, \mathcal{F}, \mathcal{P})$ *together with two maps* $\mathcal{F} \to \mathbb{N} \cup \{0\}$ *and* $\mathcal{P} \to \mathbb{N} \cup \{0\}$ *which describe the* arity *of functions and predicates.*

One can use the same set $\mathcal{V}$ of variables in all signatures. This is a countably infinite set; we use the letters x, y, z, $x_1, x_2, \ldots$ etc. to denote its elements. As function symbols we will use f, g, h, $f_1, f_2, \ldots$ and as predicate symbols P, Q, R, $P_1, P_2, \ldots$.

The strange word *arity* describes the number of arguments: a function or a predicate can be unary, binary, k-ary, and even nullary. The arity can be indicated by the number of dots (or other placeholders) between the brackets. Below is an example of a signature.

$\mathcal{F}$:	$f(\cdot)$	one unary function symbol
$\mathcal{P}$:	$P(\cdot,\cdot), Q(\cdot)$	one binary and one unary predicate symbol

Nullary functions and nullary predicates have no arguments.

Definition 1.2. Terms *are expressions that can be obtained by finitely many applications of the following rules.*

- *Any variable is a term.*
- *If f is a k-ary function symbol, and $t_1, \dots, t_k$ are terms, then $f(t_1, \dots, t_k)$ is a term.*

In particular, if f is a nullary function, then $f()$ is a term. We will usually omit the brackets in this situation. This has one drawback: syntactically, a nullary function becomes indistinguishable from a variable. (Later we will see that their semantics is different.)

For example, with a ternary function f, a binary function g, and a 0-ary function a one can compose the following terms:

$$f(x, g(y, x), a), \quad g(f(x, y, a), g(x, z)).$$

Definition 1.3. Formulas *are expressions that can be obtained by finitely many applications of the following rules.*

- *If P is a k-ary predicate symbol, and $t_1, \dots, t_k$ are terms, then $P(t_1, \dots, t_k)$ is a formula.*
- *If t_1 and t_2 are terms, then $t_1 = t_2$ is a formula.*
- *If φ is a formula, then so is $\neg\varphi$.*
- *If φ and ψ are formulas, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$.*
- *If φ is a formula, and x is a variable, then $\forall x\varphi$ and $\exists x\varphi$ are formulas.*

Formulas obtained by applying the first or the second rule are called *atomic* formulas (because they cannot be decomposed into smaller formulas).

For example, in the signature

$$\mathcal{F} : a(), f(\cdot)$$
$$\mathcal{P} : P(\cdot, \cdot)$$

one can build the following formulas:

$$\forall x \neg (f(x) = a)$$
$$\forall x (f(x) = f(y) \to x = y)$$
$$\forall x P(x, f(x)).$$

Definition 1.4. *The set of all formulas (based on a given signature) is called a* first-order language.

1.2 Free and bound variables

A variable in a predicate formula may be *bound* by a quantifier or be *free*. For example, in the formula $\forall x P(x, y)$ the variable x is bound, while the variable y is free.

Unfortunately, this is not as simple as it seems. Consider the formula

$$(\forall x P(x, y)) \to (\exists y Q(y)).$$

The variable y in the first half of the formula is free, the same variable y in the second half is bound. It is more convenient to work with formulas where such things do not happen.

Definition 1.5. *A formula is called* rectified *if the following two conditions are satisfied:*

- *no variable occurs as free and as bound;*
- *distinct quantifiers bind distinct variables.*

One can rectify any formula by renaming the variables as follows.

- For every quantifier in the formula determine its "scope". For this, look at the recursive structure of the formula, namely at the step when the quantifier appeared: $\varphi \rightsquigarrow \forall x \varphi$. The subformula φ is the scope of $\forall x$.
- If the variable x bound by a quantifier occurs not only inside the scope, but also outside, then replace all occurrences of x inside the scope and under the quantifier by some completely new symbol.

The resulting formula is semantically equivalent to the original one (although we have not yet described the semantics).

Example 1.6. Rectification of

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

produces, for example,

$$\exists z \forall t P(z, t) \to \forall y \exists x P(x, y).$$

Remark 1.7. Non-rectified formulas are absolutely legitimate. The main reason to introduce rectification is that some formal definitions and procedures are easier to describe for rectified formulas than for non-rectified ones.

1.3 Closed formulas and universal closure

Definition 1.8. *A formula is called* closed *if it does not contain free variables.*

Definition 1.9. *The* universal closure *of a formula is obtained by adding a universal quantifier for every free variable.*

Example 1.10. The universal closure of $Q(x) \wedge \exists y P(y, z)$ is

$$\forall x \forall z (Q(x) \wedge \exists y P(y, z)).$$

1.4 First-order structures

A signature only lists symbols of functions and predicates with the number of their arguments; it does not specify what operations and relations stand behind these symbols.

Definition 1.11. *A* first-order structure *(based on a given signature) is a pair (U, I), where U is a non-empty set and I is a map which assigns to each k-ary function symbol f a function $I(f)\colon U^k \to U$ and to each k-ary predicate symbol P a predicate $I(P)\colon U^k \to \{0, 1\}$.*

For $k = 0$ the set U^k is a one-element set. Therefore an interpretation of a nullary function $I(a)\colon U^0 \to U$ is equivalent to choosing an element of U. This is the reason why nullary functions are also called *constants*. Similarly, an interpretation of a nullary predicate is a choice of a truth value for it.

Example 1.12. Consider again the signature

$$\mathcal{F} : a(), f(\cdot)$$
$$\mathcal{P} : P(\cdot, \cdot)$$

Here is one of possible structures for it:

$$U = \mathbb{N}, \quad a = 1, \quad I(f)(x) = x + 1, \quad I(P) = \{x \leq y\}.$$

Here $I(P) = \{x \leq y\}$ means $I(P)(x, y)$ is true if and only if $x \leq y$. In other words, we define $I(P)$ by describing the preimage of 1 under the map $I(P)$.

Given a structure, one can evaluate every closed formula. For example, the formula $\forall x \neg (f(x) = a)$ acquires the meaning

$$\forall x \in \mathbb{N}\ x + 1 \neq 1,$$

which we identify as a true statement.

It should be intuitively clear how to find the truth value of a given closed formula. But one needs a formal definition. Of course, it proceeds

by recursion because formulas have a recursive structure. This means that, even if we want to determine the truth values only of closed formulas, we also need to determine the truth values of formulas with free variables (which appear as intermediate steps when building a closed formula).

A *variable assignment* is a map $\mu\colon \mathcal{V} \to U$ which associated to every variable an element from the universe. Together with interpretation I it allows to evaluate every term to an element of U and every formula to a truth value.

- Given terms $t_1, \ldots, t_k$ which evaluate to $u_1, \ldots, u_k \in U$ and a k-ary function symbol f, the term $f(t_1, \ldots, t_k)$ evaluates to $I(f)(u_1, \ldots, u_k)$.
- Similarly, a formula $P(t_1, \ldots, t_k)$ evaluates to $I(P)(u_1, \ldots, u_k)$ if t_i evaluates to u_i.
- Given terms t_1 and t_2 which evaluate to u_1 and u_2, the formula $t_1 = t_2$ evaluates to true if and only if $u_1 = u_2$.
- Formulas of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$, $\neg\varphi$ evaluate according to the truth tables for logical connectives.
- A formula $\exists x\varphi$ evaluates to true if there exists an evaluation μ' that differs from μ only in the value of x such that φ evaluates under μ' to true.
- A formula $\forall x\varphi$ evaluates to true if φ evaluates to true under all assignments μ' that differ from μ only in the value of x.

One sees that the truth values of $\exists x\varphi$ and $\forall x\varphi$ do not depend on the assignment value of the variable x. It follows that the truth values of closed formulas are independent of the variable assignments, hence determined by $M = (U, I)$ only.

Definition 1.13. *One says that a structure M* satisfies *a closed formula φ, if φ evaluates to true under M. In this case M is also called a* model *of φ, and one writes $M \vDash \varphi$.*

Definition 1.14. *A closed formula φ is called* satisfiable *if it has at least one model. A closed formula φ is called* valid *if $M \vDash \varphi$ for every structure M.*

1.5 Propositional logic inside predicate logic

Consider the signature without functions and with nullary predicates only:

$$\mathcal{P}:\ P_1(), P_2(), \ldots \tag{2}$$

Formulas in this signature contain no terms, because a term must occur as an argument of a predicate, but nullary predicates have no arguments.

We can introduce variables in the formulas only with quantifiers by writing something like $\forall x \exists y P \wedge Q$, but in any structure this formula evaluates in the same way as $P \wedge Q$. Thus the formulas in signature (2) look like propositional formulas with variable symbols P_i.

How does $P \wedge Q$ actually evaluate? By definition, a first-order structure (U, I) assigns to each nullary predicate P a truth value $I(P)$. Then the truth value of $P \wedge Q$ is $I(P) \wedge I(Q)$ (and the universe U has no significance). Similarly for every other formula: an evaluation with respect to interpretation I is the same as evaluation of a propositional formula with I viewed as valuation v.

Thus signature (2) realizes the propositional logic as a special case of the predicate logic. One can state this as follows.

Theorem 1.15. *A predicate formula in signature* (2) *is valid if and only if the corresponding propositional formula is a tautology.*

2 Proof theory

We will present Gentzen's sequent calculus for predicate logic. There are other proof theories, for example Hilbert-style systems. These theories are equivalent to each other, which means that a formula provable within one of them is also provable within any other. (Postfactum it follows from their soundness and completeness, but one can describe a transformation of Hilbert proof into a Gentzen proof and vice versa in a direct way.)

2.1 Substitutions

Let A be a predicate formula, and let t be a term. Assume that A contains a free variable x. Then we can define a new formula $A[t/x]$ obtained by substitution of t for all free occurrences of x. (We don't exclude the possibility that x also occurs bound in A.)

Intuitively, $A[t/x]$ should represent a special case of A, so that for example if A is true for all x, then $A[t/x]$ is valid. However, some substitutions do not have this property. Take for example

$$A = \exists y(y < x), \quad t = y.$$

Then we have $A[t/x] = \exists y(y < y)$. This is false in the universe of integers, although A was true.

Definition 2.1. *A term t is* free for *x in A if no variable of t becomes bound after the substitution of t for all free occurrences of x.*

In other words, when we are substituting t for free occurrences of x, we also want all parts of t to remain free in the resulting formula.

In order to determine the truth value of a non-closed formula A, one needs a structure $M = (U, I)$ and an assignment μ for all free variables in A (that is, μ is a map from the set of free variables in A to U). A formula A is called satisfiable (respectively, falsifiable) in M if there is an assignment μ such that $(M, \mu) \vDash A$ (respectively, $(M, \mu) \nvDash A$).

Lemma 2.2. *Assume that a term t is free for x in A. Then for every structure M the following holds:*

- *If $\forall x A$ is satisfiable in M, then $A[t/x]$ is satisfiable in M.*
- *If $\exists x A$ is falsifiable in M, then $A[t/x]$ is falsifiable in M.*

More specifically, consider substitution in A of a variable y for a free variable x.

Lemma 2.3. *Assume that a variable y is free for x in A and is not occurring freely in A. Then for every structure M the following holds:*

- *If $\exists x A$ is satisfiable in M, then $A[y/x]$ is satisfiable in M.*
- *If $\forall x A$ is falsifiable in M, then $A[y/x]$ is falsifiable in M.*

An example showing that y should not occur freely in A:

$$\exists x A = \exists x (P(x) \wedge \neg P(y)).$$

This formula is satisfiable in any structure where the interpretation of predicate P is not constant. But after the quantifier removal and substitution of y for x it becomes

$$A[y/x] = P(y) \wedge \neg P(y),$$

which is not satisfiable.

2.2 Inference rules

Gentzen system for the predicate logic without equality operates with sequents $\Gamma \vdash \Delta$, where Γ and Δ are sets of predicate formulas. As before, we are building a deduction tree with the root $\vdash A$, where A is a formula which we are trying to prove or disprove.

Definition 2.4. *A sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ is called* falsifiable *if there exists a first-order structure M and a variable assignment μ which simultaneously make all A_i true and all B_j false:*

$$(M, \mu) \vDash A_i \text{ for all } i, \quad (M, \mu) \nvDash B_j \text{ for all } j.$$

Axioms of the Gentzen system are, as before, sequents $\Gamma \vdash \Delta$ with some formula occurring on both sides: $A \in \Gamma \cap \Delta$. Axioms are valid (because no structure can at the same time satisfy and falsify A). Inference rules for the logical connectives $\wedge, \vee, \to, \neg$ are the same. There are four new rules involving the quantifiers.

$$(\forall \text{ left}): \frac{A[t/x], \forall x A, \Gamma \vdash \Delta}{\forall x A, \Gamma \vdash \Delta} \qquad (\forall \text{ right}): \frac{\Gamma \vdash A[y/x], \Delta}{\Gamma \vdash \forall x A, \Delta}$$

$$(\exists \text{ left}): \frac{A[y/x], \Gamma \vdash \Delta}{\exists x A, \Gamma \vdash \Delta} \qquad (\exists \text{ right}): \frac{\Gamma \vdash A[t/x], \exists x A, \Delta}{\Gamma \vdash \exists x A, \Delta}$$

Here t is any term free for x in A, and y is any variable free for x in A and not occurring freely in the conclusion (that is, in the sequent $\Gamma \vdash \forall x A, \Delta$ for the right $\forall$ rule and in the sequent $\exists x A, \Gamma \vdash \Delta$ for the left $\exists$ rule). In particular, one can substitute x for itself if x does not occur freely in Γ and Δ.

As before, a deduction tree is a proof tree if all of its leaves are axioms. A sequent is called provable if there is a proof tree with this sequent as a root.

Lemma 2.5. *For each of the inference rules, the conclusion is falsifiable in some structure M if and only if at least one of the premises is falsifiable in the structure M.*

Proof. For the inference rules for $\wedge, \vee, \to, \neg$ the argument simply invokes the truth tables.

For the inference rules involving quantifiers this follows from Lemmas 2.2 and 2.3. □

Theorem 2.6. *Provable formulas are valid. In other words, sequent calculus for predicate logic is sound.*

Proof. Follows from Lemma 2.5. □

Example 2.7. Let us prove the formula $\exists x(P \to Q(x)) \to (P \to \exists y Q(y))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \vdash P, \exists y Q(y) \qquad \dfrac{Q(z_1), P \vdash Q(z_1), \exists y Q(y)}{Q(z_1), P \vdash \exists y Q(y)}}{P, P \to Q(z_1) \vdash \exists y Q(y)}}{P \to Q(z_1) \vdash P \to \exists y Q(y)}}{\exists x(P \to Q(x)) \vdash P \to \exists y Q(y)}}{\vdash \exists x(P \to Q(x)) \to (P \to \exists y Q(y))}$$

At the second step we are introducing a new variable z_1 while applying the left $\exists$ inference rule. After application of the branching left $\to$ rule we

obtain an axiom on the left (with P on both sides of $\vdash$). On the right we are applying the right $\exists$ rule, where we can make a clever choice of a term to substitute for y. Choosing z_1/y as the substitution, we obtain two equal terms on both sides of $\vdash$, and the proof tree is finished.

Example 2.8. The following deduction tree allows us to find a counterexample for $\exists x(P \to Q(x)) \to (P \to \forall y Q(y))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{P \vdash P, \forall y Q(y) \qquad \dfrac{Q(z_1), P \vdash Q(z_2)}{Q(z_1), P \vdash \forall y Q(y)}}{P, P \to Q(z_1) \vdash \forall y Q(y)}}{P \to Q(z_1) \vdash P \to \forall y Q(y)}}{\exists x(P \to Q(x)) \vdash P \to \forall y Q(y)}}{\vdash \exists x(P \to Q(x)) \to (P \to \forall y Q(y))}$$

The leaf on the right has a counterexample: take a universe with two elements z_1 and z_2, evaluate the nullary predicate P to true, and the unary predicate Q to true on z_1 and false on z_2.

2.3 Completeness of the sequent calculus

Theorem 2.9. *The sequent calculus is complete: every valid formula is provable.*

We will not give a detailed proof of this theorem, but will describe a procedure which, for every closed predicate formula A, constructs a deduction tree with the root $\vdash A$ with one of the following two properties. The algorithm assumes that the signature does not contain functions; if it does, some modifications are needed.

- It is finite and each of its leaves is an axiom.
- It is finite and has a leaf from which a counterexample can be read off.
- It contains an infinite path producing a counterexample.

One cannot exclude the possibility of an infinite tree because there are formulas satisfied by all finite structures but falsified by some infinite structure (see the exercises).

Algorithm description. We assume that the signature contains no function symbols. Given a closed formula A, put the sequent $\vdash A$ at the root.

From the set of variables choose an infinite sequence of symbols $u_1, u_2, \ldots$, not occurring in A. (The set $U = \{u_1, u_2, \ldots\}$ or a finite initial segment of it will later be our universe.) During the algorithm, the variables from the sequence $u_1, u_2, \ldots$ will be *activated* in their natural order. At the beginning, only u_1 is active, the rest is not.

At every step of the algorithm we have some previously constructed deduction tree and are going to apply an inference rule to one of its leaves. Each inference rule consists in copying most of the formulas of the sequent and changing only one of them, called the *principal formula*. The type of the inference rule is determined uniquely by the principal formula: this is the rule for the outermost connective in the principal formula, and it is the left or the right rule according on which side of $\vdash$ the principal formula is situated. For the connectives of propositional logic the inference rules are unique. For the quantifiers, one needs to specify the substitutions. This is done as follows.

- Right $\forall$ or left $\exists$ rule: substitute the first inactive variable from the sequence $\{u_1, u_2, \ldots\}$.
- Left $\forall$ or right $\exists$ rule: substitute all active u_i not previously substituted into this formula.

That means, if there are several active variables not yet substituted into $\forall \vdash$ or $\vdash \exists$, then the corresponding inference rule is applied several times. We abbreviate this sequence of steps as

$$\frac{A[u_{k+1}/x], \ldots, A[u_l/x], \forall x A, \Gamma \vdash \Delta}{\forall x A, \Gamma \vdash \Delta}$$

(and similarly for $\exists x A$ on the right), where $u_{k+1}, \ldots, u_l$ are all active variables not substituted into $\forall x A$ at some previous step.

Let us note that if the signature contains function symbols, then in the left $\forall$ and right $\exists$ case one must substitute all possible terms with currently active variables (this is a finite but possibly quite large set).

The inference rules are applied in a breadth-first way: in one round we go over all leaves, and every non-atomic formula inside a leaf is used as a principal formula. More exactly, if $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ is a leaf sequent, then we first apply the inference rule with A_1 as the principal formula. After that we apply the inference rule with A_2 as the principal formula to all "newly grown" leaves. Then we work with A_3 in all new leaves, and so on. Only after we finished the work with the formula B_n, we proceed to the next leaf sequent.

A leaf is called closed if one of the following occurs:

- its sequent contains only atomic formulas;
- its sequent contains only atomic formulas and $\forall x$-formulas on the left or $\exists x$-formulas on the right for which all substitutions (of active variables, see the instructions above) have already been performed.

A closed leaf with only atomic formulas is either an axiom leaf or a counterexample leaf (see Example 2.8).

Example 2.10 illustrates the second possibility.

It can happen that after every round there are non-closed leaves. In this case the algorithm never terminates, and the sequent at its root is falsifiable. Namely, the tree will contain an infinite path, and a counterexample can be read off this path as illustrated in Example 2.11 below. □

Example 2.10. Disprove the formula $\forall x \exists y P(x, y)$.

$$\dfrac{\dfrac{\vdash P(u_2, u_1), P(u_2, u_2), \exists y P(u_2, y)}{\vdash \exists y P(u_2, y)}}{\vdash \forall x \exists y P(x, y)}$$

This tree is closed, and its only leaf describes a counterexample:

$$U = \{u_1, u_2\}, \quad I(P)(u_2, u_1) = I(P)(u_2, u_2) = \text{false}$$

(the value of $I(P)$ on (u_1, u_2) and (u_1, u_1) is inessential).

Example 2.11. Disprove the formula $\exists x \forall y P(x, y)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash P(u_1, u_2), P(u_2, u_3), \exists x \forall y P(x, y)}}{\vdash P(u_1, u_2), \forall y P(u_2, y), \exists x \forall y P(x, y)}}{\vdash P(u_1, u_2), \exists x \forall y P(x, y)}}{\vdash \forall y P(u_1, y), \exists x \forall y P(x, y)}}{\vdash \exists x \forall y P(x, y)}$$

The tree is infinite, and it is easy to read off a counterexample:

$$U = \{u_1, u_2, \dots\}, \quad I(P)(u_i, u_{i+1}) = \text{false for all } i.$$

(Again, the values of $I(P)$ on other pairs of arguments do not matter.)

Remark 2.12. It is important to work in a breadth-first way, otherwise one can obtain an infinite tree for a valid formula. This happens, for example, in the following deduction tree for the formula $(P \vee \neg P) \vee \exists x \forall y P(x, y)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash Q, \neg Q, P(u_1, u_2), P(u_2, u_3), \exists x \forall y P(x, y)}}{\dots}}{\vdash Q, \neg Q, \exists x \forall y P(x, y)}}{\vdash (Q \vee \neg Q) \vee \exists x \forall y P(x, y)}}{\vdash (Q \vee \neg Q) \vee \exists x \forall y P(x, y)}$$

Here we are neglecting the non-atomic formula $\neg Q$ and working with $\exists x \forall y P(x, y)$ only, which produces an infinite path from Example 2.11. Constructed in a breadth-first way, this tree will close with an axiom leaf with Q on both sides of $\vdash$.

For details see [8].

3 Gödel's incompleteness theorems

3.1 Theories and models

Very often, we will deal with closed predicate formulas. For brevity, a closed formula will be called a *sentence*.

Definition 3.1. *A (first-order)* theory *is any set of sentences of the predicate logic. These sentences are called* axioms *of the theory.*

Example 3.2. The theory consisting of the following two sentences:

$$\forall x(\neg P(x,x))$$
$$\forall x \forall y(P(x,y) \to P(y,x))$$

is called *first-order graph theory*. Any model of this theory is a graph: the universe corresponds to the vertex set, and the predicate interpretation to the edge set.

Definition 3.3. *A theory T is called* satisfiable *if it has at least one model, that is a structure which satisfies all sentences of the theory:*

$$M \vDash T \overset{\text{def}}{\iff} M \vDash A \text{ for all } A \in T.$$

Example 3.4. The theory $\{A, \neg A\}$, where A is any sentence, is unsatisfiable. Indeed, in every model one of the sentences $A, \neg A$ takes the true value, the other one the false value.

Example 3.5. Every model M defines a theory $T_M = \{A \mid M \vDash A\}$, the set of all sentences satisfied by the model M.

The theory T_M has the property that for every A either $A \in T_M$ or $\neg A \in T_M$.

Definition 3.6. *Two models M and M' are called* elementarily equivalent *(denoted $M \sim M'$) if $T_M = T_{M'}$, that is every sentence satisfied by one is also satisfied by the other.*

Definition 3.7. *A theory T is called* complete *if it is satisfiable and any two models satisfying it are elementarily equivalent:*

$$M \vDash T, M' \vDash T \Rightarrow M \sim M'.$$

Example 3.8. First-order graph theory 3.2 is satisfiable but incomplete. The sentence $\exists x \exists y(x \neq y)$ is false in the model with one-element universe (graph with a single vertex) and true in any other model.

Remark 3.9. One can show that two finite models are elementarily equivalent if and only if the corresponding graphs are isomorphic. But this is false for infinite models: a doubly infinite path is elementarily equivalent to the union of two of its copies.

The connectivity property of a graph cannot be written as a sentence of the first-order logic. One can express it with the help of quantifiers over subsets of the universe, which belong already to the second-order logic.

Definition 3.10. *A sentence A is said to be a* semantic consequence *of a theory T (denoted $T \vDash A$) if every model of T satisfies A:*

$$T \vDash A \overset{\text{def}}{\Longleftrightarrow} \text{if } M \vDash T, \text{ then } M \vDash A.$$

Example 3.11. A semantic consequence of an empty theory is a valid sentence (one which is true in all structures). If A is valid, then we write $\vDash A$.

Lemma 3.12. *A theory T is complete if and only if it is satisfiable and for every sentence A one has either $T \vDash A$ or $T \vDash \neg A$.*

Proof. Assume T is complete, and let M be any of its models. For every sentence A, we have either $M \vDash A$ or $M \vDash \neg A$. Due to the completeness of T this implies $T \vDash A$, respectively $T \vDash \neg A$.

Assume T is incomplete. Then there are two models M and M' whose sets of satisfied formulas differ. Without loss of generality, let $M \vDash A$ and $M' \nvDash A$. Then neither $T \vDash A$ nor $T \vDash \neg A$ hold. □

Exercise 3.1. For every model M, the theory T_M is complete.

The first-order theory of graphs is incomplete, and this is good: one has many different graphs with different properties. But if we want to know everything about the arithmetics of $\mathbb{N}$, the natural numbers, then we need a complete theory. The theory should contain some axioms for addition and multiplication and maybe some other axioms, so that every sentence has a well-defined truth value (that is, all structures that satisfy our axioms assign to it the same value). For example, the question "Can every integer greater than 2 be expressed as the sum of two primes?" should have a definite answer.

