Formal Logic

Winter term 2025/26

Dr Dirk Frettlöh
Technische Fakultät
Universität Bielefeld

October 22, 2025

Contents

1 Propositional Logic **3**

1.1 Basics . . . 3
1.2 Calculation rules . . . 6
1.3 Normal forms . . . 7
1.4 Horn formulas . . . 12
1.5 Compactness Theorem . . . 12
1.6 Consequences I . . . 13
1.7 Resolution . . . 15
1.8 Tableau calculus . . . 19
Interlude: Relations . . . 21

2 First-order logic **23**

2.1 Syntax of first-order logic . . . 23
2.2 Semantics of first-order logic . . . 24
2.3 Normal forms . . . 27
2.4 Resolution calculus of first-order logic . . . 30

3 Modal logic **33**

3.1 Syntax and semantics . . . 34
3.2 Calculation rules and (no) normal forms . . . 37
3.3 Tableau calculus for modal logic . . . 39
3.4 Different flavours of modal logic . . . 42
Interlude: Infinite cardinalities . . . 43

4 Undecidability **46**

4.1 Undecidable problems . . . 46
4.2 Computable numbers . . . 49
4.3 Consequences II . . . 50
4.4 Gödel's completeness theorem . . . 51
4.5 Gödel's incompleteness theorems . . . 51

5 Zermelo-Fraenkel, axiom of choice and the Banach-Tarski paradox **53**

5.1 Zermelo-Fraenkel axioms . . . 54
5.2 Axiom of choice . . . 56

1 Propositional Logic

1.1 Basics

14. Oct.

Logic is about truth or falseness of statements. Consider some examples:

1. The university building is beautiful
2. $2+2=5$
3. X is chinese, therefore X is a human
4. X is a human, therefore X is a chinese
5. Please do not smoke
6. Hello Kitty

Items number 5 and 6 are not statements, since they are neither "true" nor "false". Such examples are out of our scope.

Statements 1-4 *are* statements, as they are either true or false. (Ok, one could argue about 1, but ...) Some of the statements 1-4 are combinations of smaller statements, while others are not. Statements 1 and 2 are not combinations of smaller statements: each smaller part (such as $2+2$ or "The university building") is neither true nor false and therefore not a statement. Statements 3 and 4 are combinations of smaller statements. Both are of the form "A, therefore B," meaning something like "A implies B" or "A $\Rightarrow$ B".

In the rest of this section, we want to make this more precise by abstracting it (i.e., separating it from the linguistic content) and formalizing it.

Syntax

A statement that cannot be divided into smaller pieces is called **atomic formula**. More complex statements are built from atomic formulas as follows:

Definition 1.1. (and notation) Atomic formulas are abbreviated by $A, B, C, \ldots$, or by $A_1, A_2, \ldots$.

A **formula** (in propositional logic) is defined inductively as follows:

- Each atomic formula is a formula.
- If F and G are formulas, then $F \vee G$, $F \wedge G$ and $\neg F$ are formulas.

If H is a formular, then every F and G from above, from which H is built, are called *partial formulas* of H.

Everything that can be built in this way is a formula (and nothing else). (Specifically: a propositional logic formula, but we will not mention the adjective as long as it does not lead to confusion; i.e. in this whole chapter. Later we will consider formulas of other logics as well.)

Example 1.2. $F = \neg((A \wedge B) \vee C)$ is a formula. Its *partial formulas* are $A, B, C, A \wedge B, (A \wedge B) \vee C$, and $\neg((A \wedge B) \vee C)$. But $(A\wedge$, or $\vee C$, or (are not partial formulas of F, since they are not formulas.

Notation

- $A, B, C, \ldots$, or $A_1, A_2, \ldots$ denote atomic formulas.
- $F, G, H, \ldots$, or $F_1, F_2, \ldots$ denote formulas.
- $F \Rightarrow G$ is short for $(\neg F) \vee G$. $F \Leftrightarrow G$ is short for $(F \wedge G) \vee (\neg F \wedge \neg G)$.

Later on we will interpret $\wedge$ as "and" etc. Up to here this is entirely abstract, the symbols are just symbols (**syntax**: symbols and rules). We want to give them some meaning (**semantics**):

Semantics

Definition 1.3. The elements of the set $\{0,1\}$ are **truth values**. (Intuition: 0 = false, 1 = true). A **valuation** of a set $M = \{A, B, C, \ldots\}$ of atomic formulas is a mapping

$$\mathcal{A} : \{A, B, C, \ldots\} \to \{0,1\}$$

$\mathcal{A}$ is extended to all formulas by

1. $\mathcal{A}(F \wedge G) = \min\{\mathcal{A}(F), \mathcal{A}(G)\} = \begin{cases} 1 & \text{if } \mathcal{A}(F) = \mathcal{A}(G) = 1 \\ 0 & \text{else} \end{cases}$
2. $\mathcal{A}(F \vee G) = \max\{\mathcal{A}(F), \mathcal{A}(G)\} = \begin{cases} 0 & \text{if } \mathcal{A}(F) = \mathcal{A}(G) = 0 \\ 1 & \text{else} \end{cases}$
3. $\mathcal{A}(\neg F) = 1 - \mathcal{A}(F) = \begin{cases} 0 & \text{if } \mathcal{A}(F) = 1 \\ 1 & \text{else} \end{cases}$