One might use the result of the above exercise: if we assume that $\mathbb{N}$ is something objectively real, then there is the corresponding complete theory $T_{\mathbb{N}}$. But this theory has "too many" axioms, and we have no explicit description for it, apart from saying "what is true, is true". Gödel's incompleteness theorem says that there is no complete theory of $\mathbb{N}$ such that for every sentence one can decide in finite time is it an axiom of the theory or not.

Before we say more about the incompleteness theorem, we must discuss proofs, a constructive way to derive consequences of a theory.

3.2 Models and proofs: Gödel's completeness theorem

Definition 3.13. *A sentence A is said to be a* syntactic consequence *of a theory T (denoted $T \vdash A$) if there is a deduction tree with $\Gamma \vdash A$ at the root, where $\Gamma \subset T$ is any finite subset, and logical axioms at the leaves. One says that A is a* theorem *of theory T. The set of all theorems of T is denoted by* $\mathrm{Th}(T)$.

Exercise 3.2. If $T \vdash A$ and $T' \supset T$, then $T' \vdash A$.

Example 3.14. The definition of a group can easily be formulated as a first-order theory. It is also possible to state and to prove the following theorem: If all elements of a group have order two, then the group is commutative.

Lemma 3.15. *A syntactic consequence is a semantic consequence. That is, $T \vdash A$ implies $T \vDash A$.*

Proof. Proof by contradiction. Assume that there is a model M of T which does not satisfy A. Then this model falsifies the sequent $\Gamma \vdash A$, and therefore falsifies one of the leaves of the deduction tree. But this is impossible, thus every model that satisfies T also satisfies A. □

Theorem 3.16 (Gödel's completeness theorem)**.** *A semantic consequence of a theory is also its syntactic consequence:*

$$T \vDash A \Rightarrow T \vdash A.$$

In other words, every sentence which is true in a given theory has a formal proof.

A special case of this theorem is Theorem 2.9, which says that $\vDash A$ implies $\vdash A$ (here the theory is empty).

The completeness theorem can be formulated in a different way.

Definition 3.17. *A theory T is called (syntactically)* consistent *if there is no sentence A such that $T \vdash A$ and $T \vdash \neg A$.*

Lemma 3.18. *Every satisfiable theory is consistent.*

Proof. Proof by contraposition. Let T be an inconsistent theory: there is a sentence A such that $T \vdash A$ and $T \vdash \neg A$. Assume that T is satisfiable: $M \vDash T$. Then the soundness of the proof system implies that $M \vDash A$ and $M \vDash \neg A$, which is a contradiction. Thus every inconsistent theory is unsatisfiable. □

The inverse implication is also correct; it can be derived from the completeness theorem and vice versa by some manipulations with deduction trees. Therefore it is also called the completeness theorem.

Theorem 3.19 (Gödel's completeness theorem, second version)**.** *Every consistent theory is satisfiable.*

Corollary 3.20. *A theory T is complete if and only if it is consistent and syntactically complete, that is for every sentence A either $T \vdash A$ or $T \vdash \neg A$ but not both.*

3.3 Peano arithmetic

The signature consists of a nullary function 0, a unary function s (successor), two binary functions $+$ and $\cdot$. The equality predicate $=$. Axioms of PA:

1. $\forall x \neg(s(x) = 0)$
2. $\forall x \forall y(s(x) = s(y) \to x = y)$
3. $\forall x(x = 0 \vee \exists y(s(y) = x))$
4. $\forall x(x + 0 = x)$
5. $\forall x \forall y(x + s(y) = s(x + y))$
6. $\forall x(x \cdot 0 = 0)$
7. $\forall x \forall y(x \cdot s(y) = x \cdot y + x)$
8. $\forall \bar{y}\,((A(0, \bar{y}) \wedge \forall x(A(x, \bar{y}) \to A(s(x), \bar{y})) \to \forall x A(x, \bar{y}))$

The last item is the *induction schema*, that is it encodes infinitely many sentences. Here $\bar{y}$ denotes $y_1, \ldots, y_n$, and $\forall \bar{y}$ denotes $\forall y_1 \ldots \forall y_n$. The number n can be any non-negative integer, and A can be any formula with $n + 1$ free variables $x, y_1, \ldots, y_n$.

The theory consisting of the first seven axioms is called Robinson arithmetic, we will denote it by PA_0. It does not imply that the addition is commutative.

Since we assume that $\mathbb{N}$ is an objective reality and the operations with positive integers satisfy the above properties, theory PA is satisfiable and hence consistent.

3.4 Recursive functions and recursive sets

Definition 3.21. *A* recursive function *is a function $f\colon X \to \mathbb{N}$ defined on a subset X of $\mathbb{N}^p$ for some p which can be computed in finite time.*

The above definition is highly informal. The formal definition says that a recursive function is one that can be obtained from basic functions (constants, $x \mapsto x + 1$, projections) by finitely many operations such as composition, primitive recursion (which allows to construct $f(x, y) = x + y$,

for example), and minimization. (Functions which can be obtained without using minimization are called primitive recursive functions.) It can be shown that recursive functions are exactly those which can be computed by Turing machines. The input of a machine is a p-tuple of positive integers $x = (x_1, \dots, x_p)$; if $x \in X$, then the machine outputs $f(x)$, if $x \notin X$, then the machine outputs an error or never stops. (Without loss of generality, one can assume that for $x \notin X$ the machine never stops: if one has a machine that outputs an error, attach to it a machine which reacts on this error by starting an infinite loop.)

Definition 3.22. *A set $X \subset \mathbb{N}^p$ is called* recursive *if the function*

$$\mathbf{1}_X \colon \mathbb{N}^p \to \mathbb{N}, \quad \mathbf{1}_X(x) = \begin{cases} 1, & \text{if } x \in X \\ 0, & \text{if } x \notin X \end{cases}$$

is recursive.

In other words, a set X is recursive if there is an algorithm which for every input x terminates in finite time and tells whether x belongs to X or not.

Definition 3.23. *A set $X \subset \mathbb{N}^p$ is called* recursively enumerable *if the function*

$$\overline{\mathbf{1}}_X \colon X \to \mathbb{N}, \quad \overline{\mathbf{1}}_X(x) = 1 \text{ for all } x \in X$$

is recursive.

In other words, X is recursively enumerable if there is an algorithm which terminates only when the input x belongs to X.

Recursive sets are also called decidable, and recursively enumerable are called semi-decidable.

Example 3.24. The set of prime numbers is recursive. It is not easy to give an example of a recursively enumerable but not recursive set.

Lemma 3.25. *A set $X \subset \mathbb{N}^p$ is recursive if and only if both X and $\mathbb{N}^p \setminus X$ are recursively enumerable.*

Proof. If we have an algorithm for computing $\mathbf{1}_X$, then modify it by going into an endless loop if the output is 0. This computes the function $\overline{\mathbf{1}}_X$. The function $\overline{\mathbf{1}}_{\mathbb{N}^p \setminus X}$ is computed similarly.

If one has an algorithm which stops only for inputs $x \in X$ and an algorithm which stops only for inputs $x \notin X$, then run them parallelly and interpret the output of the first algorithm as 1, and the output of the second algorithm as 0. This allows us to compute the function $\mathbf{1}_X$. □

Definition 3.26. *Let B be a formula of the predicate logic in the signature of Peano arithmetic with exactly one free variable y. One says that B* represents *a subset $Y \subset \mathbb{N}$ if*

$$n \in Y \text{ if and only if } PA_0 \vdash B[\underline{n}/y],$$

where $\underline{n} = \underbrace{s \circ s \circ \cdots \circ s}_{n \text{ times}}(0)$.

Theorem 3.27. *Every recursive subset of $\mathbb{N}$ is representable.*

3.5 Gödel's incompleteness theorems

Denote by L the language of PA (that is, the set of all formulas in the signature of PA). There is an injective map

$$\sharp\colon L \to \mathbb{N}$$

called *Gödel numbering* such that its image $\sharp(L) \subset \mathbb{N}$ is a recursive set. (Such a map can be constructed by associating to every symbol a number (similarly ASCII encoding) and then encoding a sequence of numbers by a single number for example through $(k_1, \ldots, k_n) \mapsto p_1^{k_1} \cdots p_n^{k_n}$, where p_i is the i-th prime number.) That the image of this map is recursive means that there is an algorithm which determines for any number $m \in \mathbb{N}$ whether it encodes a well-formed formula. It is clear how to reconstruct a sequence of symbols from its Gödel number, and it is clear how to check whether a sequence of symbols is a formula.

The Gödel numbering can be extended to sequences of formulas:

$$\sharp\sharp\colon L^* \to \mathbb{N}.$$

A proof of a formula can be represented as a sequence of formulas (this is so in the Hilbert proof system, for the Gentzen system one has to agree how to transform a tree into a list; this is doable). Thus every proof has a Gödel number as well, with different numbers corresponding to different proofs (and some numbers not corresponding to any proof).

Gödel numbering allows to speak about recursive sets of formulas.

Let $\mathrm{Th}(T)$ denote the set of all theorems of theory T.

Definition 3.28. *A theory T is called* recursive *if the set*

$$\sharp T = \{\sharp A \mid A \in T\} \subset \mathbb{N}$$

is recursive.

A theory T is called decidable *if the set*

$$\sharp\mathrm{Th}(T) = \{\sharp A \mid A \in \mathrm{Th}(T)\} \subset \mathbb{N}$$

is recursive.

For a recursive theory there is an algorithm which, for any given formula, tells us in finite time whether this formula is an axiom of our theory or not. This is a very reasonable condition for a theory: one should be able to distinguish axioms from non-axioms. For a decidable theory there is an algorithm which, for any given formula, tells us whether this formula can be derived from the axioms or not.

Lemma 3.29. *If theory T is recursive, then its set of theorems* $\mathrm{Th}(T)$ *is recursively enumerable.*

Intuitively, if we can distinguish axioms from non-axioms, then the set of proofs is recursive. From certain properties of recursive sets it follows that the set of last terms of proofs is recursively enumerable.

Theorem 3.30. *A complete and recursive theory is decidable.*

Proof. By the lemma, the set $\mathrm{Th}(T)$ is recursively enumerable, so that it remains to show that its complement is recursively enumerable. But in a complete theory, $T \nvdash A$ if and only if $T \vdash \neg A$. Thus, if we use $\neg A$ as an input for the algorithm which stops if and only if its input is a theorem, then this algorithm will stop if and only if A is not a theorem. □

Theorem 3.31. *Let $T \supset PA_0$ be a consistent theory. Then T is undecidable. In particular, PA_0 and PA are undecidable.*

Observe that this theorem together with the previous one immediately imply

Theorem 3.32 (First Gödel's incompleteness theorem)**.** *Every consistent and recursive theory which contains PA_0 is incomplete.*

Proof of Theorem 3.31. Assume that T is decidable. Consider the set

$$X = \{(m, n) \mid m = \sharp A(\cdot), T \vdash A[\underline{n}/\cdot]\} \subset \mathbb{N}^2.$$

(Here $A(\cdot)$ means that A has one free variable, and $A[t/\cdot]$ means substitution of term t for this variable.) Since T is decidable, the set X is recursive (there is an algorithm which decides whether $(m, n) \in X$ or not).

Now let

$$Y = \{n \mid (n, n) \notin X\}.$$

Clearly, since X is recursive, so is Y. By Theorem 3.27, the set Y is representable. That is, there is a formula $B(y)$ such that

$$n \in Y \Rightarrow PA_0 \vdash B[\underline{n}/y]$$
$$n \notin Y \Rightarrow PA_0 \vdash \neg B[\underline{n}/y]$$

Since $T \supset PA_0$, for every C such that $PA_0 \vdash C$ we also have $T \vdash C$.

Now let $n = \sharp B(y)$. Let us ask ourselves if $n \in Y$ or not. Assume $n \in Y$. By definition of Y, X, and n we have

$$n \in Y \Rightarrow (n, n) \notin X \Rightarrow T \nvdash B[\underline{n}/y].$$

On the other hand, by definition of $B(y)$ we have

$$n \in Y \Rightarrow T \vdash B[\underline{n}/y],$$

a contradiction. Now, if we assume that $n \notin Y$, then we derive similarly that $T \vdash B[\underline{n}/y]$ and $T \vdash \neg B[\underline{n}/y]$, an inconsistency. Thus the decidability assumption contradicts the consistency assumption, and the theorem is proved. □

Corollary 3.33. *The set $V = \{A \mid A \text{ is valid}\}$ is recursively enumerable but not recursive.*

Proof. The set V is the set of theorems of the empty theory. Since the empty theory is recursive, V is recursively enumerable.

Denote by B the conjunction of all seven axioms of PA_0. If V is recursive, then for any sentence A one can decide in finite time whether $B \to A$ belongs to V or not. But an answer to this question is equivalent to an answer to the question whether $A \in \mathrm{Th}(PA_0)$ or not. This contradicts the fact that PA_0 is undecidable, thus V cannot be recursive. □

For details see [4, 5].

Chapter V

Combinatorics II

Additional reading for this part of the course: [1].

1 Linear recursive sequences

1.1 Fibonacci sequence and Binet formula

The *Fibonacci sequence* is defined as follows:

$$a_0 = 0, \quad a_1 = 1, \quad a_n = a_{n-1} + a_{n-2} \text{ for } n \geq 2.$$

Here are the first several terms:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ \ldots$$

Theorem 1.1 (Binet)**.** *The n-th Fibonacci number is equal to*

$$a_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

At the first sight it is not clear at all that this formula produces a rational and even integer number.

A direct consequence of the Binet formula is that the Fibonacci sequence grows as a geometric progression with ratio equal to the golden ratio

$$\tau = \frac{1+\sqrt{5}}{2} = 1,618\ldots.$$

Indeed, the second summand is

$$\left(\frac{1-\sqrt{5}}{2}\right)^n = (-0,618\ldots)^n \to 0 \text{ as } n \to \infty,$$

which implies that

$$a_n \sim \frac{\tau^n}{\sqrt{5}},$$

and even more exactly

$$a_n = \left\lfloor \frac{\tau^n}{\sqrt{5}} \right\rceil,$$

where $\lfloor x \rceil$ denotes the closest integer to the number x.

1.2 Linear recursive sequences of order 2

We will prove not only the Binet formula, but show that similar formulas hold for all sequences defined similarly to the Fibonacci sequence.

Definition 1.2. *A sequence a_n, $n = 0, 1, \ldots$, is called a* linear recursive sequence of order 2 *if it satisfies the relation*

$$a_n = ra_{n-1} + sa_{n-2} \text{ for all } n \geq 2 \tag{1}$$

for some constant coefficients r and s, where $s \neq 0$.

Such a sequence is uniquely determined by the values of a_0 and a_1.

Definition 1.3. *The* characteristic polynomial *of the sequence* (1) *is the quadratic polynomial*

$$x^2 - rx - s.$$

Theorem 1.4. *1. If the characteristic polynomial of* (1) *has two different roots λ_1, λ_2, then one has*

$$a_n = c_1\lambda_1^n + c_2\lambda_2^n$$

for some $c_1, c_2 \in \mathbb{R}$.

2. If the characteristic polynomial of (1) *has one (double) root λ, then one has*

$$a_n = c_1\lambda^n + c_2 n\lambda^n$$

for some $c_1, c_2 \in \mathbb{R}$.

Proof. Part 1. Let us show that each of the two geometric progressions λ_1^n and λ_2^n satisfies the recurrence relation (1). Indeed, one has

$$\lambda_i^n - r\lambda_i^{n-1} - s\lambda_i^{n-2} = \lambda_i^{n-2}(\lambda_i^2 - r\lambda_i - s) = 0,$$

which implies $\lambda_i^n = r\lambda_i^{n-1} + s\lambda_i^{n-2}$. It follows that for any constant coefficients c_1 and c_2 the sequence $c_1\lambda_1^n + c_2\lambda_2^n$ also satisfies the relation (1).

Let us show that the coefficients c_1 and c_2 can be chosen so that

$$\begin{aligned} c_1 + c_2 &= a_0 \\ c_1\lambda_1 + c_2\lambda_2 &= a_1. \end{aligned}$$

This is a system of two linear equations for two unknowns c_1 and c_2, which has a (unique) solution because the matrix of the system has a non-zero determinant:

$$\begin{vmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{vmatrix} = \lambda_2 - \lambda_1 \neq 0.$$

Once the sequences $c_1\lambda_1^n + c_2\lambda_2^n$ and a_n coincide in the first two terms, they coincide everywhere. This proves the first part of the Theorem.

Part 2. The proof is similar, but as the basis sequences we take λ^n and $n\lambda^n$. The first of them satisfies the linear recurrence by the same reason as in Part 1. To check the recurrence for the second sequence, observe that λ being the double root of the characteristic polynomial means that

$$x^2 - rx - s = (x - \lambda)^2 \Rightarrow r = 2\lambda,\ s = -\lambda^2.$$

Thus we have

$$\begin{aligned} n\lambda^n - r(n-1)\lambda^{n-1} - s(n-2)\lambda^{n-2} &= \lambda^{n-2}(n\lambda^2 - 2\lambda(n-1)\lambda + \lambda^2(n-2)) \\ &= \lambda^n(n - 2(n-1) + (n-2)) = 0. \end{aligned}$$

As next one has to find the coefficients c_1 and c_2 which make the linear combination $c_1\lambda^n + c_2 n\lambda^n$ to coincide with the sequence a_n in the first two terms:

$$\begin{aligned} c_1 &= a_0 \\ c_1\lambda + c_2\lambda &= a_1. \end{aligned}$$

Clearly, this linear system has a solution. □

Let us apply the algorithm from the above proof to find an explicit formula for Fibonacci numbers.

The characteristic polynomial is $\lambda^2 - \lambda - 1$. Its roots are

$$\lambda_1 = \frac{1+\sqrt{5}}{2}, \quad \lambda_2 = \frac{1-\sqrt{5}}{2}. \tag{2}$$

If we start the Fibonacci sequence from the zeroth term so that the recurrence relation holds between a_0, a_1, a_2 as well, then we must put $a_0 = 0$. Thus the coefficients c_1 and c_2 are found from the system

$$\begin{aligned} c_1 + c_2 &= 0 \\ c_1\lambda_1 + c_2\lambda_2 &= 1. \end{aligned}$$

From the first equation one has $c_2 = -c_1$. Substituting this into the second equation one obtains

$$c_1 = \frac{1}{\lambda_1 - \lambda_2} = \frac{1}{\sqrt{5}}.$$

The result is the formula of Binet:

$$a_n = \frac{1}{\sqrt{5}}\lambda_1^n - \frac{1}{\sqrt{5}}\lambda_2^n.$$

1.3 Linear recursive sequences of higher order

Linear recursive sequences of higher orders are defined similarly and can be handled in a similar way.

Definition 1.5. *A sequence $a_0, a_1, a_2, \ldots$ is called a* linear recursive sequence of order k *if it satisfies the relation*

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_k a_{n-k} \tag{3}$$

for all $n \geq k$ for some constants $r_1, \ldots, r_k$, where $r_k \neq 0$.

A linear recursive sequence of order k is completely determined by the values of its first k terms.

Definition 1.6. *The* characteristic polynomial *of the sequence* (3) *is*

$$P(x) = x^k - r_1 x^{k-1} - r_2 x^{k-2} - \cdots - r_k.$$

Theorem 1.7. *Let $P(x) = (x - \lambda_1)^{k_1} \cdots (x - \lambda_m)^{k_m}$ be the complete factorization of the characteristic polynomial of a linear recursive sequence a_n. Then the sequence a_n is a linear combination of the sequences $n^j \lambda_i^n$, $0 \leq j \leq k_i - 1$.*

We don't give the proof, which is similar to the case of sequences of order 2. If all roots of $P(x)$ are distinct, then the coefficients c_i in

$$a_n = c_1 \lambda_1^n + \cdots + c_k \lambda_k^n$$

are found by solving a system of k linear equations with k unknowns. If there are multiple roots, some work should be done in order to prove that the sequences $n^j \lambda_i^n$ satisfy the recursive relation (3).

1.4 The case of complex roots

Back to the case of a recursive sequence of order 2, what happens if the characteristic polynomial

$$x^2 - rx - s$$

has no real roots? This happens when the discriminant $D = r^2 + 4s$ is negative. Then it has two complex roots

$$\lambda_{1,2} = \frac{r \pm \sqrt{D}}{2},$$

and all arguments remain valid: the geometric progressions λ_i^n solve the recurrence and every sequence satisfying the recurrence can be written as a linear combination of these two progressions.

The two complex roots are conjugate to each other:

$$\lambda_1 = \lambda, \quad \lambda_2 = \overline{\lambda}.$$

A linear combination of the progressions λ^n and $\overline{\lambda}^n$ takes real values only if the coefficients are conjugate:

$$a_n = c\lambda^n + \overline{c}\overline{\lambda}^n.$$

Let us write the roots in the exponential form:

$$\lambda = \rho e^{i\varphi} = \rho(\cos\varphi + i\sin\varphi), \quad \overline{\lambda} = \rho e^{-i\varphi} = \rho(\cos\varphi - i\sin\varphi).$$

Then we have

$$\begin{aligned} c\lambda^n + \overline{c}\overline{\lambda}^n &= \rho^n(ce^{in\varphi} + \overline{c}e^{-in\varphi}) \\ &= \rho^n(c(\cos n\varphi + i\sin n\varphi) + \overline{c}(\cos n\varphi - i\sin n\varphi)) \\ &= \rho^n((c+\overline{c})\cos n\varphi + i(c-\overline{c})\sin n\varphi) \end{aligned}$$

That is, every recursive sequence is a linear combination with real coefficients of the following two sequences:

$$\rho^n \cos n\varphi \quad \text{and} \quad \rho^n \sin n\varphi.$$

Consider the special case $\rho = 1$. Then the recursive relation has the form

$$a_n = ra_{n-1} - a_{n-2},$$

where $r = 2\cos\varphi$. If φ is a rational multiple of π, then this sequence will be periodic independently of the initial values a_0, a_1. For example, this is the case of the recurrence

$$a_n = a_{n-1} - a_{n-2}$$

Here $\varphi = \frac{\pi}{3}$, and the sequence will have period 6. (This is also easy to check by iterating the recurrence relation and writing $a_2, a_3, \ldots$ in terms of a_0 and a_1.) If φ is not a rational multiple of π, then the sequence will not be periodic. It will fill densely some interval. This is the case of the recurrence

$$a_n = \frac{a_{n-1}}{2} - a_{n-2}.$$

1.5 An application of the Binet formula

Theorem 1.8. *The square of every Fibonacci number differs from the product of its left and right neighbors by* 1. *For example,*

$$3^2 = 2 \cdot 5 - 1, \quad 5^2 = 3 \cdot 8 + 1, \quad 8^2 = 5 \cdot 13 - 1.$$

This and many other relations between Fibonacci numbers can be proved by induction, sometimes in a not very straightforward way. When the Binet formula is used, the proof consists of simple algebraic manipulations.

Proof. We have $a_n = \frac{1}{\sqrt{5}}(\lambda_1^n - \lambda_2^n)$ with λ_i as in (2). Taking into account that $\lambda_1\lambda_2 = -1$, one computes

$$a_n^2 = \frac{1}{5}(\lambda_1^{2n} - 2\lambda_1^n\lambda_2^n + \lambda_2^{2n}) = \frac{1}{5}(\lambda_1^{2n} + \lambda_2^{2n} - 2(-1)^n).$$

On the other hand,

$$\begin{aligned} a_{n-1}a_{n+1} &= \frac{1}{5}(\lambda_1^{n-1} - \lambda_2^{n-1})(\lambda_1^{n+1} - \lambda_2^{n+1}) \\ &= \frac{1}{5}(\lambda_1^{2n} - \lambda_1^{n-1}\lambda_2^{n+1} - \lambda_2^{n-1}\lambda_1^{n+1} + \lambda_2^{2n}) \\ &= \frac{1}{5}(\lambda_1^{2n} + \lambda_2^{2n} - \lambda_1^{n-1}\lambda_2^{n-1}(\lambda_1^2 + \lambda_2^2)). \end{aligned}$$

One computes

$$\lambda_1^2 + \lambda_2^2 = \frac{1 + 2\sqrt{5} + 5}{4} + \frac{1 - 2\sqrt{5} + 5}{4} = 3,$$

which implies

$$a_{n-1}a_{n+1} = \frac{1}{5}(\lambda_1^{2n} + \lambda_2^{2n} - 3(-1)^{n-1}) = a_n^2 + (-1)^n.$$

□

2 Generating functions

2.1 Fibonacci again

Take the Fibonacci sequence

$$(a_0, a_1, a_2, a_3, \ldots) = (0, 1, 1, 2, \ldots)$$

and write a power series

$$A(x) = \sum_{k=0}^{\infty} a_k x^k = a_0 + a_1 x + a_2 x^2 + \cdots .$$

Because of

$$\begin{aligned} xA(x) &= a_0x + a_1x^2 + a_2x^3 + \cdots \\ x^2A(x) &= a_0x^2 + a_1x^3 + \cdots \end{aligned}$$

we have

$$\begin{aligned} xA(x) + x^2A(x) &= a_0x + (a_1 + a_0)x^2 + (a_2 + a_1)x^3 + \cdots \\ &= a_2x^2 + a_3x^3 + \cdots = A(x) - a_0 - a_1x = A(x) - x. \end{aligned}$$

This implies

$$A(x)(1 - x - x^2) = x \Rightarrow A(x) = \frac{x}{1 - x - x^2}.$$

(At the moment it is not clear what this equation means and why can we perform with the power series $A(x)$ the above algebraic manipulations. A justification will be given later. Now let us continue to do whatever looks reasonable.)

We claim that there are real numbers A, B such that

$$\frac{x}{1 - x - x^2} = \frac{x}{(1 - \lambda_1x)(1 - \lambda_2x)} = \frac{A}{1 - \lambda_1x} + \frac{B}{1 - \lambda_2x}$$

Here $\lambda_1 = \frac{1+\sqrt{5}}{2}$, and $\lambda_2 = \frac{1-\sqrt{5}}{2}$.

The numbers A and B can be found by a smart guess:

$$\begin{aligned} \frac{x}{(1 - \lambda_1x)(1 - \lambda_2x)} &= \frac{1}{\lambda_1 - \lambda_2} \frac{(1 - \lambda_2x) - (1 - \lambda_1x)}{(1 - \lambda_1x)(1 - \lambda_2x)} \\ &= \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \lambda_1x} - \frac{1}{1 - \lambda_2x} \right) \end{aligned}$$

Or they can be found by writing down a system of linear equations:

$$\begin{aligned} \frac{x}{(1 - \lambda_1x)(1 - \lambda_2x)} &= \frac{A}{1 - \lambda_1x} + \frac{B}{1 - \lambda_2x} \\ &= \frac{A(1 - \lambda_2x) + B(1 - \lambda_1x)}{(1 - \lambda_1x)(1 - \lambda_2x)} = \frac{(A + B) - (A\lambda_2 + B\lambda_1)x}{(1 - \lambda_1x)(1 - \lambda_2x)} \\ &\Rightarrow \begin{cases} A + B = 0 \\ A\lambda_2 + B\lambda_1 = -1 \end{cases} \end{aligned}$$

Anyway, we have

$$A(x) = \frac{x}{1 - x - x^2} = \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \lambda_1x} - \frac{1}{1 - \lambda_2x} \right).$$

Now, from the formula for geometric progression

$$\frac{1}{1-y} = 1 + y + y^2 + y^3 + \cdots$$

by substituting $y = \lambda x$ we get

$$\frac{1}{1-\lambda x} = 1 + \lambda x + \lambda^2 x^2 + \lambda^3 x^3 + \cdots .$$

Thus we have

$$A(x) = \frac{1}{\sqrt{5}} \left(\sum_{k=0}^{\infty} \lambda_1^k x^k - \sum_{k=0}^{\infty} \lambda_2^k x^k \right) = \sum_{k=0}^{\infty} \frac{\lambda_1^k - \lambda_2^k}{\sqrt{5}} x^k,$$

which means that

$$a_k = \frac{\lambda_1^k - \lambda_2^k}{\sqrt{5}},$$

the Binet formula again.

2.2 Operations with formal power series

There are two ways of interpreting the calculations we made above. The first approach is by viewing x as a real number close to 0 such that the power series $\sum_{k=0}^{\infty} a_k x^k$ converges. Then some theorems from calculus ensure that all our operations were correct. The second approach is to deal with $\sum_{k=0}^{\infty} a_k x^k$ as a formal expression, to define operations with such expressions, and to show that these operations satisfy all of the usual algebraic properties. We choose the second approach: *formal power series.*

Definition 2.1. *Let* $A(x) = \sum_{k=0}^{\infty} a_k x^k$ *and* $B(x) = \sum_{k=0}^{\infty} b_k x^k$ *be two formal power series. Their* sum *is the formal power series*

$$A(x) + B(x) = \sum_{k=0}^{\infty} (a_k + b_k) x^k,$$

and their product *is the formal power series*

$$A(x)B(x) = \sum_{k=0}^{\infty} c_k x^k,$$

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}.$$

The formula for the product comes from expanding the brackets in

$$(a_0 + a_1x + a_2x^2 + \cdots)(b_0 + b_1x + b_2x^2 + \cdots).$$

It can easily be checked that the sum and the product defined above satisfy the usual rules, such as

$$A(x)(B(x) + C(x)) = A(x)B(x) + A(x)C(x)$$

etc.

Definition 2.2. *A formal power series $B(x)$ is called (a multiplicative)* inverse *of $A(x)$ if*

$$A(x)B(x) = 1 = 1 + 0 \cdot x + 0 \cdot x^2 + \cdots .$$

Lemma 2.3. *Every formal power series $A(x) = \sum_{k=0}^{\infty} a_k x^k$ such that $a_0 \neq 0$ has a unique multiplicative inverse.*

Proof. The equation $A(x)B(x) = 1$ consists of an infinite sequence of equations

$$\begin{gathered} a_0b_0 = 1 \\ a_0b_1 + a_1b_0 = 0 \\ a_0b_2 + a_1b_1 + a_2b_0 = 0 \\ \cdots \end{gathered}$$

with unknowns $b_0, b_1, \ldots$. The first equation implies $b_0 = \frac{1}{a_0}$ (which is defined because $a_0 \neq 0$). Knowing b_0 we can express b_1 from the second equation:

$$b_1 = -\frac{a_1b_0}{a_0}$$

and continue in the same spirit, because $(k+1)$-st equation can be solved for b_k:

$$b_k = -\frac{1}{a_0}\sum_{i=1}^{k} a_i b_{k-i}.$$

This shows that the inverse series $B(x)$ exists and is unique. □

We denote the inverse series to $A(x)$ by $(A(x))^{-1}$ or $\frac{1}{A(x)}$.

Example 2.4. One has

$$(1-x)^{-1} = 1 + x + x^2 + \cdots ,$$

as the brackets expansion

$$(1-x)(1 + x + x^2 + \cdots) = 1 + (x - x) + (x^2 - x^2) + \cdots = 1$$

shows.

Definition 2.5. *For* $A(x) = \sum_{k=0}^{\infty} a_k x^k$ *and* $B(x) = \sum_{k=0}^{\infty} b_k x^k$ *define the* composition *as*

$$A(B(x)) = \sum_{k=0}^{\infty} a_k (B(x))^k.$$

The composition is not always well-defined. Indeed, we have

$$A(B(x)) = a_0 + a_1(b_0 + b_1 x + b_2 x^2 + \cdots) + a_2(b_0 + b_1 x + b_2 x^2 + \cdots)^2 + \cdots .$$

In particular, the constant term is an infinite sum $a_0 + a_1 b_0 + a_2 b_0^2 + \cdots$, which is not good. However, if $b_0 = 0$, then we have

$$a_k(B(x))^k = a_k(b_1 x + b_2 x^2 + \cdots)^k = a_k b_1^k x^k + \text{ higher order terms},$$

so that the coefficient at x^k in $A(B(x))$ is a finite expression in a_i, b_j. Thus we arrive to the following conclusion.

Lemma 2.6. *The composition* $A(B(x))$ *is well-defined if* $b_0 = 0$.

(Note that it is also well-defined for any value of b_0 provided that $A(x)$ is a polynomial, but we are not going to need this.)

The composition is compatible with sum and product:

$$\text{if } C(x) = A(x) + B(x), \text{ then } C(D(x)) = A(D(x)) + B(D(x)),$$
$$\text{if } C(x) = A(x)B(x), \text{ then } C(D(x)) = A(D(x))B(D(x)).$$

This implies in particular that for every formal power series $B(x)$ with $b_0 = 0$ one has

$$(1 - B(x))^{-1} = 1 + B(x) + (B(x))^2 + \cdots .$$

Example 2.7. One has

$$(1 + x)^{-1} = 1 - x + x^2 - \cdots ,$$
$$(1 - \lambda x)^{-1} = 1 + \lambda x + \lambda^2 x^2 + \cdots ,$$
$$(1 - x^2)^{-1} = 1 + x^2 + x^4 + \cdots .$$

The definition and lemmas of this section give a meaning to the manipulations done in Section 2.1.

2.3 Linear recursive sequences and partial fraction decomposition

Definition 2.8. *The formal power series*

$$a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{k=0}^{\infty} a_k x^k$$

is called the generating function *of the sequence* $a_0, a_1, a_2, \ldots$.

Theorem 2.9. *The generating function of a sequence satisfying the linear recursion*

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_k a_{n-k}, \quad n \geq k$$

can be written in the form $\frac{B(x)}{\overline{P}(x)}$*, where* $B(x)$ *is some polynomial, and*

$$\overline{P}(x) = 1 - r_1 x - r_2 x^2 - \cdots - r_k x^k.$$

Proof. Exercise. □

Note that $\overline{P}(x)$ is related to the characteristic polynomial $P(x)$ through

$$\overline{P}(x) = x^k P\left(\frac{1}{x}\right).$$

It follows that the roots of the polynomial $\overline{P}(x)$ are reciprocals of the roots of $P(x)$. More exactly, if $P(x) = (x - \lambda_1)^{k_1} \cdots (x - \lambda_m)^{k_m}$, then

$$\overline{P}(x) = (1 - \lambda_1 x)^{k_1} \cdots (1 - \lambda_m x)^{k_m}.$$

The following theorem generalizes our representation of the fraction $\frac{x}{1-x-x^2}$ as a sum of two simpler fractions.

Theorem 2.10 (Partial fraction decomposition)**.** *1. Let* $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$*, and let* $B(x)$ *be a polynomial of degree* $< k$*. Then there are real numbers* $c_1, \ldots, c_k$ *such that*

$$\frac{B(x)}{(1 - \lambda_1 x) \cdots (1 - \lambda_k x)} = \frac{c_1}{1 - \lambda_1 x} + \cdots + \frac{c_k}{1 - \lambda_k x}$$

2. Let $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$*,* $k_1, \ldots, k_m \in \mathbb{N}$ *such that* $k_1 + \cdots + k_m = k$*, and let* $B(x)$ *be a polynomial of degree* $< k$*. Then there are real numbers* c_{ij}*,* $1 \leq i \leq m$*,* $1 \leq j \leq k_i$ *such that*

$$\frac{B(x)}{(1 - \lambda_1 x)^{k_1} \cdots (1 - \lambda_m x)^{k_m}} = \sum_{i=1}^{m} \sum_{j=1}^{k_i} \frac{c_{ij}}{(1 - \lambda_i x)^j}$$

Remark 2.11. Note that if $\deg B \geq k$, then we can divide $B(x)$ by the denominator with remainder:

$$B(x) = Q(x)(x - \lambda_1)^{k_1} \cdots (x - \lambda_m)^{k_m} + R(x), \quad \deg R < k,$$

and then apply the theorem to a fraction with R in place of B.

We don't give a proof of the above theorem, but there is a simple algorithm that allows to compute the coefficients c_i, respectively c_{ij}. Bring the equation to a common denominator, it becomes an equation between the numerators. The numerators are polynomials; they are equal if and only if their corresponding coefficients are equal. This yields a system of linear equations on the unknowns c_i (respectively c_{ij}).

Because of

$$\frac{1}{1-\lambda_i x} = 1 + \lambda_i x + \lambda_i^2 x^2 + \cdots$$

the first part of the above theorem implies that every linear recursive sequence has the form $a_n = \sum_i c_i \lambda_i^n$, if the characteristic polynomial has only simple roots λ_i. In the case of multiple roots we need to represent the quotient $\frac{1}{(1-\lambda x)^j}$ as a formal power series.

2.4 Generalized binomial theorem

Theorem 2.12. *For every positive integer n one has*

$$(1+x)^{-n} = \sum_{k=0}^{\infty} \binom{-n}{k} x^k, \tag{4}$$

where

$$\binom{-n}{k} = \frac{-n \cdot (-n-1) \cdot \ldots \cdot (-n-k+1)}{k!}.$$

What do we mean by $(1+x)^{-n}$? This is a power series $A(x)$ such that $A(x)(1+x)^n = 1$.

Remark 2.13. Observe that

$$\begin{aligned}\binom{-n}{k} &= (-1)^k \frac{(n+k-1)(n+k-2) \cdot \ldots \cdot n}{k!} \\ &= (-1)^k \binom{n+k-1}{k} = (-1)^k \binom{n+k-1}{n-1}.\end{aligned}$$

Therefore, by substituting $-x$ instead of x one can rewrite (4) as

$$(1-x)^{-n} = \sum_{k=0}^{\infty} \binom{n+k-1}{n-1} x^k.$$

Theorem 2.12 will follow from a more general theorem below. Now let us just check it for a special case $n = 2$:

$$\begin{aligned}(1-x)^{-2} &= (1-x)^{-1}(1-x)^{-1} = (1+x+x^2+\cdots)(1+x+x^2+\cdots) \\ &= 1 + (x \cdot 1 + 1 \cdot x) + (x^2 + x \cdot x + x^2) + \cdots = 1 + 2x + 3x^2 + \cdots\end{aligned}$$

Definition 2.14. *For every $\alpha \in \mathbb{R}$ and every non-negative integer k define the* generalized binomial coefficient $\binom{\alpha}{k}$ *as*

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}.$$

For $k = 0$ the products in the denominator and in the numerator are empty, therefore we have $\binom{\alpha}{k} = \frac{1}{1} = 1$.

Theorem 2.15 (Vandermonde's identity)**.** *For every $\alpha, \beta \in \mathbb{R}$ the following holds:*

$$\binom{\alpha+\beta}{k} = \sum_{i=0}^{k} \binom{\alpha}{i}\binom{\beta}{k-i}.$$

A straightforward consequence of this is the following.

Corollary 2.16. *For every $\alpha \in \mathbb{R}$ put by definition*

$$(1+x)^\alpha = \sum_{k=0}^{\infty} \binom{\alpha}{k} x^k$$

(which for $\alpha \in \mathbb{N}$ agrees with the binomial formula, so that this definition does not override the usual definition of $(1+x)^n$). Then for every $\alpha, \beta \in \mathbb{R}$ the following identity between formal power series holds:

$$(1+x)^\alpha (1+x)^\beta = (1+x)^{\alpha+\beta}.$$

In particular, for every $p \in \mathbb{Z}$ and $q \in \mathbb{N}$ one has

$$\left((1+x)^{\frac{p}{q}}\right)^q = (1+x)^p.$$

It follows that one can extract q-th root from any formal power series with non-zero constant term.

Proof of Theorem 2.15. Step 1. If $\alpha = m$ and $\beta = n$ are positive integers, then

$$\binom{m+n}{k} = \sum_{i=0}^{k} \binom{m}{i}\binom{n}{k-i}$$

can be proved by a combinatorial argument: $\binom{m+n}{k}$ is the number of different choices of k elements from the set $\{1, \ldots, m+n\}$. To choose k elements, one has to choose i elements among $\{1, \ldots, m\}$ and $k-i$ elements among $\{m+1, \ldots, m+n\}$ for some i between 0 and k. The number of such choices is $\binom{m}{i}\binom{n}{k-i}$. Summing over i we obtain the desired formula.

Step 2. Let us prove

$$\binom{\alpha+n}{k} = \sum_{i=0}^{k} \binom{\alpha}{i}\binom{n}{k-i}$$

for all $\alpha \in \mathbb{R}$ and all positive integers n. For fixed n and k the left hand side of the above formula is a polynomial in α of degree k; the right hand side is also a polynomial in α of degree k. Both polynomials take equal values at $\alpha = m$ for all positive integers m. It follows that the polynomials are identical (if two polynomials of degree k coincide at $k+1$ points, then their difference is a polynomial of degree $\leq k$ with $> k$ roots, hence identically zero).

Step 3. Finally let us prove

$$\binom{\alpha+\beta}{k} = \sum_{i=0}^{k} \binom{\alpha}{i}\binom{\beta}{k-i}$$

for all $\alpha, \beta \in \mathbb{R}$. Fix $\alpha \in \mathbb{R}$ and $k \in \mathbb{Z}_{\geq 0}$. Then the left and the right hand sides are polynomials in β of degree k. By Step 2, the values of these polynomials coincide whenever β is a positive integer. Thus the polynomials are identically equal. (In particular, evaluating the left hand side and the right hand side for any values of α, k, β leads to the same results.) □

2.5 Summary of the generating function method

The combined results of this section on generating functions provide a general method to give a recursive formula for a linear recursive sequence a_n. We start by setting

$$A(x) = \sum_{k=0}^{\infty} a_k x^k.$$

Step 1: Use Theorem 2.9 to write $A(x)$ as a quotient

$$\frac{B(x)}{\overline{P}(x)},$$

where $B(x)$ is some polynomial and $P(x)$ is the characteristic polynomial of the recursive sequence. One can find $B(x)$ by multiplying:

$$B(x) = A(x)\overline{P}(x).$$

Step 2: Use partial fraction decomposition (Theorem 2.10) to write the quotient

$$\frac{B(x)}{\overline{P}(x)}$$

as a simple sum of summands of the type

$$\frac{c}{1-\lambda x}.$$

Depending on the case of Theorem 2.10 we use, we may get powers of the denominator.

Step 3: Use the generalised binomial theorem (Theorem 2.12) to write each such summand as

$$\frac{c}{1-\lambda x} = c\sum_{k=0}^{\infty}\binom{n+k-1}{n-1}\lambda^k x^k$$

.

Step 4: Compare $A(x)$ to the sum of the terms as we wrote them in Step 3. We get that a_k equals the sum of coefficients for x^k for all the terms as we wrote them in Step 3. This provides a closed formula for a_k.

Exercise 2.1. Go back to the example of the Fibonacci numbers in Section 2.1 and identify the 4 steps of the generating function method.

The generating function method is robust in the sense that it can be applied also to some sequences that are not linearly recursive: it works as soon as it is possible to write the generating function for a sequence as a quotient of two polynomials.

3 Partition of integers

In this section we will count the number of decompositions of a positive integer into summands. There are different versions of this problem. Before coming to the most difficult and interesting one in Section 3.4 we look at simpler versions.

3.1 Money changing problem

Imagine a country where only 9, 17, and 31 dollar banknotes are in circulation. In how many different ways can one pay 1000 dollars without change?

The problem can be reformulated as finding the number of integer solutions of the equation

$$9k + 17l + 31m = 1000, \quad k, l, m \geq 0$$

More generally, denote by a_n the number of different ways to pay n dollars without change. What can one say about a_n?

Theorem 3.1. *The generating function of the sequence a_n has the following form:*

$$\sum_{n=0}^{\infty} a_n x^n = \frac{1}{(1-x^9)(1-x^{17})(1-x^{31})}.$$

Proof. The right hand side is equal to

$$(1+x^9+x^{18}+\cdots)(1+x^{17}+x^{34}+\cdots)(1+x^{31}+x^{62}+\cdots) = \sum_{k,l,m\geq 0} x^{9k+17l+31m}.$$

(We pick x^{9k} from the first brackets, x^{17l} from the second brackets, and x^{31m} from the third brackets). Thus the coefficient at x^n is the number of solutions of the equation

$$9k+17l+31m=n, \quad k,l,m\geq 0,$$

that is a_n. □

One can represent the quotient $\frac{1}{(1-x^9)(1-x^{17})(1-x^{31})}$ as the sum of partial fractions. For this one has to factorize $1-x^9$. Complex roots of unity will appear. It is ultimately possible (but very time-consuming) to write a closed formula for the number of ways to change n dollars. What is easier to prove is the asymptotics of the number a_n:

$$a_n \sim \frac{n^2}{9\cdot 17\cdot 31} = \frac{n^2}{4743}, \text{ that is } \lim_{n\to 0}\frac{a_n}{n^2} = \frac{1}{4743}.$$

3.2 Compositions again

Recall that a weak composition of a number n is a representation of n as a sum of non-negative integers. (One counts ordered sums: $5=2+3$ and $5=3+2$ are different compositions.) We have computed the number of weak compositions in Section 3.5 by the "stones and sticks" method. Let us do it again with generating functions.

Fix a positive integer k and denote by a_n the number of weak compositions of n from k parts.

Theorem 3.2. *The generating function of the sequence a_n has the following form:*

$$\sum_{n=0}^{\infty} a_n x^n = \frac{1}{(1-x)^k}.$$

Proof. One has

$$\frac{1}{(1-x)^k} = \underbrace{(1+x+x^2+\cdots)\cdots(1+x+x^2+\cdots)}_{k}$$

When one expands the brackets in the product on the right hand side, one picks from the first brackets a monomial x^{m_1}, from the second x^{m_2} and so on up to x^{m_k} from the last brackets. The product of these monomials is

$x^{m_1+\cdots+m_k}$. When one collects the monomials of degree n, one obtains a term $a_n x^n$, where n is the number of solutions of the equation

$$m_1 + \cdots + m_k = n,$$

where the unknowns $m_1, \ldots, m_k$ can take only non-negative integer values. The number of solutions is exactly the number of weak compositions of n from k parts. □

Now, by applying the generalized binomial theorem one obtains

$$\sum_{n=0}^{\infty} a_n x^n = \frac{1}{(1-x)^k} = (1-x)^{-k} = \sum_{n=0}^{\infty} (-1)^n \binom{-k}{n} x^n$$

Thus we have

$$a_n = (-1)^n \binom{-k}{n} = \binom{n+k-1}{n} = \binom{n+k-1}{k-1}.$$

3.3 Fibonacci once again

Write the generating function for the Fibonacci sequence in the following way:

$$\frac{x}{1-x-x^2} = \frac{x}{1-(x+x^2)} = x(1 + (x+x^2) + (x+x^2)^2 + \cdots)$$

Multiplying out the power $(x+x^2)^k$, one obtains monomials of the form $x^{m_1+\cdots+m_k}$, where each of m_i is equal 1 or 2. Summing $(x+x^2)^k$ over all k (and not forgetting the factor x on the right hand side of the above equation) one obtains

$$\frac{x}{1-x-x^2} = \sum_{n=1}^{\infty} a_{n-1} x^n,$$

where a_n is the number of ways to represent n as a sum of ones and twos. The number of summands is not prescribed, and representations that differ in the order of summands are counted separately.

The above interpretation of Fibonacci numbers is equivalent to the following one: the n-th Fibonacci number is the number of domino tilings of the $2 \times (n-1)$ rectangle. See Figure 1.

3.4 Partitions and their generating function

Definition 3.3. *A* partition *of a positive integer n is its representation as a sum of positive integers. Representations that differ only in the order of summands are considered the same.*

The number of partitions of n is denoted by p_n.

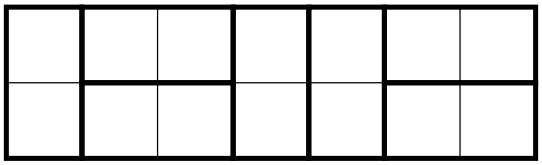

Figure 1: This tiling corresponds to the representation $7 = 1+2+1+1+2$.

In order to distinguish between different partitions, it is convenient to write the summands in the non-increasing order. This means, one can also define partitions as compositions with non-increasing summands.

Example 3.4. Below is the list of all partitions of the number 5.

$$\begin{aligned} 5 &= 5 \\ &= 4+1 \\ &= 3+2 \\ &= 3+1+1 \\ &= 2+2+1 \\ &= 2+1+1+1 \\ &= 1+1+1+1+1 \end{aligned}$$

Thus we have $p_5 = 7$.

Here are the first few terms of the sequence p_n, starting with p_0 which we by definition set to be equal to 1:

$$1, 1, 2, 3, 5, 7, 11, 15, 22, \ldots.$$

Unlike for the money changing problem and for Fibonacci numbers, there is no closed formula for the number of partitions. But there are a lot of beautiful theorems about partitions, and we will only get a glimpse of the theory. Often we will be using the generating function method.

Theorem 3.5. *The generating function of the sequence p_n is*

$$\sum_{n=0}^{\infty} p_n x^n = \frac{1}{(1-x)(1-x^2)(1-x^3)\cdots}.$$

Proof. This is similar to the money changing problem, but with banknotes of any denomination available.

The right hand side is equal to

$$(1+x+x^2+\cdots)(1+x^2+x^4+\cdots)(1+x^3+x^6+\cdots)\cdots = \sum x^{m_1+2m_2+3m_3+\cdots+km_k},$$

where the sum is taken over all k and all collections of non-negative integers $m_1, \ldots, m_k$. When we collect the like terms, the coefficient at x^n will be equal to the number of representations of n in the form $m_1+2m_2+\cdots+km_k$. This corresponds to a unique partition, namely to

$$n = \underbrace{k + \cdots + k}_{m_k} + \cdots + \underbrace{1 + \cdots + 1}_{m_1}.$$

Thus the product on the right hand side is equal to the generating function of the number of partitions. □

3.5 Infinite products of power series

In the above proof we met an infinite product of power series. This product is again a power series because in order to compute the coefficient at x^n only finitely many factors from the infinite product are needed (the first n factors in the above case). The definitions below formalize this.

Definition 3.6. *One says that a sequence c_n* stabilizes *to c if $c_n = c$ for all sufficiently big n:*

$$\exists N \text{ such that } c_n = c \forall n > N.$$

Definition 3.7. *Let $B_1(x), B_2(x), \ldots$ be a sequence of formal power series with*

$$B_k(x) = \sum_{n=0}^{\infty} b_{k,n}(x) x^n$$

One says that the sequence $B_k(x)$ of power series converges *to the power series $B(x) = \sum_{n=0}^{\infty} b_n x^n$:*

$$\lim_{k\to\infty} B_k(x) = B(x)$$

if the sequence of coefficients at x^n in $B_k(x)$ stabilizes to the coefficient at x^n in $B(x)$:

$$\forall n \exists K_n \text{ such that } b_{k,n} = b_n \forall k > K_n.$$

Definition 3.8.

$$\prod_{i=1}^{\infty} A_i(x) = \lim_{k\to\infty} \prod_{i=1}^{k} A_i(x)$$

Lemma 3.9. *Let $A_i(x) = 1 + a_{i,1}x + a_{i,2}x^2 + \cdots$. The infinite product $\prod_{i=1}^{\infty} A_i(x)$ is well-defined if and only if $\lim_{k\to\infty} \deg(A_k(x) - 1) = \infty$. Here the* degree *of a formal power series is the index of the first non-zero coefficient:*

$$\deg(a_m x^m + a_{m+1}x^{m+1} + \cdots) = m \text{ if } a_m \neq 0.$$

3.6 Algebraic and bijective proofs

Here is the first amazing fact about partitions.