Example 1.4. Since we consider only finitely many atomic formulas, a valuation can be viewed as a finite list. For instance, let $\mathcal{A}(A) = 1, \mathcal{A}(B) = 1, \mathcal{A}(C) = 0$. Shortly we can write this $\mathcal{A}$ as

A	B	C
1	1	0

Now, for this particular $\mathcal{A}$, and and for $F = \neg((A \wedge B) \vee C)$, what is the value of $\mathcal{A}(\mathcal{F})$?

$$\mathcal{A}(\neg((A \wedge B) \vee C)) = 1 - \mathcal{A}((A \wedge B) \vee C) = 1 - \begin{cases} 0 & \text{if } \mathcal{A}(A \wedge B) = \mathcal{A}(C) = 0 \\ 1 & \text{else} \end{cases}$$

$$= 1 - \begin{cases} 0 & \text{if } \mathcal{A}(A \wedge B) = 0 \\ 1 & \text{else} \end{cases} \qquad = 1 - 1 = 0 \quad (\text{since } \mathcal{A}(A) = \mathcal{A}(B) = 1)$$

Of course this method of evaluation the value of F is tedious. A more efficient method is to use **truth tables**. A formula with n atomic formulas has 2^n possible distinct valuations $\mathcal{A}$. Hence we might define $\neg, \vee$ and $\wedge$ by truth tables, and we may construct the truth tables also for $\Rightarrow$ and $\Leftrightarrow$ by considering all possibilities:

$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}(\neg(F))$	$\mathcal{A}(F \wedge G)$	$\mathcal{A}(F \vee G)$	$\mathcal{A}(F \Rightarrow G)$ $= \mathcal{A}(\neg F \vee G)$	$\mathcal{A}(F \Leftrightarrow G)$ $= \mathcal{A}((F \wedge G) \vee (\neg F \wedge \neg G))$
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	0	0
1	1		1	1	1	1

Note that each row in the truth table corresponds to one valuation $\mathcal{A}$. We can now also determine $\mathcal{A}(F)$ from Example 1.4 by setting up the truth value table for F and reading the $\mathcal{A}(F)$ in the correct line (namely 1 1 0). This is not difficult (exercise).

Common interpretations of these rules are that $\vee$ means "or," $\wedge$ means "and," and $\neg$ means "not." For "and" and "not," this is indeed very appropriate. For "or," however, one could argue. If someone says they will bring pasta salad or meatballs, one does not expect them to bring both, but rather only one of the two. Similarly, if an examiner asks in an exam whether answer A is correct or answer B is correct, the response "Yes" will not be well received, even though it is completely correct with respect to the logical "or" (assuming that at least one of the two answers A or B is correct).

However, keep in mind this is just an interpretation. Since we want to formalize here, we do not concern ourselves with interpretation.

Furthermore, there is another operator that covers the missing meaning: the "either-or," also known as "exclusive or," briefly written as $\oplus$: $\mathcal{A}(A \oplus B) = 1$ if and only if $\mathcal{A}(A) \neq \mathcal{A}(B)$. It is easy to see that $A \oplus B$ is the same as $\neg(A \Leftrightarrow B)$. (Exercise: how?)

A common interpretation of $\Rightarrow$ is implication: "therefore" or "A implies B." A common interpretation of $\Leftrightarrow$ is equivalence: "if and only if." This is how these symbols are used in mathematics. But beware: we must be careful not to mix up the symbols $\Rightarrow$ and $\Leftrightarrow$ with their meta-meaning. The levels would be mixed if we wrote $\mathcal{A}(A \Rightarrow B) = 0 \Rightarrow \mathcal{A}(A \Leftrightarrow B) = 0$. We will therefore carefully distinguish this and, for example, instead write "$\mathcal{A}(A \Rightarrow B) = 0$, so $\mathcal{A}(A \Leftrightarrow B) = 0$ holds."

Remark 1.5. To save parentheses, we introduce a hierarchy among the operators (analogous to the order of evaluationg operations in arithmetic, i.e., "$\cdot$ before $+$ and $-$"):

$$\neg \text{ before } \wedge \text{ , } \vee \text{ before } \Rightarrow \text{ before } \Leftrightarrow.$$

Hence we may write for instance $\neg A \vee B \Rightarrow C$ rather than $((\neg A) \vee B) \Rightarrow C$.

The **main questions** we will address in the following are:

Which formulas have the same meaning? ((semantic) **equivalence**)

Given a formula F, is there an valuation that makes it true? (**satisfiability**)

Given two formulas F, G, is G a logical consequence of F? That is, whenever F is true, is G also true? ((semantic) **consequence**)

In addition to the precise formal structure, the focus of this lecture is to provide algorithmic tools to answer these questions (efficiently, if possible). Here, "algorithmic" means learning and understanding the fundamental (meta-)algorithms. We will not implement anything or solve problems using SAT solvers.