Theorem 3.10. *The number of partitions of n into distinct parts is equal to the number of partitions of n into odd parts.*

Algebraic proof. The generating function for partitions into distinct parts is

$$(1+x)(1+x^2)(1+x^3)\cdots$$

The generating function for partitions into odd parts is

$$(1+x+x^2+\cdots)(1+x^3+x^6+\cdots)(1+x^5+x^{10}+\cdots)\cdots = \frac{1}{1-x}\frac{1}{1-x^3}\frac{1}{1-x^5}\cdots$$

Let us show that the first formal power series is equal to the second one.

$$(1+x)(1+x^2)(1+x^3)\cdots = \frac{1-x^2}{1-x}\frac{1-x^4}{1-x^2}\frac{1-x^6}{1-x^3}\cdots = \frac{1}{1-x}\frac{1}{1-x^3}\frac{1}{1-x^5}\cdots \tag{5}$$

This equation is a bit more subtle than it appears. We have

$$(1+x)\cdots(1+x^{2k}) = \frac{1-x^2}{1-x}\cdots\frac{1-x^{4k}}{1-x^{2k}} = \frac{(1-x^{2k+2})\cdots(1-x^{4k})}{(1-x)\cdots(1-x^{2k-1})}$$

The right hand side has the same coefficient at x^i for $i \le 2k$ as the infinite product on the right hand side of (5). And the left hand side has the same coefficient at x^i for $i \le 2k$ as the infinite product on the left hand side of (5). □

Bijective proof. Take a partition of n into odd parts:

$$n = 1\cdot m_1 + 3\cdot m_3 + 5\cdot m_5 + \cdots$$

It can be transformed into a partition into distinct parts as follows. Write m_{2k+1} in the binary system:

$$m_{2k+1} = 2^{d_1} + \cdots + 2^{d_s}, \quad d_i \neq d_j.$$

Then replace $(2k+1)\cdot m_{2k+1}$ by

$$(2k+1)(2^{d_1}+\cdots+2^{d_s}) = 2^{d_1}(2k+1) + \cdots + 2^{d_s}(2k+1).$$

Being done for all k, this gives a new partition of n. The parts of the new partition are different. Indeed, any two parts that come from the same k are distinct: $2^{d_i}(2k+1) \neq 2^{d_j}(2k+1)$ because $d_i \neq d_j$. Any two parts that come from different k are also distinct: $2^{d_i(k)}(2k+1) \neq 2^{d_j(l)}(2l+1)$ because they have different greatest odd divisors $2k+1 \neq 2l+1$.

In the opposite direction, we transform every partition of n into distinct parts as follows:

$$\begin{aligned} n = k_1 + \cdots + k_t, \quad & k_i \neq k_j \\ & = o_1 2^{d_1} + \cdots + o_t 2^{d_t}, \quad o_i \text{ odd} \\ & = 1 \cdot m_1 + 3 \cdot m_3 + \cdots, \quad m_{2k+1} = \sum_{o_i = 2k+1} 2^{d_i} \end{aligned}$$

It can easily be shown that this transformation is inverse to the first one. Thus we have a bijection between partitions into odd and partitions into distinct parts. □

Example 3.11. Turning a partition with odd parts into a partition with distinct parts:

$$42 = 7{+}7{+}7{+}\underbrace{3 + \cdots + 3}_{7} = 7(2{+}1){+}3(4{+}2{+}1) = 14{+}7{+}12{+}6{+}3 = 14{+}12{+}7{+}6{+}3$$

Turning a partition with distinct parts into a partition with odd parts:

$$42 = 20{+}12{+}10 = 4{\cdot}5{+}4{\cdot}3{+}2{\cdot}5 = (4{+}2)5{+}4{\cdot}3 = \underbrace{5 + \cdots + 5}_{6}{+}3{+}3{+}3$$

3.7 Recursive formulas for the number of partitions

Denote by $p(n, \leq k)$ the number of partitions of n into at most k parts. Clearly, one has $p_n = p(n, \leq n)$.

Theorem 3.12. *One has*

$$p(n, \leq k) = p(n, \leq k-1) + p(n-k, \leq k). \tag{6}$$

Proof. Partitions of n into $\leq k$ parts can be split into two classes: partitions into $\leq k-1$ parts and partitions into exactly k parts. By definition, the first class contains $p(n, \leq k-1)$ partitions. Take a partition from the second class and subtract 1 from each of its parts. Since it had exactly k parts, the sum of the parts becomes $n-k$, and their number becomes $\leq k$ (it will be strictly less than k if the smallest part was of size 1, and exactly k if the smallest part was larger than 1). This establishes a bijection between the second class and partitions of $n-k$ into $\leq k$ parts and proves the theorem. □

Using relation (6), one can compute the entries of the table $p(n, \leq k)$ recursively. First, one fills the row $k = 1$ and the column $n = 0$ with ones. Then, one can fill the table row after row or column after column. One should observe that for $k > n$ one has $p(n, \leq k) = p(n, \leq n)$. If one goes column after column, then in order to fill the column for $n = i$ one marks the diagonal $n+k = i$ and computes the (n, k)-entry as the sum of the entry immediately above it and the entry on the intersection of the current row and the marked diagonal.

k^n	0	1	2	3	4	5	6
1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	4
3	1	1	2	3	4	5	7
4	1	1	2	3	5	6	9
5	1	1	2	3	5	7	10
6	1	1	2	3	5	7	11

There is a recurrence which allows a much faster computation of the number of partitions.

Theorem 3.13 (MacMahon's recurrence)**.** *The number of partitions satisfies the following recurrence relation:*

$$p(n) = \sum_{k=1}^{\infty} (-1)^{k-1} \left(p\left(n - \frac{3k^2 - k}{2}\right) + p\left(n - \frac{3k^2 + k}{2}\right)\right)$$

Here the notation $p(n)$ is used instead of p_n.

Computing the first few terms of the sequences $\frac{3k^2 \pm k}{2}$, one rewrites MacMahon's recurrence as

$$p_n = p_{n-1} + p_{n-2} - p_{n-5} - p_{n-7} + p_{n-12} + p_{n-15} - \cdots$$

The sum on the right hand side looks infinite but is in fact finite: $p(n-i)$ becomes zero as soon as i exceeds n. Thus the sum contains about $2\sqrt{\frac{2}{3}n}$ summands and allows a very fast computation of p_n. MacMahon, more than a hundred years ago, has computed $p(n)$ for n up to 200. In particular, he found that

$$p(200) = 3972999029388.$$

We do not give a proof MacMahon's recurrence.

3.8 More about partitions

- An infinite (but fast convergent) series that computes p_n was found by Ramanujan and Hardy and later improved by Rademacher. A consequences of the latter is the asymptotics for the number of partitions:

$$p_n \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}}.$$

- Ramanujan observed and later proved that

$$p_{5n+4} \text{ is divisible by } 5,$$
$$p_{7n+5} \text{ is divisible by } 7,$$
$$p_{11n+6} \text{ is divisible by } 11.$$

- Erdös and Lehner proved that a "random" partition of n has $\frac{2\pi}{\sqrt{6}}\sqrt{n}\log n$ summands.

Both Ramanujan and Erdös were extraordinary figures. For the biography of Ramanujan see, for example, `http://www-history.mcs.st-andrews.ac.uk/Biographies/Ramanujan.html`.

Further reading about partitions: [2].

4 Catalan numbers

4.1 Rooted binary trees

Recall that a *rooted tree* is a tree with a marked vertex, the root. We have used rooted trees (with labels at the vertices) as parse trees of propositional formulas and as proof structures in the sequent calculus. Edges or a rooted tree have a natural orientation such that the path from the root to every vertex goes in the direction of edges. The *out-degree* of a vertex in a rooted tree is the number of outward-directed edges incident to this vertex. Similarly, the *in-degree* is the number of inward-directed edges; the in-degree of the root is zero, and the in-degrees of all other vertices are one.

A *binary rooted tree* is a rooted tree where the out-degrees of all vertices are at most two. A *full binary rooted tree* is a rooted tree where the out-degree of each vertex is either two or zero. (Vertices with out-degree zero are the leaves of the tree.)

With the help of the handshake lemma and the relation $|V| = |E| + 1$ one can show that a full binary rooted tree with $n + 1$ leaves has $2n + 1$ vertices and $2n$ edges.

Definition 4.1. *The number of different full binary rooted trees with $n + 1$ leaves is called the n-th Catalan number and is denoted by c_n.*

We count trees not up to isomorphism, but take into account also the way they are drawn in the plane. For example, Figure 2 shows all binary trees with 4 leaves. (When there is no risk of confusion, by "binary tree" we mean "full binary rooted tree".)

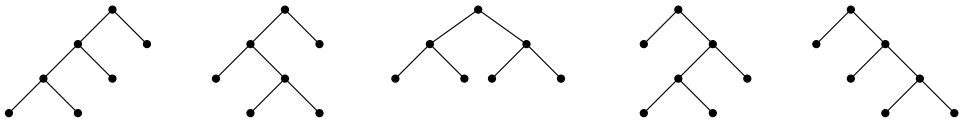

Figure 2: All 5 binary trees with 4 leaves.

One has $c_1 = 1, c_2 = 2, c_3 = 5, c_4 = 14, \ldots$.

Theorem 4.2. *The sequence of Catalan numbers satisfies the recurrence*

$$c_{n+1} = \sum_{k=0}^{n} c_k c_{n-k} = c_0 c_n + c_1 c_{n-1} + \cdots + c_n c_0, \tag{7}$$

where one puts $c_0 = 1$.

Proof. Take a binary tree with $n + 2$ leaves and remove its root. This splits the tree into two parts: the left subtree and the right subtree, see Figure 3.

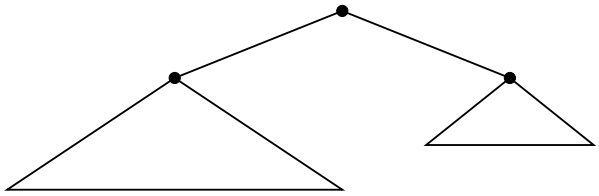

Figure 3: Proof of the recurrence for Catalan numbers.

Each of them is either a full binary tree or a single vertex. If the left subtree contains $k+1$ leaves, then the right subtree contains $n-k+1$ leaves (if the subtree has only one vertex, then the number of leaves is one). For every k, there are c_k possible left subtrees and c_{n-k} possible right subtrees. This leads to the formula. □

4.2 Generating function and the formula for c_n

Theorem 4.3. *The n-th Catalan number is equal to*

$$c_n = \frac{1}{n+1}\binom{2n}{n}. \tag{8}$$

Lemma 4.4. *For the generating function* $C(x) = c_0 + c_1 x + c_2 x^2 + \cdots$ *of the Catalan sequence the following identity holds:*

$$(C(x))^2 = \frac{C(x)-1}{x}.$$

Observe that on the right hand side we divide a formal power series by x. In Section 2.2 we have shown that division by formal power series with non-zero constant term is possible. In general, it is not allowed to divide by x: for example, $\frac{1+x}{x}$ does not correspond to any power series (there is no series $A(x)$ such that $1+x = xA(x)$). But in the above case one has $c_0 = 1$, therefore $C(x) - 1 = c_1 x + c_2 x^2 + \cdots$, and we put *by definition*

$$\frac{C(x)-1}{x} = c_1 + c_2 x + c_3 x^2 + \cdots .$$

Proof. One has

$$\begin{aligned}(C(x))^2 &= (c_0 + c_1 x + c_2 x^2 + \cdots)(c_0 + c_1 x + c_2 x^2 + \cdots) \\ &= c_0^2 + (c_0 c_1 + c_1 c_0)x + (c_0 c_2 + c_1^2 + c_2 c_0)x^2 + \cdots \\ &= 1 + c_2 x + c_3 x^2 + \cdots = \frac{C(x)-1}{x}\end{aligned}$$

□

Proof of Theorem 4.3. Lemma 4.4 implies that $C(x)$ satisfies a quadratic equation

$$x(C(x))^2 - C(x) + 1 = 0.$$

It follows that

$$C(x) = \frac{1 - \sqrt{1-4x}}{2x}.$$

(We choose the minus sign before the square root because otherwise the numerator is not divisible by x. One can check that this is a solution of the quadratic equation by direct substitution.) By the generalized binomial formula one has

$$\sqrt{1-4x} = \sum_{k=0}^{\infty} \binom{\frac{1}{2}}{k} (-4x)^k,$$

where

$$\begin{aligned}\binom{\frac{1}{2}}{k} &= \frac{\frac{1}{2} \cdot -\frac{1}{2} \cdot \cdots \cdot \left(\frac{1}{2} - k + 1\right)}{k!} = (-1)^{k-1} \frac{(2k-3)!!}{2^k k!} \\ &= (-1)^{k-1} \frac{(2k-2)!}{2^{2k-1} k! (k-1)!} = (-1)^{k-1} \frac{1}{2^{2k-1} k} \frac{(2k-2)!}{(k-1)!(k-1)!} \\ &= (-1)^{k-1} \frac{1}{2^{2k-1} k} \binom{2k-2}{k-1}\end{aligned}$$

for all $k \geq 1$, while $\binom{\frac{1}{2}}{0} = 1$. By substituting the expression for $\binom{\frac{1}{2}}{k}$ into the binomial formula one gets

$$\sqrt{1-4x} = 1 - \sum_{k=1}^{\infty} \frac{1}{2^{2k-1} k} \binom{2k-2}{k-1} 4^k x^k = 1 - \sum_{k=1}^{\infty} \frac{2}{k} \binom{2k-2}{k-1} x^k.$$

It follows that

$$C(x) = \frac{1}{2x} \sum_{k=1}^{\infty} \frac{2}{k} \binom{2k-2}{k-1} x^k = \sum_{k=1}^{\infty} \frac{1}{k} \binom{2k-2}{k-1} x^{k-1} = \sum_{k=0}^{\infty} \frac{1}{k+1} \binom{2k}{k} x^k,$$

which proves the theorem. □

4.3 Bracket-variable expressions

Assume that we have to multiply three variables x, y, and z. Here by "multiplication" we mean any binary operation. If this operation is commutative and associative, then neither the order of variables nor the order of operations is important:

$$(xy)z = x(yz) = x(zy).$$

If the operation is associative but not commutative (as multiplication of matrices for example), then the order of variables matters, but the order of operations does not:

$$(xy)z = x(yz) \neq x(zy).$$

Finally, if the operation is neither commutative nor associative (as, for example, $xy := x^y$), then we must take care both of the order of variables and the order of operations:

$$(xy)z \neq x(yz).$$

In how many ways can one multiply a given sequence of variables without changing their order? For two variables there is only one way, for three variables two: $(xy)z$ and $x(yz)$, below are all expressions with four variables:

$$((x_1x_2)x_3)x_4, \quad (x_1(x_2x_3))x_4, \quad (x_1x_2)(x_3x_4), \quad x_1((x_2x_3)x_4), \quad x_1(x_2(x_3x_4)).$$

Theorem 4.5. *The number of different multiplication orders in a sequence of $n+1$ variables is equal to the n-th Catalan number.*

Proof. We establish a bijection between binary trees with $n+1$ leaves and bracket-variable expressions with $n+1$ variables.

Any tree with $n+1$ leaves is the parse tree of some bracket-variable expression. Put the variables $x_1, \ldots, x_{n+1}$ at the leaves of the tree, in the order from the left to the right. Then mark the non-leaf vertices of the tree in the following way: if the children of a vertex are marked with A and B, then mark the vertex with (AB). The expression which appears at the root is the bracket-variable expression parsed by the tree. (It has extra brackets around it, which can be removed.)

Thus one has a map from the set of binary trees to the set of bracket-variable expressions. In order to show that this map is a bijection, one has to show that from *every* bracket-variable expression one can reconstruct *uniquely* the tree which produces this expression by the above procedure.

This reconstruction (the inverse map from expressions to trees) is described as follows. Consider the last multiplication to be performed and split the expression at this place. Draw a root with two children and write the left part of the expression at the left child, and the right part at the right child. Split in the same way the expressions at the child vertices and continue until all leaves will be marked with variables. □

4.4 Triangulations of polygons

A *triangulation* of a polygon is a subdivision of the polygon into triangles by diagonals.

Theorem 4.6. *The number of different triangulations of a convex $(n+2)$-gon is equal to the n-th Catalan number.*

In Figure 4 all triangulations of a regular pentagon are presented. As one can see, triangulations which differ by rotation or reflection of the polygon are counted separately.

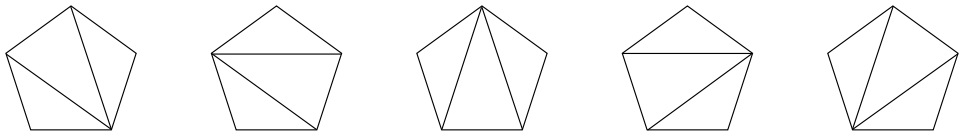

Figure 4: All 5 triangulations of the pentagon.

Proof. We establish a bijection between triangulations and binary trees.

Place a dot inside every triangle of the triangulation and place a dot near every edge of the polygon just outside of the polygon. Then draw a segment between every pair of vertices separated by an edge of the triangulation or by an edge of the polygon. The result is a tree; every vertex inside of a triangle has degree 3, and the vertices outside of the polygon are leaves. Remove the dot at the base edge of the polygon and the edge incident to it. The result is a rooted binary tree. See Figure 5 for an example.

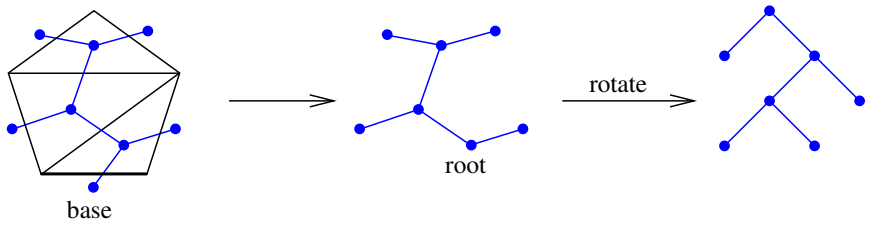

Figure 5: From a triangulation to a binary tree.

The base edge is chosen in advance. For example, if the polygon "stands" on a line, one can declare the lowest edge the base edge. Distinguishing the base edge reflects the fact that the polygon is not allowed to rotate.

From every binary tree one can reconstruct the corresponding triangulation in a unique way. First, add an edge to the root so that all non-leaf vertices have degree 3. Then draw a triangle around each vertex so that each of the sides of the triangle intersects one edge of the tree. Finally, for every edge of the tree glue the triangles surrounding the incident vertices along their sides intersecting this edge. □

4.5 Dyck paths

Recall that a monotone lattice path is a path on the coordinate grid moving only upwards and to the right. As we know, there are $\binom{2n}{n}$ monotone paths

from $(0,0)$ to (n,n).

Definition 4.7. *A* Dyck path *is a monotone lattice path from* $(0,0)$ *to* (n,n) *that stays above the diagonal. The path is allowed to touch the diagonal, but not to cross it.*

It is convenient to look at a Dyck path on a rotated grid as shown in Figure 6.

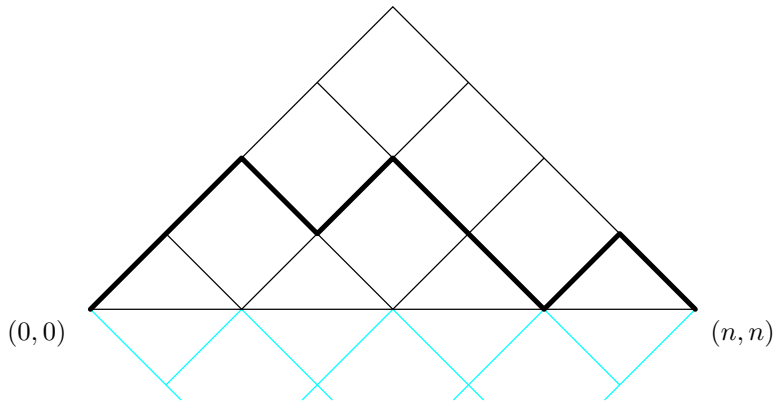

Figure 6: A Dyck path.

Theorem 4.8. *The number of Dyck paths from* $(0,0)$ *to* (n,n) *is equal to the* n*-th Catalan number.*

Proof. Let us show that the sequence d_n of numbers of Dyck paths satisfies the same recurrence relation that the sequence of Catalan numbers. Take a path from $(0,0)$ to $(n+1,n+1)$ and let $(k+1,k+1)$ be the first point after $(0,0)$ where it touches the diagonal. The number k can take any value between 0 and n. The point $(k+1,k+1)$ separates the path into two parts. The first part never touches the diagonal except at the endpoints. If we remove from it the initial and the terminal segments, then we get a Dyck path from $(1,0)$ to $(k+1,k)$. It can be identified by translation with a Dyck path from $(0,0)$ to (k,k). The part of the path after the point $(k+1,k+1)$ can be identified with a Dyck path from $(0,0)$ to $(n-k,n-k)$.

Conversely, from any k-Dyck path and any $(n-k)$-Dyck path one can build a $(n+1)$-Dyck path by "lifting up" the k-path and concatenating it with the $(n-k)$-path. Thus the number of $(n+1)$-Dyck paths whose first contact with the diagonal is at $(k+1,k+1)$ is $d_k d_{n-k}$, and the total number of $(n+1)$-Dyck paths is

$$d_{n+1} = \sum_{k=0}^{n} d_k d_{n-k}.$$

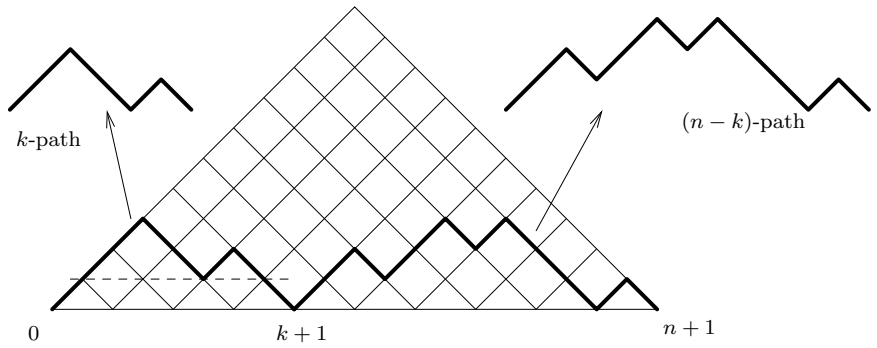

Figure 7: Proving the recursive relation for the number of Dyck paths.

Thus we have for the sequence d_n the same recursive relation, and also the same starting value $d_0 = 1$. It follows that $d_n = c_n$ for all n. □

Corollary 4.9. *The number of balanced bracket sequences of n opening and n closing brackets is c_n.*

Balanced bracket sequences are characterized by the property that, when reading it from the left to the right, the number of closing brackets never exceeds the number of opening brackets. This leads to a natural bijection with Dyck paths: interpret an opening bracket as a step upwards, and a closing bracket as a step to the right.

4.6 A combinatorial proof of the formula for c_n

The proof of the formula for c_n in Section 4.2 was algebraic, building upon the recursive formula (7). Since Catalan numbers have so many combinatorial interpretation, it would be good to have a combinatorial proof of the same formula.

Instead of counting Dyck paths let us count paths that enter the triangle below the diagonal. They may stay all the time below the diagonal or cross the diagonal one or several times. We call them non-Dyck paths.

Lemma 4.10. *Among the monotone paths from $(0,0)$ to (n,n) there are exactly $\binom{2n}{n-1}$ non-Dyck paths.*

If we prove this lemma, then the formula for c_n follows immediately:

$$c_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n+1)!(n-1)!}$$
$$= \frac{(2n)!}{(n+1)!n!}((n+1)-n) = \frac{1}{n+1}\binom{2n}{n}.$$

Proof of Lemma 4.10. Let $(k, k-1)$ be the first point where a non-Dyck path enters the triangle below the diagonal. The number k can take any value between 1 (which means that the first step goes below the diagonal) and n. Reflect the part of the path from $(k, k-1)$ to (n, n) as shown in Figure 8.

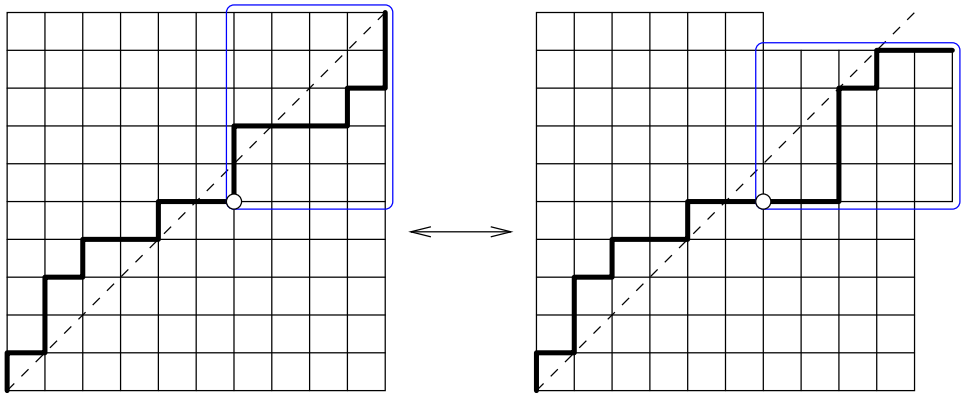

Figure 8: Counting non-Dyck paths.

This transforms every non-Dyck path to a path from $(0, 0)$ to $(n+1, n-1)$. For every path from $(0, 0)$ to $(n+1, n-1)$ there is a unique non-Dyck path that produces it. To reconstruct this non-Dyck path, apply the same operation: take the first point of the form $(k, k-1)$ on the path to $(n+1, n-1)$ and reflect the part of the path after this point. □

Chapter VI

Automata theory

The main source for this chapter is [7].

1 Finite automata

1.1 Alphabets, words, and languages

An *alphabet* is any finite set of symbols. Examples:

- the binary alphabet $\{0, 1\}$;
- the alphabet of a single symbol $\{0\}$;
- the alphabet $\{p_1, \dots, p_n\} \cup \{\neg, \wedge, \vee, \to, (,)\}$ of the propositional logic.

A *string* or a *word* is a finite sequence of symbols from a given alphabet. The set of words of length n in the alphabet Σ is denoted by Σ^n:

$$\Sigma^n = \{x_1 \dots x_n \mid x_i \in \Sigma \ \forall i\}.$$

This is the same as the Cartesian power Σ^n, with only a notational difference: $x_1 \dots x_n$ instead of $(x_1, \dots, x_n)$.

The concatenation of two words defines a map $\Sigma^m \times \Sigma^n \to \Sigma^{m+n}$. Clearly, $uv \neq vu$ in general. Denote by $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$ the set of all words in the alphabet Σ.

There is a unique element in Σ^0: the word of zero length; it is denoted by ε. One has

$$\varepsilon w = w = w\varepsilon \text{ for all } w \in \Sigma^*.$$

A *language* is a subset of Σ^*. Here are some examples of languages.

- The set of all sequences of zeros of prime length:

$$\{0^p \mid p \text{ is a prime number}\}.$$

- The set of all binary palindromes (binary sequences that read the same forward and backward):

$$\{\varepsilon, 0, 1, 00, 11, 000, 010, \ldots\}.$$

- In the alphabet of propositional logic, the set of all propositional formulas.
- In the same alphabet, the set of all propositional formulas which are tautologies.

1.2 Deterministic finite automata

A *finite automaton* is a machine with finitely many states that changes its states according to the input.

Definition 1.1. *A* deterministic finite automaton (DFA) *is a quintuple* $(Q, \Sigma, \delta, q_0, F)$*, where*

- Q *is the set of states;*
- Σ *is the input alphabet;*
- δ *is the transition function, that is, a map* $\delta\colon Q \times \Sigma \to Q$*;*
- q_0 *is the initial state;*
- $F \subset Q$ *is the set of final states.*

The sets Q *and* Σ *are assumed to be finite, and* F *non-empty.*

It is convenient to represent a DFA in the form of a *transition diagram*. A transition diagram is a graph whose vertex set is the set of states, and edges are directed and labeled by the alphabet symbols. The edges describe the transition function: if $\delta(q_i, a) = q_j$, then we draw a directed edge from q_i to q_j and label it with a. The initial state is indicated by an incoming arrow starting at nowhere. The final states are indicated by double circles. Figure 1 shows an example of a DFA.

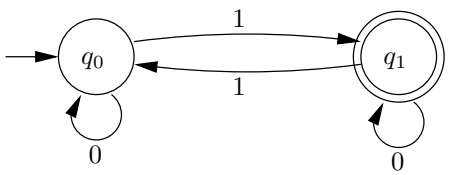

Figure 1: A deterministic finite automaton.

At the beginning the automaton is in the initial state. When it receives an input word $w \in \Sigma^*$, it reads it from left to right and changes its state after each letter according to the transition function. If after reading the input the automaton is in one of the final states, then one says that the word w is *accepted* by the automaton. (Because of this, the final states are sometimes called *accepting states*.)

One describes it formally by extending the transition function δ to a function $\widehat{\delta}\colon Q \times \Sigma^* \to Q$. The value of $\widehat{\delta}(q, w)$ is the state in which the automaton ends if it starts at q and reads the word w. The definition is recursive:

1. $\widehat{\delta}(q, \varepsilon) = q$ for every state q;
2. $\widehat{\delta}(q, wa) = \delta(\widehat{\delta}(q, w), a)$ for every state q, every word w, and every letter a.

Definition 1.2. *Let M be a DFA. A word $w \in \Sigma^*$ is called* accepted *by M if $\widehat{\delta}(q_0, w) \in F$. The* language $L(M)$ accepted by M *is the set of all words accepted by M:*

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \in F\}.$$

Example 1.3.

a) The language accepted by the automaton on Figure 1 consists of all strings with an odd number of ones.

b) One has $\varepsilon \in L(M)$ if and only if the initial state q_0 of M belongs to the set of final states of M.

Definition 1.4. *A language is called* regular *if it is accepted by some DFA.*

The main question which will be studied is:

WHAT LANGUAGES ARE REGULAR?

In other words, what decision problems can be solved by DFAs? For example, is there a DFA with the alphabet of propositional logic, which can tell if a given sequence of symbols is a tautological propositional formula?

Remark 1.5. Alternatively to a transition diagram, a DFA can be represented by a table. For the DFA from Figure 1 this is

	0	1
q_0	q_0	q_1
q_1	q_1	q_0

The table contains the information about the transition function, the set of states, and the input alphabet. In addition one should specify the set of final states, $F = \{q_1\}$ in our case.

1.3 Nondeterministic finite automata

In a *nondeterministic finite automaton* (NFA) one allows the transition from a given state on a given input to be not uniquely defined or undefined. In other words, while on the transition diagram of a DFA for every state q and every input symbol a there is a unique outgoing arrow from q labeled with a, on the transition diagram of an NFA there might be several arrows like that or none at all. Figure 2 shows an example of an NFA.

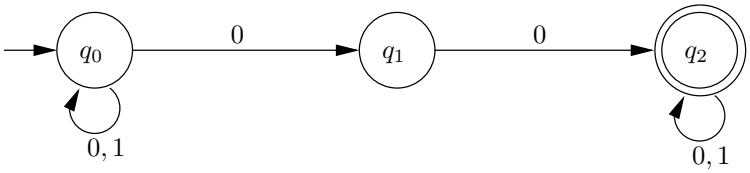

Figure 2: A nondeterministic finite automaton.

A formal definition is as follows.

Definition 1.6. *A* nondeterministic finite automaton (NFA) *is a quintuple* $(Q, \Sigma, \delta, q_0, F)$*, where as before* Q *is a set of states,* Σ *is a finite alphabet,* $q_0 \in Q$ *is the initial state,* $F \subset Q$ *is the set of final states. However, the transition function*

$$\delta\colon Q \times \Sigma \to 2^Q$$

associates to a pair (q, a) *not a state, but a set of states* $P \subset Q$*, which is allowed to be empty.*

(Recall that 2^Q denotes the set of all subsets of Q.)

As next, we need to describe how an NFA works, that is what words does it accept. Informally speaking, an NFA can be in several states at the same time, and when an input symbol is read, each of these states generates a new set of states. The number of current states does not necessarily increase with each step, because for some input there may be no transition defined from some states: some branches die off. In a more dramatic way this can be imagined as creation of parallel universes when the transition is not uniquely defined and an apokalypsis in a given universe if the transition for a given input symbol is undefined.

Formally, we define an extended transition function recursively as

1. $\widehat{\delta}(q, \varepsilon) = \{q\}$;
2. $\widehat{\delta}(q, wa) = \bigcup_{p \in \widehat{\delta}(q,w)} \delta(p, a)$.

Now, a word w is considered *accepted* by an NFA if there is a sequence of transitions corresponding to the input w that ends up in a final state.

(That is, a word is accepted if it is accepted in at least one of the parallel universes.)

Formally, we define the language accepted by a non-deterministic automaton M as

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \cap F \neq \varnothing\}. \tag{1}$$

Example 1.7.

a) The automaton in Figure 2 accepts all words that contain two consecutive zeros. Indeed, the state q_2 can be reached only if the input contains two consecutive zeros, and once q_2 is reached, one stays there forever.

b) The automaton in Figure 3 accepts all words that do not contain two consecutive zeros. Indeed, it "breaks down" only if the input contains two consecutive zeros, and if it does not break down, then it accepts the input.

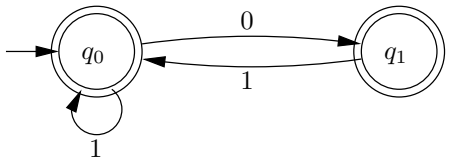

Figure 3: A nondeterministic finite automaton with $\delta(q_1, 0) = \varnothing$.

Definition 1.8. *Two finite automata M and M' are called* equivalent *if they accept the same languages: $L(M) = L(M')$.*

Theorem 1.9. *For any NFA there is an equivalent DFA.*

Proof. The proof is based on the interpretation of an NFA as "being in several states at the same time". One constructs a deterministic automaton whose states correspond to sets of states of NFA.

Take any NFA $M = (Q, \Sigma, \delta, q_0, F)$. Define a DFA $M' = (Q', \Sigma, \delta', q_0', F')$ as follows.

- $Q' = 2^Q$: the states of M' are all subsets of the set of states of M.
- $q_0' = \{q_0\}$, a one-element set.
- $F' = \{P \subset Q \mid P \cap F \neq \varnothing\}$, all subsets of Q that contain at least one final state of M.
- For any $P \subset Q$ put $\delta'(P, a) = \bigcup\limits_{p \in P} \delta(p, a)$.

We claim that

$$\widehat{\delta'}(\{q\}, u) = \widehat{\delta}(q, u)$$

for all $q \in Q$ and all $u \in \Sigma^*$ and prove it by induction on the length of u. If $u = \varepsilon$, then both sides are equal to $\{q\}$. Take any word of length at least 1, and let a be its last letter. Then this word can be written as wa, and we have

$$\widehat{\delta'}(\{q\}, wa) = \delta'(\widehat{\delta'}(\{q\}, w), a) = \bigcup_{p \in \widehat{\delta'}(\{q\}, w)} \delta(p, a)$$

$$\widehat{\delta}(q, wa) = \bigcup_{p \in \widehat{\delta}(q, w)} \delta(p, a)$$

By induction hypothesis, $\widehat{\delta'}(\{q\}, w) = \widehat{\delta}(q, w)$, and the induction step is proved.

Definition of F' and definition (1) imply that $L(M) = L(M')$. □

Example 1.10. Let us construct a DFA equivalent to the NFA in Figure 2. For convenience, write first the table of our NFA.

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\varnothing$
q_2	$\{q_2\}$	$\{q_2\}$

The set $Q = \{q_0, q_1, q_2\}$ has 8 subsets, so if we follow the construction given in the theorem literally, we must write a table with 8 rows. However, not all of the 8 states will be accessible from the initial state. The inaccessible states can be removed from the automaton without affecting the language. Therefore we will introduce new rows in our table for M' only as soon as they are needed. Also, for a better distinction we will use in M' the [ ] brackets instead of the set brackets { }. The result is the following table:

δ'	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_2]$

For brevity, rename the states so that the table takes the form

δ'	0	1
q'_0	q'_1	q'_0
q'_1	q'_2	q'_0
q'_2	q'_2	q'_3
q'_3	q'_2	q'_3

The corresponding transition diagram is shown in Figure 4. The final states are q'_2 and q'_3 because they correspond to the sets which contain the final state q_2 of M'.

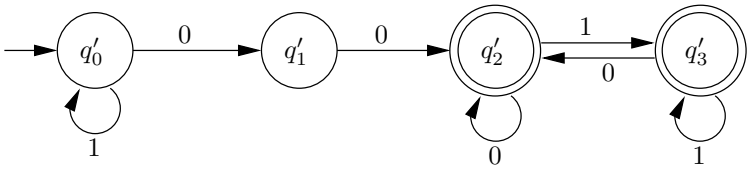

Figure 4: A DFA equivalent to the NFA in Figure 2.

Example 1.11. Figure 5 shows the DFA constructed from the NFA in Figure 3. The state q'_2 corresponds to the empty set.

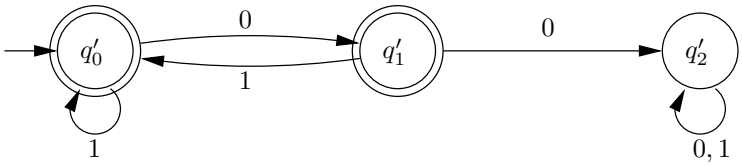

Figure 5: A DFA equivalent to the NFA in Figure 3.

1.4 Finite automata with epsilon-transitions

This is a further extension of the concept of a finite automaton. In addition to "branching" and "emergency stops" present in NFA we allow spontaneous transitions between some states. That is, the transition diagram may contain arrows marked by ε, the empty word. Spontaneous transitions may branch as well: there might be several ε-arrows starting from the same state.

Here is a formal definition.

Definition 1.12. *A* nondeterministic finite automaton with ε-transitions (ε-NFA) *is a quintuple* $(Q, \Sigma, \delta, q_0, F)$*, where* Q*,* Σ*,* $q_0 \in Q$*, and* $F \subset Q$ *are, as before, the set of states, the input alphabet, the initial state, and the set of final states, but*

$$\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q.$$

A word w is *accepted* by an ε-NFA if there is a path from the initial state to one of the final states which corresponds to the input w with any number of ε-transitions inbetween.

Example 1.13. The automaton on Figure 6 accepts positive and negative integers: strings of digits, possibly preceded by the minus sign, the only string beginning with 0 is 0.

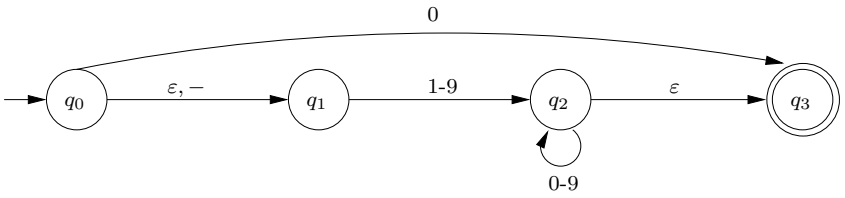

Figure 6: An ε-NFA recognizing decimally represented integers.

The transition table of this automaton is given below.

	ε	$-$	0	1-9
q_0	$\{q_1\}$	$\{q_1\}$	$\{q_3\}$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_2\}$
q_2	$\{q_3\}$	$\varnothing$	$\{q_2\}$	$\{q_2\}$
q_3	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

In order to proceed we need the following notion.

Definition 1.14. *A subset of the set of states $P \subset Q$ is called* ε-closed *if all ε-transitions from states of P lead to P: for every $q \in P$ one has $\delta(q, \varepsilon) \subset P$.*

The ε-closure *of a subset $P \subset Q$ is the minimal ε-closed subset containing P. We denote the ε-closure of P by $\overline{P}$.*

In other words, $\overline{P}$ is P together with all states that can be reached from P by sequences of ε-transitions.

Let us modify and extend the transition function so that it will tell us what states are accessible from a given state for a given input.

1. $\widehat{\delta}(q, \varepsilon) = \overline{\{q\}}$
2. $\widehat{\delta}(q, wa) = \overline{\delta(\widehat{\delta}(q, w), a)}$ for all $w \in \Sigma^*$

(Note that $\widehat{\delta}(q, w)$ is a set, so that $\delta(\widehat{\delta}(q, w), a)$ denotes the union of $\delta(p, w)$ over all $p \in \widehat{\delta}(q, w)$.)

Observe that, contrarily to the case of DFA and NFA, $\widehat{\delta}(q, a) \neq \delta(q, a)$, but rather

$$\widehat{\delta}(q, a) = \overline{\delta(\overline{\{q\}}, a)} \supset \delta(q, a).$$

It is not hard to see that $\widehat{\delta}(q, w)$ consists of all states reachable from q on the input w with arbitrarily many ε-transitions before w, in the middle of w, and after w.

In terms of the extended transition function the language accepted by an ε-NFA is defined as follows.

Definition 1.15. *The language accepted by an ε-NFA M is*

$$L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \cap F \neq \varnothing\}.$$

We will now show that ε-NFAs are not more powerful than DFA: any language accepted by an ε-NFA is also accepted by some DFA.

Theorem 1.16. *For every ε-NFA there is an equivalent DFA.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an ε-NFA. Construct a DFA $M' = (2^Q, \Sigma, \delta', q_0', F')$ by putting

$$q_0' = \overline{\{q_0\}}, \quad F' = \{P \subset Q \mid P \cap F \neq \varnothing\},$$

and defining the transition function by

$$\delta'(P, a) = \overline{\delta(P, a)}.$$

We claim that for any word $w \in \Sigma^*$ holds

$$\widehat{\delta'}(q_0', w) = \widehat{\delta}(q_0, w), \tag{2}$$

where $\widehat{\delta}$ and $\widehat{\delta'}$ are the extended transition functions of M and M'. This is proved by induction on the length of the word w. The base: $|w| = 0$, that is $w = \varepsilon$. We have

$$\widehat{\delta'}(q_0', \varepsilon) = q_0' = \overline{\{q_0\}} = \widehat{\delta}(q_0, \varepsilon).$$

The induction step: assume (2) holds for all words of length n. Any word of length $n+1$ has the form wa, where $|w| = n$ and $a \in \Sigma$. Using the induction hypothesis, we obtain

$$\widehat{\delta'}(q_0', wa) = \delta'(\widehat{\delta'}(q_0', w), a) = \delta'(\widehat{\delta}(q_0, w), a) = \overline{\delta(\widehat{\delta}(q_0, w), a)} = \widehat{\delta}(q_0, wa).$$

Now, by definition we have

$$w \in L(M) \Leftrightarrow \widehat{\delta}(q_0, w) \cap F \neq \varnothing,$$
$$w \in L(M') \Leftrightarrow \widehat{\delta'}(q_0', w) \in F' \Leftrightarrow \widehat{\delta'}(q_0', w) \cap F \neq \varnothing,$$

which implies $L(M) = L(M')$ due to (2). □

Example 1.17. Let us construct a DFA equivalent to the ε-NFA from Example 1.13.

The transition table is obtained by consulting the table from Example 1.13 and applying the rule $\delta'(P, a) = \overline{\delta(P, a)}$. As in the construction of an NFA out of a DFA, it might be not necessary to consider all subsets of Q. We start with the row corresponding to the initial state, and add a new row for every state which appeared in one of the previous rows. The construction ends when no new states appear.

The initial state in our case is $\overline{\{q_0\}} = \{q_0, q_1\}$.

	−	0	1-9
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_3\}$	$\{q_2, q_3\}$
$\{q_1\}$	$\varnothing$	$\varnothing$	$\{q_2, q_3\}$
$\{q_3\}$	$\varnothing$	$\varnothing$	$\varnothing$
$\{q_2, q_3\}$	$\varnothing$	$\{q_2, q_3\}$	$\{q_2, q_3\}$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

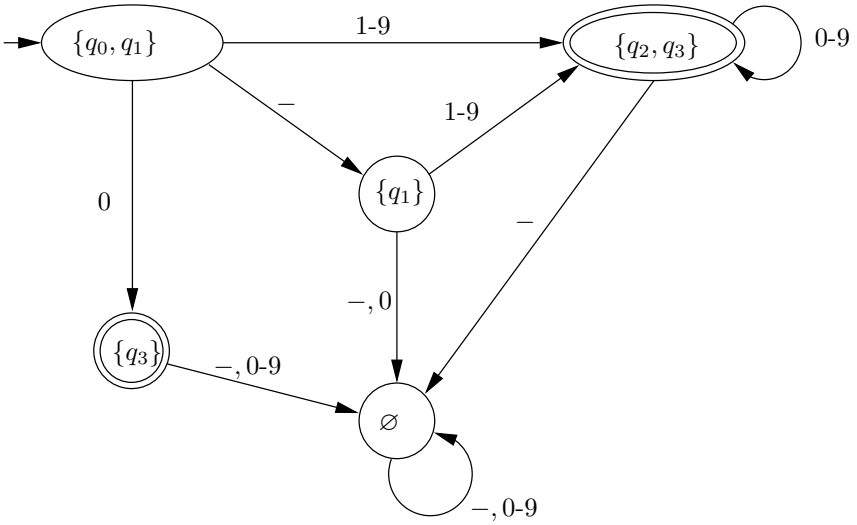

Figure 7: A DFA equivalent to the ε-NFA from Example 1.13.

The diagram of this automaton is shown in Figure 7.

2 Regular expressions

A regular expression is a formula which describes a language. We will see that languages represented by regular expressions are regular (i.e. are accepted by a finite automaton) and that every regular language can be represented by a regular expression.

2.1 Definition and examples

A regular expression is defined recursively. The basic building blocks are the following.

1. $\varnothing$ is a regular expression and denotes the language $\varnothing$.
2. ε is a regular expression and denotes the language $\{\varepsilon\}$.
3. a is a regular expression for every $a \in \Sigma$ and denotes the language $\{a\}$.

Don't confuse the empty language $\varnothing$ and the language $\{\varepsilon\}$ consisting of an empty word.

Sometimes one uses the boldface $\mathbf{a}$ to denote the language $\{a\}$. We will use the same symbol a.

From these building blocks one constructs more complex regular expressions by using the following operations. If r and s are regular expressions denoting the languages R and S respectively, then

1. $(r + s)$ is a regular expression and denotes the language $R \cup S$;

2. (rs) is a regular expression and denotes the language $RS = \{uv \mid u \in R, v \in S\}$;

3. r^* is a regular expression and denotes the language $R^* = \cup_{i=0}^{\infty} R^i$, where $R^i = \underbrace{RR\cdots R}_{i}$ (the *Kleene closure* of language R).

Example 2.1. The language of all binary words can be represented by the expression $(0 + 1)^*$.

One can omit some of the brackets in regular expressions by adopting the convention that $*$ precedes the concatenation, and the concatenation precedes the sum. For example, $((0(1^*)) + 0)$ may be written as $01^* + 0$, and we have

$$01^* + 0 = \{0, 01, 011, 0111, \ldots\}.$$

Two regular expressions are called equivalent if they describe the same language. Here are some simple equivalences:

$$(rs)t \sim rs(t), \quad (r + s)t \sim rt + st.$$

Instead of the equivalence sign we will use the equality sign to denote the equivalence of regular expressions. For example,

$$01^* + 0 = 01^*, \quad \varnothing r = \varnothing, \quad \varepsilon r = r.$$

Recall that a language is called regular if there is a finite automaton (DFA, NFA, or ε-NFA, which does not matter, as we have shown) that accepts this language. The main theorem is the following.

Theorem 2.2. *A language is regular if and only if it can be represented by a regular expression.*

This theorem will be proved in the next two sections. Now let us give some examples of regular expressions and languages corresponding to them.

Example 2.3. The language of words consisting of alternating 0's and 1's. It can be represented by a regular expression $(01)^* + (10)^* + 0(10)^* + 1(01)^*$. In this expression a case distinction is incorporated: if the word is of even length and starts with 0, then it belongs to the language $(01)^*$, etc.

The same language can be described by the expression

$$(01)^* + 1(01)^* + (01)^*0 + 1(01)^*0 = (\varepsilon + 1)(01)^*(\varepsilon + 0).$$

Example 2.4. The set of all binary words whose tenth symbol from the right is 1 can be described by a regular expression $(0+1)^*1(0+1)^9$, where $(0+1)^9$ denotes $\underbrace{(0+1)\cdots(0+1)}_{9}$.

Example 2.5. The set of all binary words that contain at least one 0 and at least one 1 can be represented by

$$(0+1)^*0(0+1)^*1(0+1)^* + (0+1)^*1(0+1)^*0(0+1)^*.$$

2.2 Equivalence of regular expressions and regular languages

We will now prove Theorem 2.2. It splits in two parts.

Lemma 2.6. *The language described by a regular expression is regular.*

Proof. We describe a construction algorithm of an ε-NFA that accepts the language described by a given regular expression r. Moreover, the resulting automaton will have a unique accepting state. The construction uses the recursive structure of the regular expression.

Let r be any regular expression. By definition, r is either basic or is obtained from one or two simpler expressions through sum, concatenation or closure.

If r is basic, then the corresponding language is accepted by one of the automata shown in Figure 8.

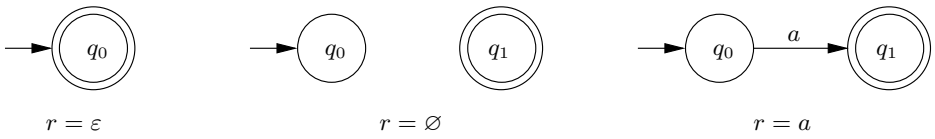

Figure 8: Automata for basic regular expressions.

If $r = r_1 + r_2$, then by assumption there are ε-NFAs M_1 and M_2, each with a unique final state, for the languages represented by r_1 and r_2. The automaton in Figure 9 accepts the language of the expression $r_1 + r_2$.

If $r = r_1 r_2$, then we combine the automata for r_1 and r_2 as shown in Figure 10.

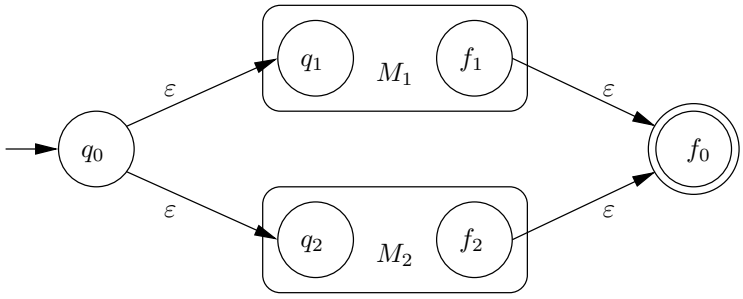

Figure 9: Automaton realizing the union of two languages.

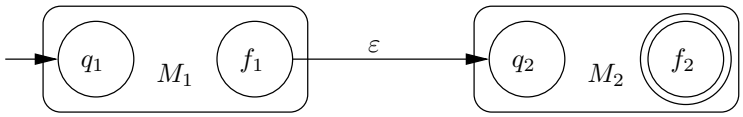

Figure 10: Automaton realizing the concatenation of two languages.

Finally, the automaton in Figure 11 accepts the language of $(r_1)^*$.

In order to show that these automata do what they are meant to do, one has to prove two things: first, each word from the language $R_1 + R_2$ (respectively, R_1R_2, or R_1^*) is accepted by the automaton; second, each word accepted by the automaton belongs to the respective language. The arguments proving this are rather straightforward, and we omit them. □

Example 2.7. Construct an automaton accepting the language $01^* + 1$.

The automaton obtained with the above argument has ten states, see Example 2.12 from the first edition of Hopcroft-Ullman. One can rather easily construct an ε-NFA with three states. The above algorithm aims not for the smallest number of states, but for the simplicity of the construction.

Now we proceed to the second part of Theorem 2.2.

Definition 2.8. *A word x is called a* prefix *of a word w if $w = xy$ for some word y. In particular, both ε and w are prefixes of w. A* proper prefix *of w is a prefix distinct from ε and w.*

Lemma 2.9. *Every regular language can be described by a regular expression.*

Proof. Let R be the language accepted by a DFA with the set of states $Q = \{q_1, \ldots, q_n\}$, where q_1 is the initial state. Our goal is to construct a regular expression r that describes R.

Denote by R_{ij}^k the set of all words w such that

- $\widehat{\delta}(q_i, w) = q_j$;

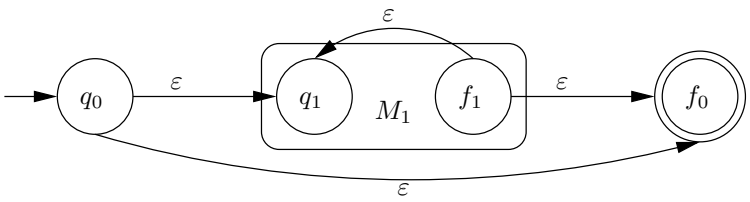

Figure 11: Automaton realizing the Kleene closure of a language.

- $\widehat{\delta}(q_i, x) \in \{q_1, \ldots, q_k\}$ for all proper prefixes x of w.

In other words, R_{ij}^k is the set of all words that lead you from q_i to q_j through the states with indices less or equal k only. Note that we allow i or j to be bigger than k: one may be in a state with number bigger than k at the beginning or at the end, but not in between. One has

$$R = \bigcup_{q_j \in F} R_{1j}^n.$$

We will prove by induction on k that each of R_{ij}^k can be represented by a regular expression.

Base: $k = 0$. By definition, R_{ij}^0 consists of direct transitions from q_i to q_j. Thus

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\}, & \text{if } i \neq j, \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\}, & \text{if } i = j. \end{cases}$$

In both cases, the set R_{ij}^0 is finite and therefore described by a regular expression of the following form:

$$r_{ij}^0 = \begin{cases} a_1 + \cdots + a_p, & \text{if } i \neq j \text{ and } R_{ij}^0 = \{a_1, \ldots, a_p\}, \\ \varnothing, & \text{if } i \neq j \text{ and } R_{ij}^0 = \varnothing, \\ a_1 + \cdots + a_p + \varepsilon, & \text{if } i = j \text{ and } R_{ij}^0 = \{a_1, \ldots, a_p\}, \\ \varepsilon, & \text{if } i = j \text{ and } R_{ij}^0 = \{\varepsilon\}. \end{cases}$$

Step. Let us prove that for $k \geq 1$ and for all i, j one has

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}.$$

Indeed, take any $w \in R_{ij}^k$ and look at the corresponding path from q_i to q_j. If this path does not pass through q_k, then $w \in R_{ij}^{k-1}$. Otherwise split the path into the following pieces:

- from q_i to q_k without passing through q_k;

- from q_k to q_k without passing through q_k (there may be several pieces like this);
- from q_k to q_j without passing through q_k.

This represents the word w as a concatenation $w = w_1 \cdots w_m$, where

$$w_1 \in R_{ik}^{k-1}, \quad w_2, \ldots, w_{m-1} \in R_{kk}^{k-1}, \quad w_m \in R_{kj}^{k-1}.$$

Thus $w \in R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$. It is also clear than any $w \in R_{ij}^{k-1}$ or $R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$ leads from q_i to q_j without passing through states with the number $> k$.

By induction assumption, for all i, j there is a regular expression r_{ij}^{k-1} which describes the language R_{ij}^{k-1}. The language R_{ij}^{k} is then described by the regular expression

$$r_{ij}^{k} = r_{ij}^{k-1} + r_{ik}^{k-1}(r_{kk}^{k-1})^* r_{kj}^{k-1}.$$

Finally, the language R is described by the regular expression

$$r = r_{1,m+1}^{n} + \cdots + r_{1,n}^{n},$$

where $F = \{q_{m+1}, \ldots, q_n\}$. □

Example 2.10. (Example 2.13. from [7].) Find a regular expression for the language accepted by the automaton on Figure 12.

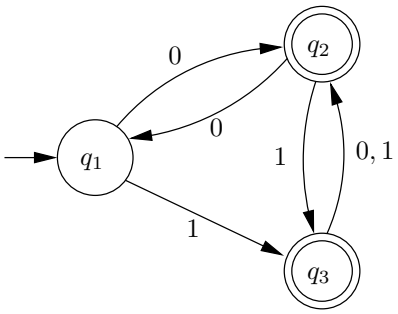

Figure 12: Automaton for Example 2.10.

One fills the table in Figure 13 column after column according to the above algorithm.

The first column is easy. For the second and the third column use the recursive formula. Sometimes a regular expression can be simplified, and this was done at several places in this table. For example,

$$r_{22}^{1} = r_{22}^{0} + r_{21}^{0}(r_{11}^{0})^* r_{12}^{0} = \varepsilon + 0(\varepsilon)^* 0 = \varepsilon + 00.$$

	$k=0$	$k=1$	$k=2$
r_{11}^k	ε	ε	$(00)^*$
r_{12}^k	0	0	$0(00)^*$
r_{13}^k	1	1	0^*1
r_{21}^k	0	0	$0(00)^*$
r_{22}^k	ε	$\varepsilon+00$	$(00)^*$
r_{23}^k	1	$1+01$	0^*1
r_{31}^k	$\varnothing$	$\varnothing$	$(0+1)(00)^*0$
r_{32}^k	$0+1$	$0+1$	$(0+1)(00)^*$
r_{33}^k	ε	ε	$\varepsilon+(0+1)0^*1$

Figure 13: Finding a regular expression for Example 2.10.

More interesting things happen to r_{13}^2, which by the direct application of the recursive formula is equal to

$$r_{13}^2 = r_{12}^1(r_{22}^1)^* r_{23}^1 + r_{13}^1 = 1 + 0(\varepsilon+00)^*(1+01).$$

Because of $(\varepsilon+00)^* = (00)^*$ and $1+01 = (\varepsilon+0)1$ this can be rewritten as

$$r_{13}^2 = 1 + 0(00)^*(\varepsilon+0)1.$$

Further, one has $(00)^*(\varepsilon+0) = 0^*$, so that

$$r_{13}^2 = 1 + 00^*1 = 0^*1.$$

A regular expression for the language accepted by this automaton is $r_{12}^3 + r_{13}^3$. Each of the summands is a lengthy expression. After some simplifications one obtains

$$r = 0^*1((0+1)0^*1)^*(\varepsilon + (0+1)(00)^*) + 0(00)^*.$$

It should be noted that one does not need all of the table 13 to compute the expression r.

3 Properties of regular languages

3.1 Closure under boolean operations

Theorem 3.1. *Let R, R_1, R_2 be any regular languages in the alphabet Σ. Then the languages $R_1 \cup R_2, R_1 \cap R_2, \Sigma^* \setminus R$ are also regular.*

Proof. If r_1 is a regular expression for R_1, and r_2 is a regular expression for R_2, then the regular expression $r_1 + r_2$ describes the language $R_1 \cup R_2$, which is therefore regular.

Given regular expressions r_1, r_2, r, it is very difficult to find regular expressions for the intersection $R_1 \cap R_2$ and the complement $\Sigma^* \setminus R$. Let us approach the problem from a different direction.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting the language R. Then $\overline{M} := (Q, \Sigma, \delta, q_0, Q \setminus F)$ accepts the language $\Sigma^* \setminus R$. Indeed,

$$w \in \Sigma^* \setminus R \Leftrightarrow w \notin R \Leftrightarrow \widehat{\delta}(q_0, w) \notin F \Leftrightarrow \widehat{\delta}(q_0, w) \in Q \setminus F \Leftrightarrow w \in L(\overline{M}).$$

Therefore $\Sigma^* \setminus R$ is regular.

With the intersection we are helped by de Morgan's rule:

$$R_1 \cap R_2 = \overline{\overline{R_1} \cup \overline{R_2}},$$

where the overline denotes the complement. Since the operations applied on the right hand side preserve regularity, the intersection of two regular languages is regular. □

Theorem 3.2. *The equivalence of finite automata and the equivalence of regular expressions is decidable. (That is, there is an algorithm for each of these problems, which gives a correct answer in finite time.)*

Proof. We have an algorithm which converts a regular expression into a finite automaton. Therefore it suffices to find an algorithm for the equivalence of finite automata.

Given two finite automata M_1 and M_2, let $L_1 = L(M_1)$ and $L_2 = L(M_2)$. By Theorem 3.1, the symmetric difference

$$L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$$

is a regular language. Thus there is a finite automaton M that accepts the language $L_1 \triangle L_2$. One has

$$L_1 = L_2 \Leftrightarrow L_1 \triangle L_2 = \varnothing.$$

Therefore it suffices to decide whether the language accepted by M is empty. The language is empty if and only if from the initial state no final state can be reached. This is easy to check algorithmically. □

3.2 The pumping lemma

Let Σ be a finite alphabet. The set Σ^* of all words in Σ is countably infinite. The set 2^{Σ^*} of all languages in Σ is uncountable: it has the cardinality of the continuum. On the other hand, the set of regular languages is countable, because the set of all regular expressions (and of all finite automata) is countable. (Note that different automata or different regular expressions can define the same language, but this is not a problem.) It follows that "most" languages are non-regular.

The above argument is a pure existence proof. In this section we prove the *pumping lemma*, a tool that allows to prove the non-regularity of some languages.

Before stating the lemma, let us explain the underlying idea. Let M be a DFA. Any word z accepted by M determines a path from the initial state q_0 to one of the finals states $q \in F$. If the word z is long enough (namely if its length is bigger than the number of states of M), then the corresponding path contains a cycle. This cycle gives rize to infinitely many other words accepted by M, because one can run along it several times. For example, in Figure 14 one has $z = a_1a_2a_3a_4a_5a_6a_7 \in L(M)$. By repeating or removing the cycle contained in the path, one obtains

$$a_1a_2(a_3a_4a_5)^k a_6a_7 \in L(M)$$

for all k, including $k = 0$.

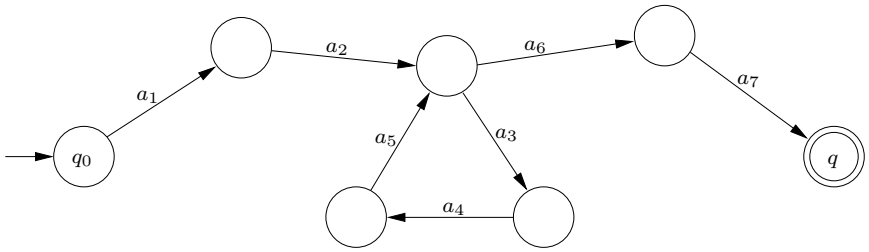

Figure 14: Idea of the pumping lemma: a long word contains a subword which can be repeated.

The existence of a cycle can be stated as follows.

Lemma 3.3. *Any path of length $\geq n$ (the length of a path is the number of edges) in a graph with n vertices contains a cycle. Besides, there is a cycle within the first n edges of this path.*

We are now ready to state and prove the pumping lemma.

Theorem 3.4 (Pumping lemma for regular languages)**.** *Let L be a regular language. Then there is an integer n such that for any word $z \in L$ of length $|z| \geq n$ the word z can be represented as $z = uvw$ in such a way that*

$$|uv| \leq n, \quad |v| \geq 1, \quad \text{and for all } k \geq 0 \text{ one has } uv^k w \in L.$$

Proof. Since L is a regular language, there is a DFA accepting it. Let n be the number of states of such a DFA. Then any word $z \in L$ determines a path of length $|z|$ from q_0 to one of the final states. By Lemma 3.3, if $|z| \geq n$, then within the first n edges of this path there is a cycle. Let u be the prefix of z before the beginning of this cycle, let v be the subword corresponding

to the cycle, and let w be the remaining suffix. Then $z = uvw$, and the path determined by the word $uv^k w$ differs from the path determined by uvw in the number of times it runs along the cycle of v, but has the same endpoint, a final state of the DFA. Thus $uv^k w \in L$ for all $k \geq 0$. □

Example 3.5. Take the alphabet of one symbol $\Sigma = \{0\}$ and consider the language $L = \{0^{k^2} \mid k \text{ is a positive integer}\}$: all sequences of 0's whose length is a perfect square. Let us show that L is not regular. Assume the contrary, and let n be the integer in the pumping lemma. Let $z = 0^{n^2}$. By the pumping lemma, $z = uvw$, where $1 \leq |v| \leq |uv| \leq n$, and $uv^k w \in L$ for all k. For $k = 2$ one has $n^2 + 1 \leq |uv^2 w| \leq n^2 + n < (n+1)^2$, thus $uv^2 w \notin L$. This contradiction shows that our assumption was false, and L is not regular.

The only property of the sequence of perfect squares that was used is the existence of arbitrarily large gaps. Therefore any language of the form $\{0^{a_k}\}$, where a_k is a monotone sequence of integers such that for every n there is k such that $a_{k+1} - a_k > n$, is non-regular. Later we will be able to show that the only regular languages of the form $\{0^{a_k}\}$ are those for which the sequence a_k is periodic.

Example 3.6. The language L consisting of all binary words with an equal number of zeros and ones is not regular. Indeed, assume the contrary, and let n be the integer in the pumping lemma. Then $0^n 1^n \in L$. By the pumping lemma, one can write $0^n 1^n = uvw$, where $|uv| \leq n$, and $uv^k w \in L$ for all k. It follows that the words u and v consist of zeros only. Since $|v| \geq 1$, the word $uw = uv^0 w$ has less zeros than ones, thus it does not belong to L. This contradiction shows that our assumption was false.

The pumping lemma can be used to prove the following theorem.

Theorem 3.7. *A language accepted by a DFA with n states is*

1. *nonempty if and only if the automaton accepts some word of length less than n;*

2. *infinite if and only if the automaton accepts some word of length ℓ, where $n \leq \ell < 2n$.*

Proof. 1) The path corresponding to the shortest accepted word does not visit any state more than once. Otherwise it contains a cycle, and by removing this cycle we obtain a shorter accepted word. Therefore the length of the shortest accepted word is strictly less than n.

2) *The "if" direction.* If z is an accepted word of length $\geq n$, then its path contains a cycle. By pumping this cycle, we obtain infinitely many accepted words.

The "only if" direction. If the language is infinite, then it contains a word z with $|z| \geq n$. If $|z| < 2n$, then we are done. If $|z| \geq 2n$, then write z as $z = uvw$ according to the pumping lemma. Then uw is also accepted, and we have $|uw| = |z| - |v| \geq |z| - n \geq n$. Thus we can apply the same case distiction to the word uw and proceed until we get a word of length $\geq n$ and $< 2n$. □

3.3 Closure under homomorphisms

Let Σ and Δ be two finite alphabets.

Definition 3.8. *A* homomorphism *is a map* $h\colon \Sigma^* \to \Delta^*$ *such that*

$$h(xy) = h(x)h(y) \text{ for all } x, y \in \Sigma^*.$$

Lemma 3.9. *A homomorphism is uniquely determined by the images of the letters of the alphabet* Σ*. That is, any* $h\colon \Sigma \to \Delta^*$ *extends to a unique homomorphism.*

Proof. Let us prove the uniqueness. If we know $h(a)$ for all $a \in \Sigma$, then we have no other choice but put

$$h(a_1 \dots a_n) = h(a_1) \dots h(a_n).$$

Also by definition of a homomorphism one has

$$h(x) = h(\varepsilon x) = h(\varepsilon)h(x),$$

which implies $h(\varepsilon) = \varepsilon$. Thus there is no more than one homomorphism with given values on the letters of the alphabet.

On the other hand, putting $h(a_1 \dots a_n) = h(a_1) \dots h(a_n)$ and $h(\varepsilon) = \varepsilon$ defines a homomorphism, so the extension exists and is unique. □

Definition 3.10. *Let* $L \subset \Sigma^*$ *be a language. The* homomorphic image *of* L *is the language*

$$h(L) = \{h(w) \mid w \in L\} \subset \Delta^*,$$

where $h\colon \Sigma^* \to \Delta^*$ *is some homomorphism.*

Example 3.11.

- $\Sigma = \Delta = \{0\}$, $L = \Sigma^*$, $h(0) = 00$. Then $h(L)$ consists of all even length sequences of zeros.
- $\Sigma = \Delta = \{0, 1\}$, $L = \Sigma^*$, $h(0) = 0, h(1) = 10$. For every word w its image $h(w)$ is obtained by inserting a 0 after every 1. The language $h(L)$ consists of all words without two consecutive 1's and not ending with 1.

Theorem 3.12. *A homomorphic image of a regular language is regular.*

Proof. Let L be a regular language, and let r be a regular expression representing L. Make a substitution in r, replacing every symbol by its image under the homomorphism. The result is a regular expression in the alphabet Δ; denote it by $h(r)$. Then the language defined by $h(r)$ is $h(L)$, thus $h(L)$ is regular.

The claim $L(h(r)) = h(L(r))$ is proved by induction on the complexity of the expression r. Here is the induction step to $h = h_1 + h_2$:

$$\begin{aligned} L(h(r_1 + r_2)) &= L(h(r_1) + h(r_2)) = L(h(r_1)) \cup L(h(r_2)) \\ &= h(L(r_1)) \cup h(L(r_2)) = h(L(r_1) \cup L(r_2)) = h(L(r_1 + r_2)). \end{aligned}$$

□

For example, if $h(0) = 0$ and $h(1) = 10$, then $h((0 + 1)^*) = (0 + 10)^*$.

The next example shows that a homomorphic image of a non-regular language can be regular.

Example 3.13. The language $\{0^k 1^k \mid k \geq 0\}$ is non-regular, as can be shown with the help of pumping lemma. But under the homomorphism $h(0) = 0, h(1) = 0$ it goes to the language $\{0^{2k} \mid k \geq 0\}$, which is regular.

Lemma 3.14. *The language of all propositional formulas is not regular.*

Proof. Let Σ be the alphabet of the propositional logic. Consider the homomorphism $h\colon \Sigma^* \to \{0,1\}^*$ defined by

$$(\mapsto 0, \quad) \mapsto 1, \quad x \mapsto \varepsilon \text{ for all other symbols of } \Sigma.$$

Every propositional formula is sent to a balanced bracket sequence (now represented by zeros and ones), and every balanced bracket sequence is the image of some propositional formula. If the language of all propositional formulas is regular, then the language of all balanced bracket sequences is also regular. But it is not, by the pumping lemma: if n is the number from the lemma, then $0^n 1^n = uvw$ belongs to the language, but uv^2w is unbalanced. □

3.4 Closure under inverse homomorphism

Definition 3.15. *Let $h\colon \Sigma^* \to \Delta^*$ be a homomorphism, and $L \subset \Delta^*$ be a language in the alphabet Δ. The* inverse homomorphic image *of L is*

$$h^{-1}(L) = \{x \in \Sigma^* \mid h(x) \in L\}.$$

Example 3.16. Let $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$, and $h(a) = 01$, $h(b) = 10$. Then

$$h^{-1}(1001) = \{ba\}, \quad h^{-1}(0011) = \varnothing.$$

For $L = (00{+}1)^*$ one has $h^{-1}(L) = (ba)^*$ (check this!). One has $h(h^{-1}(L)) = (1001)^* \neq L$.

One always has

$$h(h^{-1}(L)) \subset L \subset h^{-1}(h(L)),$$

and the inclusions can be strict.

Theorem 3.17. *An inverse homomorphic image of a regular language is regular.*

Proof. Let $L \subset \Delta^*$ be a regular language, and $h\colon \Sigma^* \to \Delta^*$ a homomorphism. Let $M = (Q, \Delta, \delta, q_0, F)$ be a DFA accepting L. We will construct a DFA accepting $h^{-1}(L)$ thus proving that this language is regular. The idea is to use the same set of states and the same set of final states, but interpret each symbol $a \in \Sigma$ as $h(a) \in \Delta$. Then a word $w \in \Sigma^*$ will be accepted by the new automaton if and only if $h(w)$ was accepted by the old one. Formally, put

$$M' = (Q, \Sigma, \delta', q_0, F), \text{ where } \delta'(q, a) = \widehat{\delta}(q, h(a)).$$

It can be shown by induction on the length of a word w that $\widehat{\delta'}(q, w) = \widehat{\delta}(q, h(w))$. Thus we have

$$w \in L(M') \Leftrightarrow \widehat{\delta'}(q, w) \in F \Leftrightarrow \widehat{\delta}(q, h(w)) \in F \Leftrightarrow h(w) \in L \Leftrightarrow w \in h^{-1}(L),$$

which means $L(M') = h^{-1}(L)$. □

4 The Myhill-Nerode theorem

4.1 Equivalence of words with respect to a language

Definition 4.1. *Let $L \subset \Sigma^*$ be a language. Two words $u, v \in \Sigma^*$ are called* L-equivalent *(written as $u \sim_L v$) if*

$$\forall x \in \Sigma^* \text{ either } ux, vx \in L \text{ or } ux, vx \notin L.$$

In other words, u and v are *not L-equivalent* if they have a *distinguishing extension*: a word $x \in \Sigma^*$ such that one of the words ux, vx is in L, and the other not.

Example 4.2. If $u \in L$ and $v \notin L$, then $u \not\sim_L v$. Indeed, ε is a distinguishing extension for u and v.

Lemma 4.3. *The relation $\sim_L$ is an equivalence relation.*

Proof. The reflexivity and the symmetry are obvious. To prove the transitivity, let $u \sim_L v$, $v \sim_L w$, and assume that $u \not\sim_L w$. Then there is a distinguishing extension x for u and w. Without loss of generality, $ux \in L$, $wx \notin L$. Then if $vx \in L$, we have $v \not\sim_L w$, and if $vx \notin L$, we have $u \not\sim_L v$. □

An equivalence relation splits the set Σ^* of all words into *equivalence classes*:

$$\Sigma^* = S_0 \cup S_1 \cup S_2 \cup \cdots \tag{3}$$

where $u \sim_L v$ if and only if u and v belong to the same class. As we noticed in Example 4.2, if $u \sim_L v$, then either $u, v \in L$ or $u, v \notin L$. It follows that every class S_i is either contained in L or disjoint from L.

Example 4.4. Let L consist of all binary words with the number of zeros not divisible by 3. Then $u \sim_L v$ if and only if the number of zeros in u and v has the same remainder under division by 3:

$$\Sigma^* = S_0 \cup S_1 \cup S_2, \quad S_i = \{u \mid \ell_0(u) \equiv i (\mathrm{mod}\, 3)\}.$$

One has $L = S_1 \cup S_2$.

Example 4.5. Let L be the set of all binary words with equal numbers of zeros and ones. Then $u \sim_L v$ if and only if the difference between the numbers of zeros and ones in u and in v is the same:

$$\Sigma^* = \cdots \cup S_{-2} \cup S_{-1} \cup S_0 \cup S_1 \cup S_2 \cup \cdots, \quad S_i = \{u \mid \ell_0(u) - \ell_1(u) = i\}.$$

One has $L = S_0$.

Lemma 4.6. *If $u \sim_L v$, then $ux \sim_L vx$ for all $x \in \Sigma^*$.*

Proof. Assume that $ux \not\sim_L vx$, and let y be a distinguishing extension. Then xy is a distinguishing extension for u and v. □

4.2 The theorem

Theorem 4.7. *A language L is regular if and only if the number of L-equivalence classes is finite.*

Proof. Assume that L is regular. Take a DFA that accepts L. Let $q_0, \ldots, q_n$ be its states, and q_0 be the initial state. Denote

$$T_i = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) = q_i\}.$$

One has

$$\Sigma^* = T_0 \cup T_1 \cup \cdots T_n.$$

We claim that every T_i is a subset of some S_j from the decomposition (3). In other words, every S_j is the union of one or several T_i, which means that the number of $\sim_L$-equivalence classes is at most $n + 1$ and implies the first part of the theorem.

In order to prove the claim it suffices to show that if u and v belong to the same T_i, then $u \sim_L v$. Then for every $x \in \Sigma^*$ one has

$$\widehat{\delta}(q_0, ux) = \widehat{\delta}(\widehat{\delta}(q_0, u), x) = \widehat{\delta}(q_i, x) = \widehat{\delta}(\widehat{\delta}(q_0, v), x) = \widehat{\delta}(q_0, vx).$$

Since both words ux and vx bring us to the same state, they either both belong to L (if this state is final) or both not belong to L (if this state is not final). Thus $u \sim_L v$.

In the opposite direction, let $\Sigma^* = S_0 \cup \cdots \cup S_n$, where $\varepsilon \in S_0$. Construct a DFA with states $q_0, \ldots, q_n$, the initial state q_0, and the transition function defined as follows. To find $\delta(q_i, a)$, take some $u \in S_i$ and look in which class the word ua lies. If $ua \in S_j$, then put $\delta(q_i, a) = q_j$. The result is independent of the choice of a representative $u \in S_i$. Indeed, by Lemma 4.6 $u \sim_L v \Rightarrow ua \sim_L va$. A state q_i is designated as final if and only if $S_i \subset L$. It is easy to see that the language accepted by this automaton is L. □

4.3 Minimization of a DFA

Myhill-Nerode theorem implies

Corollary 4.8. *The minimum number of states in a DFA accepting a regular language L is equal to the number of L-equivalence classes. The minimal DFA is unique up to renaming the states.*

From any DFA one can construct the minimum DFA accepting the same language by merging certain sets of states into one state. Two states q_i, q_j must be merged if the corresponding sets T_i, T_j are contained in the same equivalence class S_k. That is, $q_i \sim q_j$ if for all $x \in \Sigma^*$ either both $\widehat{\delta}(q_i, x)$ and $\widehat{\delta}(q_j, x)$ belong to F or both do not.

One can certify non-equivalence of two states by finding a word x such that $\widehat{\delta}(q_i, x) \in F$ and $\widehat{\delta}(q_j, x) \notin F$ or vice versa. The algorithm marks pairs of distinguishable states recursively.

At the very beginning one removes all inaccessible states. Obviously, this does not change the accepted language.

Then one draws a table whose rows and columns are marked by the remaining states. As initialization, one marks all pairs (q_i, q_j) such that $q_i \in F$ and $q_j \notin F$ or vice versa. At each of the following steps one goes through all pairs (q_i, q_j), and for each of them one considers all alphabet letters a. If the pair $(\delta(q_i, a), \delta(q_j, a))$ was marked at one of the previous steps, then one marks the pair (q_i, q_j). If at some step no new pairs are marked, then the algorithm stops. All pairs of states which are unmarked are merged into a single state (it is also possible that several states are merged into one state).

Example 4.9. Consider the DFA in Figure 15. It has one inaccessible state q_3.

We thus draw a 7×7 table and fill it step by step. It suffices to fill only a half of the table, on one side of the diagonal.

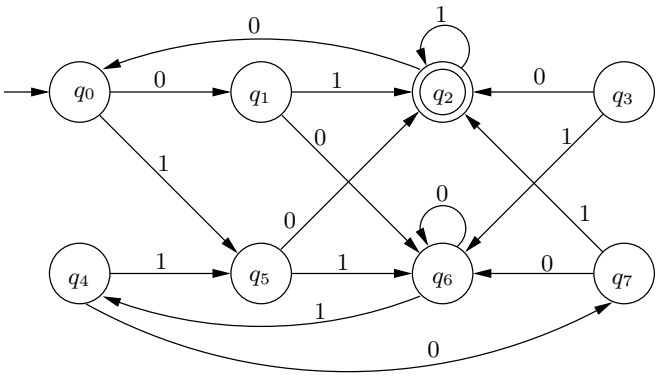

Figure 15: A non-minimal DFA.

	q_0	q_1	q_2	q_4	q_5	q_6	q_7
q_0							
q_1	$\times_1$						
q_2	$\times_0$	$\times_0$					
q_4		$\times_1$	$\times_0$				
q_5	$\times_1$	$\times_1$	$\times_0$	$\times_1$			
q_6	$\times_2$	$\times_1$	$\times_0$	$\times_2$	$\times_1$		
q_7	$\times_1$		$\times_0$	$\times_1$	$\times_1$	$\times_1$	

A pair of states is marked with $\times_n$ if these states were recognized as non-equivalent at Step n. As initialization (Step 0) we mark all pairs (q_2, q_i) because q_2 is the only final state.

A lot of cells are marked at Step 1. These are all pairs (q_i, q_j) such that either the 0-arrows or the 1-arrows lead from q_i to a final and from q_j to a non-final state or vice versa. For example, we mark (q_0, q_1) because $\delta(q_0, 1) = q_5$ is non-final and $\delta(q_1, 1) = q_2$ is final.

At Step 2 two cells are marked. For example, we mark (q_4, q_6) because $\delta(q_4, 0) = q_7$, $\delta(q_6, 0) = q_6$, and the pair (q_6, q_7) is already marked (it was marked at Step 1).

At Step 3 we check all pairs of unmarked cells, by looking where the 0- and 1-arrows lead, but do not find anything that should be marked. Thus the algorithm stops, and the minimal DFA is obtained from the one in Figure 15 by removing the state q_3, merging q_0 with q_4 and merging q_1 with q_7. See Figure 16.

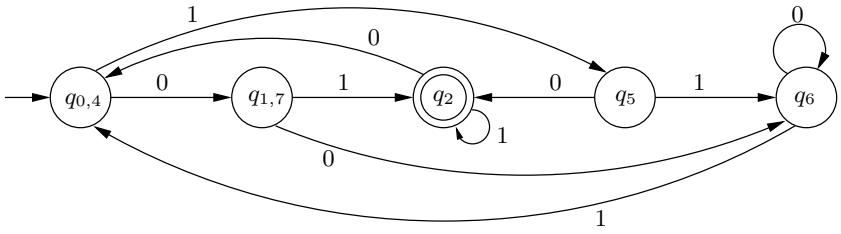

Figure 16: The minimal DFA equivalent to one in Figure 15.

5 Context-free grammars and languages

5.1 Generating a language by a grammar

Consider the sentence

Colorless green ideas sleep furiously.

Although it does not make any sense, it is syntactically correct. One distinguishes a noun phrase "colorless green ideas" and a verb phrase "sleep furiously". The noun phrase itself consists of a noun preceeded by adjectives, and the verb phrase consists of a verb followed by adverbs.

More generally, one can formulate the following rules of production of simple English sentences:

$S \to NV$	a sentence consists of a noun phrase and a verb phrase
$N \to AdjN$	a noun phrase may start with one or more adjectives
$V \to VAdv$	a verb phrase may end with one or more adverbs

At any stage one can substitute for N, V, Adj, Adv a word from a dictionary:

$$\begin{aligned} N &\to \text{list of nouns} \\ V &\to \text{list of verbs} \\ Adj &\to \text{list of adjectives} \\ Adv &\to \text{list of adverbs} \end{aligned}$$

The result is a syntactically correct (but mostly meaningless) sentence.

A production can be represented linearly:

$$\begin{aligned} S \to NV \to AdjNV \to AdjAdjNV \to \text{colorless } AdjNV \\ \to \text{colorless } AdjNVAdv \to \cdots \end{aligned}$$

or by a *derivation tree* or *parse tree*, see Figure 17.

The sentence at the beginning of this section is from a book of Noam Chomsky. In mid-1950's he proposed the above principles as description of

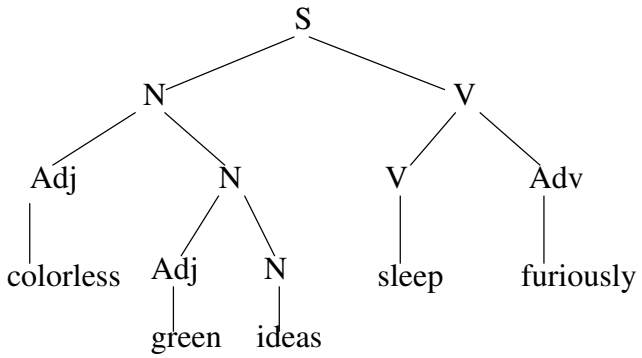

Figure 17: Derivation tree for the Chomsky example.

the structure of human languages. (Of course one needs more production rules in order to be able to generate more complicated sentences.) A couple of years later John Backus, a programming language designer acquainted with Chomsky's ideas, described the syntax of the ALGOL programming language in a similar way.

Let us now give an exact definition.

Definition 5.1. *A* context-free grammar *(or* CFG *or just* grammar*) is a quadruple* $G = (V, T, P, S)$*, where*

- V *is a finite set of* variables*;*
- T *is a finite set of* terminals*;*
- P *is a finite set of productions, each production is of the form* $A \to \alpha$*, where* $\alpha \in (V \cup T)^*$*;*
- S *is a special variable* ($S \in V$) *called the* start symbol*.*

Example 5.2. In our introductory example $V = \{S, N, V, Adj, Adv\}$, T is the set of words in a dictionary, and the productions P are as stated above.

Note that the alphabet V of variables and the alphabet T of terminals must be disjoint. To avoid confusion, we will use different symbols in the following way.

- The capital letters $A, B, C, \ldots$ are variables.
- The letters a, b, c and digits are terminals.
- The letters X, Y, Z denote symbols that may be variables or terminals.
- The letters u, v, w, x, y, z are used to denote strings of terminals, that is elements of T^*.

- The letters α, β, γ are used to denote strings of variables and terminals, that is elements of $(V \cup T)^*$.

Definition 5.3. *The* language generated by a grammar *is the set of all words in the alphabet T that can be derived from the start symbol S according to the production rules.*

In order to describe formally what it means that a word can be derived from the start symbol, let us fix some notations and terminology. If $A \to \beta$ is any production in P, and $\alpha, \gamma \in (V \cup T)^*$, then we write $\alpha A \gamma \underset{G}{\Longrightarrow} \alpha\beta\gamma$ and say that $\alpha A \gamma$ *directly derives* $\alpha\beta\gamma$ in grammar G. If $\alpha_1, \ldots, \alpha_n \in (V \cup T)^*$ are such that

$$\alpha_1 \underset{G}{\Longrightarrow} \alpha_2, \quad \alpha_2 \underset{G}{\Longrightarrow} \alpha_3, \quad \ldots, \quad \alpha_{n-1} \underset{G}{\Longrightarrow} \alpha_n,$$

then we write $\alpha_1 \overset{*}{\underset{G}{\Longrightarrow}} \alpha_n$ and say that α_1 *derives* α_n in G. When it is clear which grammar we use, then we omit G and write $\Rightarrow$ and $\overset{*}{\Rightarrow}$, respectively.

Now we can describe the language generated by G as

$$L(G) = \{w \in T^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}.$$

A language is called a *context-free language* (CFL) if it is generated by some context-free grammar.

Example 5.4. Let $V = \{S\}$, $T = \{0, 1\}$, $P = \{S \to 0S1, S \to \varepsilon\}$. From

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow \cdots \Rightarrow 0^{n-1}S1^{n-1} \Rightarrow 0^n1^n$$

we see that $L(G) = \{0^n1^n \mid n \geq 0\}$.

Note that the language $\{0^n1^n \mid n \geq 1\}$ is not regular, as can be shown with the help of the pumping lemma. We will later see that every regular language is context-free.

Example 5.5. The language of all binary words with equal numbers of 0's and 1's is context free. However, the generating grammar is more complicated, see [7, Example 4.3].

5.2 Grammars in Chomsky form

A grammar is said to be in *Chomsky form* if all of its productions are of the form $A \to BC$ and $A \to a$. According to our notation convention, A, B, C are variables and a is a terminal. Note that the "English grammar" from the beginning of Section 5.1 is in Chomsky form.

Theorem 5.6. *Any context-free language without ε is generated by a grammar in Chomsky form.*

Two grammars are called *equivalent* if they generate the same language. Thus the above theorem can also be stated as "for every grammar whose language does not contain ε there is an equivalent grammar in Chomsky form".

Definition 5.7. *An* ε-production *is a production of the form* $A \to \varepsilon$. *A* unit production *is a production of the form* $A \to B$.

Without ε-productions it is impossible to produce the empty word, so we cannot get rid of them if the language contains ε. On the other hand, if a grammar contains ε-productions, this does not necessarily mean that the generated language contains ε.

Unit productions can be helpful, they introduce sort of "branching". For example, the grammar

$$S \to A \mid B, \quad A \to AA \mid 0, \quad B \to BB \mid 1$$

is a simple grammar generating the language $(0^* \cup 1^*) \setminus \{\varepsilon\}$.

Lemma 5.8. *Any context-free language without* ε *is generated by a grammar without* ε*-productions and without unit productions.*

Proof. Let G be a context-free grammar possibly containing ε- and unit productions. We construct a grammar G' without ε-productions such that $L(G') = L(G) \setminus \{\varepsilon\}$.

First, identify *nullable* variables, those which derive ε. This is done recursively. Initialize the set of nullable variables by those A for which there is a production $(A \to \varepsilon) \in P$. The recursion step adds to the set of nullable variables those B for which $(B \to C_1 \cdots C_k) \in P$ and all $C_1, \dots C_k$ are nullable. As soon as this recursion does not find new nullable variables, the algorithm stops.

Second, remove from P all ε-productions $A \to \varepsilon$ and add new productions in the following way. Let $A \to X_1 \cdots X_n$ be a production with some of X_i nullable variables (recall that X_i can stand for a variable symbol as well as for a terminal symbol). We add all productions of the form $A \to X_1 \widehat{\cdots} X_n$, where $\widehat{\cdots}$ means that we remove any subset of nullable variables (with one exception: if all $X_1, \dots, X_n$ are nullable, then we do not add the production $A \to \varepsilon$ obtained by removing all nullable variables). That is, if m symbols among $X_1, \dots X_n$ are nullable variables, then the production $A \to X_1 \cdots X_n$ gives rise to 2^m productions if $m < n$ and to $2^n - 1$ productions if $m = n$.

It can be checked that the new set of productions allows to derive all words (except ε) which were derivable with the initial set of productions, and only those words.

Now we construct a grammar G'' equivalent to G' but without unit productions. Call a pair (A, B) *unit pair*, if $A \overset{*}{\Rightarrow} B$. The set of all unit pairs can be found recursively.

Remove all unit productions, and for each unit pair (A, B) and each non-unit production $B \to \alpha$ add the production $A \to \alpha$. Every word generated by the grammar G'' is also generated by G': the new direct productions $A \underset{G''}{\Longrightarrow} \alpha$ are compositions of two old productions $\alpha A\beta \underset{G'}{\Longrightarrow} \alpha B\beta \underset{G'}{\Longrightarrow} \alpha$. Every word generated by G' is generated by G'': a series of unit productions must always end with a non-unit production, so if we had $\alpha A\gamma \overset{*}{\underset{G'}{\Longrightarrow}} \alpha B\gamma \underset{G'}{\Longrightarrow} \alpha\beta\gamma$, then we have $\alpha A\gamma \underset{G''}{\Longrightarrow} \alpha\beta\gamma$. Thus the new set of productions generates the same language as before. □

It is important to remove first the ε-productions and then the unit productions. If first the unit, and then ε-productions are removed, then the result might contain unit productions.

Example 5.9. Let us remove the ε-productions from the grammar

$$\begin{aligned} S &\to ABA \\ A &\to aA \mid \varepsilon \\ B &\to bB \mid b. \end{aligned}$$

Only the variable A is nullable. Remove the production $A \to \varepsilon$. The nullable variable A appears on the right hand side of the productions $S \to ABA$ and $A \to aA$. Add new productions by removing any number of A's from the right hand sides of these productions:

$$\begin{aligned} S &\to ABA \mid AB \mid BA \mid B \\ A &\to aA \mid a \\ B &\to bB \mid b. \end{aligned}$$

Example 5.10. Let us remove the unit productions from the grammar

$$\begin{aligned} S &\to A \mid B \\ A &\to AA \mid 0 \\ B &\to BB \mid 1. \end{aligned}$$

The unit pairs are (S, A) and (S, B). Remove the unit productions and introduce all $S \to \alpha$ for which $A \to \alpha$ or $B \to \alpha$:

$$\begin{aligned} S &\to AA \mid BB \mid 0 \mid 1 \\ A &\to AA \mid 0 \\ B &\to BB \mid 1. \end{aligned}$$

Proof of Theorem 5.6. Let L be a language without ε. By Lemma 5.8 there is a grammar G without ε and unit productions such that $L = L(G)$. If

a production of G has a single symbol on the right, then this symbol is a terminal, so the production is of the form $A \to a$.

Any other production of G has the form

$$A \to X_1 X_2 \cdots X_n, \quad n \geq 2,$$

where every X_i is either a variable or a terminal. If $X_i = a$ is a terminal, then introduce a new variable C_a and a new production $C_a \to a$. In the "long" ($n \geq 2$) right hand sides of all productions replace a by C_a. Clearly, the new grammar G' generates the same language as the old one.

In the grammar G', all productions are of the form $A \to a$ or $A \to B_1 \cdots B_n$, $n \geq 2$. Create a new grammar G'' by introducing for each production of a "long" ($n \geq 3$) word a new set of variables $D_1, \ldots, D_{n-2}$ and replacing this production by a set of productions

$$A \to B_1 D_1, \quad D_1 \to B_2 D_2, \ldots, \quad D_{n-3} \to B_{n-2} D_{n-2}, \quad D_{n-2} \to B_{n-1} B_n.$$

Again, it is not hard to convince yourself that the new grammar generates the same language. □

Example 5.11. Let us find a Chomsky form of the grammar

$$\begin{aligned} S &\to aB \mid bA \\ A &\to a \mid aS \mid bAA \\ B &\to b \mid bS \mid aBB \end{aligned}$$

By introducing variables C_a and C_b we get rid of terminals in long words:

$$\begin{aligned} S &\to C_a B \mid C_b A \\ A &\to a \mid C_a S \mid C_b AA \\ B &\to b \mid C_b S \mid C_a BB \\ C_a &\to a \\ C_b &\to b \end{aligned}$$

There are two productions $A \to C_b AA$ and $B \to C_a BB$ which have words of length ≥ 3 on the right hand side. Replace them by two short productions each:

$$\begin{aligned} S &\to C_a B \mid C_b A \\ A &\to a \mid C_a S \mid C_b D_1 \\ B &\to b \mid C_b S \mid C_a D_2 \\ C_a &\to a \\ C_b &\to b \\ D_1 &\to AA \\ D_2 &\to BB \end{aligned}$$

6 Pushdown automata

6.1 Definition

A pushdown automaton is similar to a finite automaton: it has a finite number of states and changes the state according to the input. But it has additional memory, in the form of a stack of unbounded depth. At every step the automaton reads the input symbol and the top symbol of the stack. According to these data, the automaton changes its state and rewrites the top of the stack.

We proceed to a formal definition.

Definition 6.1. *A* pushdown automaton *or a PDA is a system* $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, where*

- Q *is a (finite) set of states,* $q_0 \in Q$ *the initial state,* $F \subset Q$ *the set of final states;*
- Σ *is the input alphabet,* Γ *is the stack alphabet,* $Z_0 \in \Gamma$ *is a special symbol called the* start symbol*;*
- δ *is a map from* $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ *to finite subsets of* $Q \times \Gamma^*$*.*

At the beginning, the automaton is in the state q_0, and the stack contains the symbol Z_0.

The transition function δ takes three arguments: the current state of the automaton, an input symbol or ε, and the top symbol of the stack. The value of the transition function

$$\delta(q, a, Z) = \{(p_1, \gamma_1), \ldots, (p_m, \gamma_m)\}, \quad \gamma_i \in \Gamma^*,$$

is interpreted as follows. If the automaton in the state q reads the input symbol a and sees the symbol Z on the top of the stack, then, for a random i, it enters the state p_i and replaces the symbol Z by the string γ_i.

Example 6.2. The rule $\delta(q, a, Z) = \{(p_1, BZ), (p_2, \varepsilon)\}$ means that the automaton can either go to state p_1 and put B into the stack or go to state p_2 and take Z out of the stack.

By definition, ε-transitions are allowed:

$$\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), \ldots, (p_m, \gamma_m)\}$$

means that from the state q and with Z on the top of the stack the automaton can do a spontaneous transition to a state p_i and replace Z with γ_i.

Example 6.3. The rule $\delta(q, \varepsilon, Z) = \varnothing$ means that no spontaneous transition (from the state q with Z on the top) is allowed.

There are two versions of acceptance criteria. In the first, a word w is accepted if some sequence of moves corresponding to the input w leads to an empty stack (which means that the start symbol Z_0 on the bottom of the stack should also be removed). In this case the set of final states is irrelevant, one may put $F = \varnothing$.

In the second version, a word w is accepted if some sequence of moves corresponding to the input w brings the automaton to a final state.

A PDA is *deterministic* if every set $\delta(q, a, Z) \cup \delta(q, \varepsilon, Z)$ contains at most one element (from every state and every input symbol at most one move is possible, taking ε-transitions into account). Note however that $\delta(q, a, Z) = \varnothing$ means that in the state q with top stack symbol Z the input symbol a is rejected (or "breaks" the automaton).

Example 6.4. Consider the language of binary palindromes with a symbol c ("center") in the middle:

$$L = \{wc\bar{w} \mid w \in \{0,1\}^*\}.$$

We describe a deterministic PDA accepting L by the empty stack. Put

$$M = (\{q_1, q_2\}, \{0, 1, c\}, \{A, B, Z_0\}, \delta, q_1, Z_0, \varnothing).$$

While the automaton is in the state q_1, it stores the input in the stack encoding 0 with A and 1 with B:

$$\delta(q_1, 0, Z) = (q_1, AZ) \quad \delta(q_1, 1, Z) = (q_1, BZ) \text{ for all } Z \in \Gamma.$$

If the input symbol is c, then the automaton switches to the state q_2:

$$\delta(q_1, c, Z) = (q_2, Z) \text{ for all } Z \in \Gamma.$$

While in the state q_2, the automaton compares the input symbol with the top symbol in the stack. If the symbols agree, then the top symbol is removed; if they disagree, the automaton "breaks down".

$$\delta(q_2, 0, A) = (q_2, \varepsilon) \quad \delta(q_2, 1, B) = (q_2, \varepsilon)$$

Finally, when the automaton sees the start symbol at the bottom of the stack, this symbol is removed, and the word is accepted:

$$\delta(q_2, \varepsilon, Z_0) = (q_2, \varepsilon).$$

Contrarily to DFA, non-deterministic PDAs are stronger than deterministic ones. The language of binary palindromes of even length cannot be accepted by a deterministic PDA but is accepted by a non-deterministic one as the next example shows.

Example 6.5. A PDA accepting the language $\{w\bar{w} \mid w \in \{0,1\}^*\}$ by the empty stack.

$$M = (\{q_1, q_2\}, \{0,1\}, \{A, B, Z_0\}, \delta, q_1, Z_0, \varnothing)$$

The principle is the same: while in the state q_1, we encode the input by putting into the stack A for the input symbol 0 and B for the input 1:

$$\delta(q_1, 0, Z_0) = (q_1, AZ_0), \quad \delta(q_1, 0, B) = (q_1, AB)$$
$$\delta(q_1, 1, Z_0) = (q_1, BZ_0), \quad \delta(q_1, 1, A) = (q_1, BA)$$

However, if the input symbol agrees with the top stack symbol, then this might be the middle of the palindrome (but also might be not). So, we make a guess and allow a multiple transition:

$$\delta(q_1, 0, A) = \{(q_1, AA), (q_2, \varepsilon)\}, \quad \delta(q_1, 1, B) = \{(q_1, BB), (q_2, \varepsilon)\}.$$

While in the state q_2, we compare the input with the content of the stack:

$$\delta(q_2, 0, A) = (q_2, \varepsilon) \quad \delta(q_2, 1, B) = (q_2, \varepsilon)$$

Finally, we have the possibility to empty the stack spontaneously if its top symbol is Z_0, because this can happen only in the case if the input word was a palindrome (including the empty input):

$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon) \quad \delta(q_2, \varepsilon, Z_0) = (q_2, \varepsilon)$$

Note that when the stack is empty, a PDA stops and does not accept any more input.

6.2 Instantaneous description

The configuration of a PDA at any given moment can be represented as follows.

Definition 6.6. *An* instantaneous description *or* ID *of a PDA is a triple* (q, w, γ)*, where* $q \in Q$ *is the current state,* $w \in \Sigma^*$ *is the not yet processed suffix of the input word, and* $\gamma \in \Gamma^*$ *is the content of the stack.*

The transition from one configuration to another is denoted by the symbol $\vdash_M$. By definition of the transition function one has

$$(q, aw, Z\alpha) \vdash_M (p, w, \beta\alpha) \Leftrightarrow (p, \beta) \in \delta(q, a, Z),$$

where $a \in \Sigma \cup \{\varepsilon\}$.

We write $I \vdash_M^* J$ if for some $I_1, \ldots, I_n$ one has

$$I \vdash_M I_1 \vdash_M I_2 \vdash_M \cdots \vdash_M I_n \vdash_M J.$$

In these terms, the acceptance criteria by final state and by empty stack are formulated as follows.

Definition 6.7. *For a PDA M the* language accepted by final state *is*

$$L(M) = \{w \mid (q_0, w, Z_0) \overset{*}{\underset{M}{\vdash}} (p, \varepsilon, \gamma) \text{ for some } p \in F \text{ and } \gamma \in \Gamma^*\},$$

and the language accepted by empty stack *is*

$$N(M) = \{w \mid (q_0, w, Z_0) \overset{*}{\underset{M}{\vdash}} (p, \varepsilon, \varepsilon) \text{ for some } p \in Q\}.$$

6.3 Equivalence of acceptance by final state and empty stack

Theorem 6.8. *If $L = L(M)$ for some PDA M, then there is a PDA M' such that $L = N(M')$.*

Idea of the proof. To replace acceptance by final states with acceptance by empty stack, add a new state (erasure state), allow the automaton to go to this state as soon as it enters a final state, and while in the erasure state delete the top stack symbols spontaneously. □

Theorem 6.9. *If $L = N(M)$ for some PDA M, then there is a PDA M' such that $L = L(M')$.*

Idea of the proof. Introduce two new states: a final state q_f and a new initial state q_0', and also a new start symbol X_0. As the first step, the automaton M' puts the old start symbol Z_0 on the top of the new start symbol and goes to the old initial state:

$$\delta(q_0', \varepsilon, X_0) = (q_0, Z_0 X_0).$$

Then one lets the old PDA M do its job. If one sees X_0 on the top of the stack, then it means that M has emptied its stack. One then goes to the final state:

$$\delta(q, \varepsilon, X_0) = (q_f, \varepsilon) \text{ for all } q \neq q_0'.$$

□

6.4 Equivalence of PDA's and CFL's

In this section we will show that the languages accepted by PDAs are exactly those generated by context-free grammars. This can be compared to the fact that the languages accepted by DFAs are those described by regular expressions.

Theorem 6.10. *For every context-free language L there is a PDA M such that $N(M) = L$.*

The proof uses the notion of a *leftmost derivation* of a word in a context-free grammar. A leftmost derivation is characterized by the property that at

each step a production rule is applied to the leftmost variable in the current word. In other words, the derivation

$$S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_m \Rightarrow w$$

is leftmost if, when we represent α_i as

$$\alpha_i = u_i A_i \gamma_i, \quad u_i \in T^*,\ A_i \in V, \gamma_i \in (T \cup V)^*,$$

then α_{i+1} comes from a production $A_i \to \beta_i$:

$$\alpha_i = u_i A_i \gamma_i \Rightarrow u_i \beta_i \gamma_i = u_{i+1}\gamma_{i+1} = \alpha_{i+1}. \tag{4}$$

(This means in particular that u_i is a prefix of u_{i+1}.)

A leftmost derivation always exists: it can be obtained from a derivation tree by the depth-first traversal.

Proof of Theorem 6.10. Let $G = (V, T, P, S)$ be a context-free grammar such that $L = L(G)$. We construct a PDA that simulates leftmost derivations of words in G and accepts them by the empty stack.

The PDA will have the following structure:

$$M = (\{q\}, T, T \cup V, \delta, q, S, \varnothing).$$

(It has only one state, thus everything is about changing the stack content. The transition rules are as follows:

$$\delta(q, \varepsilon, A) = \{(q, \beta) \mid (A \to \beta) \in P\}, \quad \delta(q, a, a) = (q, \varepsilon).$$

In other words,

$$(q, w, A\gamma) \vdash_M (q, w, \beta\gamma) \text{ whenever } (A \to \beta) \in P \tag{5}$$

$$(q, aw, a\beta) \vdash_M (q, w, \beta) \tag{6}$$

Let us show that M accepts all words generated by G. From the definition of a leftmost derivation of w it is clear that each u_i is a prefix of w: $w = u_i v_i$. The transition rules allow to transform the ID $(q, v_i, A_i\gamma_i)$ to $(q, v_{i+1}, A_{i+1}\gamma_{i+1})$:

$$(q, v_i, A_i\gamma_i) \vdash_M (q, v_i, \beta_i\gamma_i) \vdash_M^* (q, v_{i+1}, A_{i+1}\gamma_{i+1}).$$

First, the rule (5) is applied, and then a sequence (maybe empty) of rules (6). Going over i from 0 to m one transforms (q, w, S) to $(q, \varepsilon, \varepsilon)$.

Conversely, emptying the stack with the help of the rules (5) and (6) can be interpreted as a derivation of a word in the grammar G. □

Theorem 6.11. *For every PDA M the language $N(M)$ is context-free.*

We do not give a proof, but it is a sort of reversal of the above construction.

7 Properties of context-free languages

7.1 Closure properties of CFLs

Theorem 7.1. *Context-free languages are closed under union, concatenation and Kleene closure.*

Proof. Let L be a context-free language, and let $G = (V, T, P, S)$ be a grammar that generates L. We construct a grammar G' by adding to V a new symbol S' (which will be the new start symbol) and a new production rule:

$$V' = V \cup \{S'\}, \quad P' = P \cup \{S' \to SS' \mid \varepsilon\}.$$

In G' one can derive all words of the Kleene closure L^*:

$$S' \underset{G'}{\Longrightarrow} SS' \underset{G'}{\Longrightarrow} SSS' \underset{G'}{\Longrightarrow} \cdots \underset{G'}{\Longrightarrow} SS \dots S \underset{G'}{\Longrightarrow} w_1 w_2 \dots w_n.$$

Vice versa, every word derived from S' belongs to L^*.

Let now L_1 and L_2 be languages generated by context-free grammars

$$G_1 = (V_1, T_1, P_1, S_1), \quad G_2 = (V_2, T_2, P_2, S_2).$$

Put $T = T_1 \cup T_2$. We want to show that the languages $L_1 \cup L_2 \subset T^*$ and $L_1 L_2 \subset T^*$ are context-free. In both cases rename the variables so that $V_1 \cap V_2 = \varnothing$ and construct a new grammar $G' = (V', T, P', S')$ with $V' = V_1 \cup V_2 \cup \{S'\}$. It is easy to see that the production rules

$$P' = P_1 \cup P_2 \cup \{S' \to S_1 \mid S_2\}$$

generate the language $L_1 \cup L_2$, and the production rules

$$P' = P_1 \cup P_2 \cup \{S'' \to S_1 S_2\}.$$

generate $L_1 L_2$. □

Corollary 7.2. *Every regular language is context-free.*

First proof. A regular language can be constructed from the basic languages $\varnothing$, $\{\varepsilon\}$, $\{a\}$ by operations of union, concatenation and Kleene closure. Therefore it suffices to show that the basic languages are context-free. Each of them is generated by the grammars with a single variable S and the following production sets:

$P = \varnothing$ for $L = \varnothing$, $\quad P = \{S \to \varepsilon\}$ for $L = \{\varepsilon\}$, $\quad P = \{S \to a\}$ for $L = \{a\}$.

□

Second proof. A regular language is accepted by some DFA. Every DFA can be viewed as a PDA with stack playing no role and word acceptance by final states. □

7.2 The pumping lemma for CFLs

For any finite alphabet Σ the set of all words Σ^* is countably infinite. The set of all languages over Σ is the set of all subsets of Σ^* and therefore uncountably infinite. On the other hand, the set of all pushdown automata with the input language Σ is countably infinite (as is the set of all context-free grammars). It follows that there are languages which are not context-free.

The following theorem provides a tool to prove non-context-freeness of some languages.

Theorem 7.3. *Let L be a context-free language. Then there is a positive integer n such that every $z \in L$ of length $|z| \geq n$ can be split into subwords $z = uvwxy$ in such a way that*

1. *$|vx| \geq 1$, that is the words v and x are not both empty;*
2. *$|vwx| \leq n$;*
3. *for all $i \geq 0$ we have $uv^i wx^i y \in L$.*

That is to say, every sufficiently long word contains two subwords within a bounded region (the bound n depends on the language, but not on the choice of a word from the language) that can be removed or repeated several times.

Example 7.4. Let us prove that the language $L = \{a^i b^i c^i \mid i \geq 1\}$ is not context-free. Assume it is, and let n be a constant from the pumping lemma corresponding to this language. Take $z = a^n b^n c^n$ and write it as $z = uvwxy$ so that the properties of the pumping lemma are satisfied. We claim that $z' = uwy = uv^0 wx^0 y \notin L$. Indeed, since $|vwx| \leq n$, the subword vwx is eiter contained in the prefix $a^n b^n$ or in the suffix $b^n c^n$ of z. In the first case the word z' has n letters c at the end, but less than $2n$ a's and b's in total, so that $z' \notin L$. Similarly, in the second case z' has n a's, but not enough b's and c's.

Recall that the language $\{a^i b^i \mid i \geq 1\}$ is context-free (Example 5.4, slightly modified to exclude the empty word). The language from the previous example does not look more complicated, however it is no more context-free.

Similarly, we know from Example 6.5 that the set of even-length palindromes $\{w\bar{w} \mid w \in \{0,1\}^*\}$ is a context-free language. The language of the next example does not look much different, but it is not context-free. (Althoug it could be accepted by an automaton with a queue memory.)

Example 7.5. The language $L = \{ww \mid w \in \{0,1\}^*\}$ is not context-free. Assume it is, and let n be a pumping bound for this language. Take the

word $z = 0^n1^n0^n1^n \in L$ and write it as $z = uvwxy$ according to the pumping lemma. By the lemma, the word $z' = uwy$ is in L, we will however show that this is impossible.

The subword vwx is contained within some two consecutive blocks (inside 0^n1^n or 1^n0^n). If vwx is contained in the first half, then while transforming z to z' we removed some symbols from the first half. It follows that $uwy = 0^k1^l0^n1^n$, where $k, l \leq n$ and at least one of them is $< n$. This word does not belong to L. Similarly, for the other situations of vwx the depumped word has the form $0^n1^k0^l1^n$ or $0^n1^n0^k1^l$ and does not belong to L either.

Recall that a derivation of a word in a context-free grammar can be represented in the form of a derivation tree, see Figure 17. (Sometimes different trees can derive the same word, but this is not a problem.) If the grammar has Chomsky form, then its derivation trees are binary trees. More exactly, a derivation tree of a Chomsky form grammar is a full binary tree (variable nodes) together with an additional edge hanging down from every leaf (terminal nodes).

Lemma 7.6. *Let G be a grammar in Chomsky form, and T be a parse tree for a word $z \in G$. If T contains no path of length greater than i, then $|z| \leq 2^{i-1}$.*

Proof. The length of z is the number of leaves of the derivation tree. Deleting the terminal nodes does not change the number of leaves but decreases the depth of the tree by one. A full binary tree of depth $i-1$ has at most 2^{i-1} leaves. □

Proof of Theorem 7.3. Let G be a grammar in Chomsky form generating the language $L \setminus \{\varepsilon\}$. Put $n = 2^{k-1} + 1$, where k is the number of variables in G. We will show that this n satisfies the conditions of the pumping lemma.

Let $z \in L(G)$ be such that $|z| \geq n$, and let T be any derivation tree for z. By Lemma 7.6, T contains a path of length at least $k+1$. This path contains at least $k+2$ vertices. The last vertex of the path is labeled by a terminal, all of the other vertices are labeled by variables. Hence there is a variable that appears on this path at least twice.

Take any of the longest paths in T and, ascending it from the leaf, find the first repetition of variables. This gives us two vertices v_1 and v_2 such that

1. v_1 and v_2 have the same label, say A;
2. v_1 is closer to the root than v_2;
3. the portion of the path from v_1 to the leaf has length at most $k+1$.

The subtree T_1 with the root v_1 represents a derivation from A of a word z' which is a subword of z and has length at most 2^k. Indeed, the portion

of our path from v_1 down to the leaf has length at most $k+1$, and there is no longer path starting from v_1 because we have chosen a longest path.

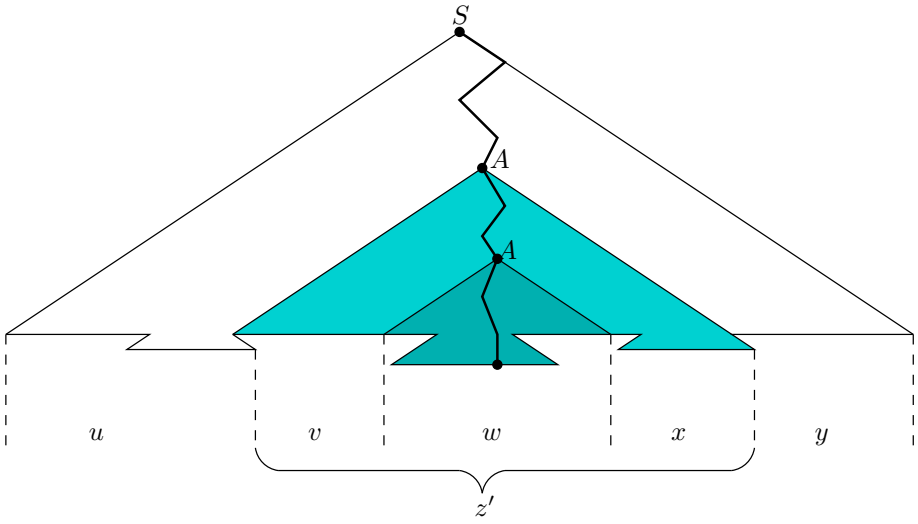

Figure 18: Proof of the pumping lemma for CFLs: structuring a word.

The tree T_1 contains a subtree T_2 growing from v_2. This subtree derives from A a word w which is a subword of z':

$$z' = vwx,$$

see Figure 18. The words v and x cannot both be ε, because v_1 has two children, only one of which is an ancestor of v_2, so that the other child generates a subword of z' disjoint from w.

To summarize, we have

$$S \stackrel{*}{\Rightarrow} uAy, \quad \stackrel{*}{\Rightarrow} vAx, \quad A \stackrel{*}{\Rightarrow} w.$$

It follows that $A \stackrel{*}{\Rightarrow} v^i w x^i$ and $S \stackrel{*}{\Rightarrow} uv^i w x^i y$ for all $i \geq 0$. A derivation tree for $uv^i w x^i y$ can be obtained from the tree T by cut, copy, and paste, see Figure 19. □

7.3 Non-closure properties of CFLs

We have already proved that context-free languages are closed under union, concatenation and Kleene closure.

Theorem 7.7. *The set of context-free languages is not closed under intersection.*

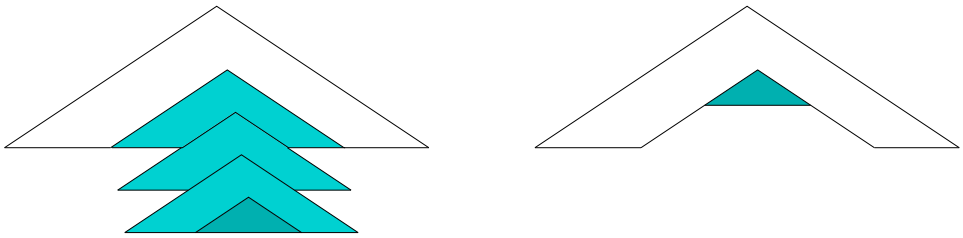

Figure 19: Proof of the pumping lemma for CFLs: pumping a word. These trees represent the words uv^3wx^3y and uwy respectively.

Proof. Consider the languages

$$L = \{a^i b^i c^i \mid i \geq 1\},$$
$$L_1 = \{a^i b^i c^j \mid i, j \geq 1\},$$
$$L_2 = \{a^i b^j c^j \mid i, j \geq 1\}.$$

We have $L = L_1 \cap L_2$. As we know, L is not context-free. On the other hand L_1 is generated by the grammar

$$S \to AB, \quad A \to aAb \mid ab, \quad B \to cB \mid c.$$

The language L_2 is generated by a similar grammar. Thus L_1 and L_2 are context-free languages whose intersection is not context-free. □

Theorem 7.8. *The set of context-free languages is not closed under complementation.*

Proof. We know that CFL's are closed under union. If they would be closed under complementation, then due to

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

they would also be closed under intersection, which is not the case. □

The above argument is non-constructive. Applied to the languages L_1, L_2, L from the proof of Theorem 7.7 it says that at least one of the following is true:

1. L_1 is context-free, but $\overline{L_1}$ is not;
2. L_2 is context-free, but $\overline{L_2}$ is not;
3. $\overline{L_1} \cup \overline{L_2}$ is context free, but its complement is not.

Remark 7.9. The situation is similar to the following proof of the existence of two irrational numbers α and β such that α^β is rational. If $\sqrt{2}^{\sqrt{2}}$ is rational, then one may take $\alpha = \beta = \sqrt{2}$. If $\sqrt{2}^{\sqrt{2}}$ is irrational, then one may take $\alpha = \sqrt{2}^{\sqrt{2}}$, $\beta = \sqrt{2}$ because of

$$\left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2.$$

(In fact, $\sqrt{2}^{\sqrt{2}}$ is rational, which follows from a more general (and more complicated) theorem by Gelfond and Schneider from 1934.)

It is possible to give a concrete example of a context-free language with a non-context-free complement. Namely, according to Example 7.5 the language $\{x \in \{0,1\}^* \mid x = ww \text{ for some } w \in \{0,1\}^*\}$ is not context-free. Its complement is context-free (exercise).

8 Turing machines

8.1 Definition

Similarly to all automata we considered before, a Turing machine has a finite number of states and changes its states according to the input. The input is a finite word on an infinite tape, and there are two new aspects in how the machine interacts with the input:

- the machine can move along the tape;
- the machine can write on the tape.

Any sort of input: a number, a list, a table, a combinatorial structure such as a graph, can be encoded as a sequence of symbols on a tape (one just needs to choose an encoding convention). For any algorithm: multiplication of integers, search for a path between two vertices in a graph, there is a Turing machine which applies this algorithm to any given input. The last sentence is, in fact, a definition of the algorithm and a form of the Church thesis: everything which "can be computed" can be done with some Turing machine.

Definition 8.1. *A* Turing machine *is*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

- *Q is the set of states, $q_0 \in Q$ is the initial state, and $F \subset Q$ is the set of final states;*

- Γ *is the set of* tape symbols, $\Sigma \subset \Gamma$ *is the set of* input symbols, *and* $B \in \Gamma \setminus \Sigma$ *is the* blank *symbol;*
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *is a partial map (that is, it can be undefined for some arguments), called the* move function.

At each moment of time, the machine is situated opposite to some cell of the input tape. The input symbol X and the current state p of the machine determine its move $\delta(p, X) = (q, Y, D)$ which consists in:

- changing the state to q,
- replacing the symbol X in the current cell by Y,
- and moving in the direction D (one cell to the left if $D = L$ or one cell to the right if $D = R$).

See Figure 20.

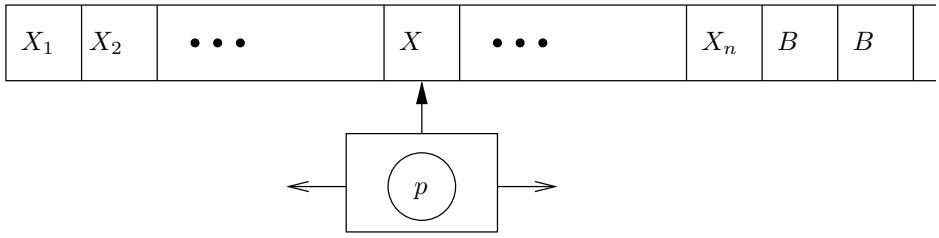

Figure 20: A Turing machine.

At the beginning, the machine is placed at the leftmost cell of the tape and is in the state q_0. The *language accepted by* M is the set of all input words for which M enters a final state at some moment of time. After entering a final state, the machine *halts*, that is $q(f, X)$ is undefined for all $f \in F$. If the input word is not accepted, then the machine either halts in a non-final state or runs forever (in an infinite loop or by increasing the data volume on the tape to infinity).

Example 8.2. Let us design a machine accepting the language $\{0^n 1^n \mid n \geq 1\}$. The algorithm is as follows: check the first zero, move right until meet the first one and check it, move left to the first unchecked one and check it, move right... etc. The machine will have several states according to the tasks: for example, a state for moving right until find the first (unchecked) one. Checking the symbols will be done by replacing them: 0 with X, and 1 with Y. By carefully working out the details one may arrive at a machine with

$$Q = \{q_0, q_1, q_2, q_3, q_4\}, \quad \Gamma = \{0, 1, X, Y, B\}, \quad F = \{q_4\}$$

and the move function described by the following table.

	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

In a similar way one can construct a machine accepting the language $\{0^n 1^n 2^n \mid n \geq 1\}$ (recall that this language is not context-free).

Definition 8.3. *A language is called* recursively enumerable *if it is accepted by some Turing machine. A language is called* recursive *if it is accepted by a Turing machine which halts on every input.*

All context-free languages are recursively enumerable because every push-down automaton can be simulated by a Turing machine. One can show that context-free languages are also recursive.

If a language is recursively enumerable but not recursive, then every Turing machine accepting it will run forever on some input word not belonging to the language. Later we will give an example of such a language.

Lemma 8.4. *A language L is recursive if and only if both L and its complement are recursively enumerable.*

Proof. A recursive language is recursively enumerable by definition. Its complement is also recursively enumerable: take the same machine and replace F by $Q \setminus F$.

If L and $\Sigma^* \setminus L$ are recursively enumerable, then one lets the corresponding machine run on the same input word w. At least one of the machines will halt and tell us whether $w \in L$ or not. □

We do not go into details how one can combine two Turing machines into one, but this can be done with the techniques similar to those we used to design new finite automata.

8.2 Modifications of Turing machines

There are several variations of Turing machines, e. g.:

- multi-tape: several tapes, each with its own head for reading and writing;
- multidimensional: a square grid instead of a tape, the head can move not only left and right, but also up and down;
- non-deterministic.

Although they seem more powerful, each of them can be simulated by an ordinary Turing machine.

8.3 Problems and languages

Consider a problem like "Is a given finite graph connected?" An *instance* of a problem is any finite graph. Since graphs can be encoded as words in an alphabet, this problem determines a language: the set of all words encoding connected graphs. This holds for any question which depends on some countable parameter and has a yes/no answer: encode the parameter values by words, and consider the set of all words for which the answer to the question is "yes".

Definition 8.5. *A problem is called* decidable *if the corresponding language is recursive, that is if there is a Turing machine which halts on every input and accepts exactly those words which correspond to problem instances with the positive answer.*

A problem is called semi-decidable *if the corresponding language is recursively enumerable.*

8.4 The universal language

Definition 8.6. *The* universal language *is the set*

$$L_u = \{(M, w) \mid \textit{Turing machine } M \textit{ accepts word } w\}.$$

A Turing machine can be encoded with a binary word (once an encoding convention is chosen). Similarly, words in all finite alphabets can be encoded by binary words (once a binary encoding of the alphabet symbols is chosen). Therefore the universal language can be viewed as a language over a binary alphabet.

Lemma 8.7. *The universal language is recursively enumerable.*

Proof. Receiving (M, w) as input, let M run on w. As soon as M halts accepting w, accept the pair (M, w).

A Turing machine which implements the above algorithm can be described in a multi-track architecture: the first tape holds the code of M, the second tape holds the word w, the third tape is used to store the state of M, the fourth tape stores the position in the tape of M. □

8.5 Undecidability of the halting problem

The *halting problem* is "Does Turing machine M accept input w?" The corresponding language is the universal language L_u.

Theorem 8.8. *The halting problem is undecidable, that is, the universal language is not recursive.*

Construct a language L_d:

$$L_d = \{w_i \mid w_i \text{ is not accepted by } M_i\}.$$

Lemma 8.9. *The language L_d is not recursively enumerable.*

Proof. Assume the contrary. Then there is a Turing machine M_j which accepts the language L_d. By inspection of the word w_j we arrive to a contradiction:

- if $w_j \in L_d$, then by definition of L_d the word w_j is not accepted by M_j, which by the choice of M_j means that $w_j \notin L_d$;
- if $w_i \notin L_d$, then by definition of L_d the word w_j is accepted by M_j, which by the choice of M_j means that $w_j \in L_d$.

□

Proof of Theorem 8.8. If L_u is recursive, then there is a Turing machine A which always halts and accepts only pairs (M, w) from L_u. Let us show that then L_d is recursive, which contradicts Lemma 8.9. Given a word w determine the integer i such that $w_i = w$. Then determine the machine M_i. Feed (M_i, w_i) into A and accept w if and only if A *does not* accept (M_i, w_i). This gives an algorithm which always stops and recognizes the language L_d. □

Remark 8.10. The above argument together with Lemma 8.7 shows that the complement of L_d is recursively enumerable (but not recursive).

Bibliography

[1] Martin Aigner. *A course in enumeration*, volume 238 of *Graduate Texts in Mathematics*. Springer, Berlin, 2007.

[2] George E. Andrews and Kimmo Eriksson. *Integer partitions*. Cambridge University Press, Cambridge, 2004.

[3] J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008.

[4] René Cori and Daniel Lascar. *Mathematical logic*. Oxford University Press, Oxford, 2000. A course with exercises. Part I, Propositional calculus, Boolean algebras, predicate calculus.

[5] René Cori and Daniel Lascar. *Mathematical logic*. Oxford University Press, Oxford, 2001. A course with exercises. Part II, Recursion theory, Gödel's theorems, set theory, model theory.

[6] Jean H. Gallier. *Logic for computer science*. Dover, 2015.

[7] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Co., Reading, Mass., 1979. Addison-Wesley Series in Computer Science.

[8] Stephen Cole Kleene. *Mathematical logic*. Dover Publications, Inc., Mineola, NY, 2002.

[9] Dirk van Dalen. *Logic and structure*. Universitext. Springer, London, fifth edition, 2013.