For the questions above, we now define the necessary terms.

Remark 1.6. When we write expressions like $\mathcal{A}(F)$ or $\mathcal{A}(F) = \mathcal{A}(G)$ in the following, we always assume that $\mathcal{A}$ is defined for all atomic formulas in F — or in F and G, etc. This saves us an additional technical definition.

Definition 1.7. A valuation $\mathcal{A}$ **satisfies** a formula F if $\mathcal{A}(F) = 1$. Notation: $\mathcal{A} \models F$.

Otherwise ($\mathcal{A}(F) = 0$) we say $\mathcal{A}$ does not satisfy F. Notation: $\mathcal{A} \not\models F$.

F is **satisfiable** if there is a valuation $\mathcal{A}$ such that $\mathcal{A}(F) = 1$. Otherwise F is unsatisfiable.

F is a **tautology** if for all valuations $\mathcal{A}$ holds: $\mathcal{A}(F) = 1$. In this case we write shortly $\models F$.

F is (semantically) **equivalent** to G if for all valuations $\mathcal{A}$, it holds that $\mathcal{A}(F) = \mathcal{A}(G)$. Notation: $F \equiv G$.

G is a **semantic consequence** of F if for all valuations $\mathcal{A}$ with $\mathcal{A}(F) = 1$, it follows that $\mathcal{A}(G) = 1$ as well. Notation: $F \models G$.

Example 1.8. $A \vee \neg A$ and $\neg A \Rightarrow (A \Rightarrow B)$ are tautologies. $A \wedge \neg A$ is unsatisfiable (truth tables!)

Remark 1.9. F and G may contain different atomic formulas and nevertheless $F \equiv G$ may hold (for instance, if both formulas are tautologies, like $F = A \vee \neg A$ and $G = \neg(B \wedge \neg B)$).

1.2 Calculation rules

21. Oct.

Rules like $F \wedge G \equiv G \wedge F$ are obvious (are they?), but there are more sophisticated ones:

Theorem 1.10. *Let F, G, H are formulas. Then the following equivalences hold:*

$F \wedge F \equiv F$, $F \vee F \equiv F$	*(Idempotence)*
$F \wedge G \equiv G \wedge F$, $F \vee G \equiv G \vee F$	*(Commutativity)*
$(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$, $(F \vee G) \vee H \equiv F \vee (G \vee H)$	*(Associativity)*
$F \wedge (F \vee G) \equiv F$, $F \vee (F \wedge G) \equiv F$	*(Absorption)*
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$, $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$	*(Distributivity)*
$\neg\neg F \equiv F$	*(Double Negation)*
$\neg(F \wedge G) \equiv \neg F \vee \neg G$, $\neg(F \vee G) \equiv \neg F \wedge \neg G$	*(de Morgan's laws)*
If F is a tautology, then $F \vee G \equiv F$, $F \wedge G \equiv G$	
If F is unsatisfiable, then $F \vee G \equiv G$, $F \wedge G \equiv F$	

All identities can be proven by truth tables, compare Exercise Sheet 2.

Because of the associativity law we may write $F \wedge G \wedge H$ rather than $(F \wedge G) \wedge H$ or $F \wedge (G \wedge H)$.

Example 1.11. Of course, we can use truth tables to show the equivalence of any formulas. Since we now have the rules above at our disposal (let's assume we have proven them all), we can also demonstrate the equivalence of two formulas, e.g., $F = (A \vee (B \vee C)) \wedge (C \vee \neg A)$ and $G = (\neg A \wedge B) \vee C$, using the calculation rules above:

$$(A \vee (B \vee C)) \wedge (C \vee \neg A) \overset{(Ass.,Comm.)}{\equiv} (C \vee (A \vee B)) \wedge (C \vee \neg A) \overset{(Distr.)}{\equiv} C \vee ((A \vee B) \wedge \neg A)$$
$$\overset{(Comm.,Distr)}{\equiv} C \vee ((\neg A \wedge A) \vee (\neg A \wedge B)) \overset{(unsat.)}{\equiv} C \vee (\neg A \wedge B) \overset{(Comm.)}{\equiv} (\neg A \wedge B) \vee C.$$

If one is very precise, one could object that, for example, we know that for the subformulas above, it holds that $A \vee (B \vee C) \equiv C \vee (A \vee B)$. But why does this imply that the complete formulas (which contain these subformulas) are also equivalent? In fact, this must be proven.

Theorem 1.12 (Replacement theorem)**.** *Let F, G be formulas with $F \equiv G$. Let F be a partial formula of H. Let H' be the formula arising from H by replacing F with G. Then $H \equiv H'$.*

Proof. By induction (*structural induction*, on the inductive construction of a formula)

Base of induction: Let H be an atomic formula. Then H has only one subformula, namely H itself: So $H = F$, hence $H' = G$, thus $H' = G \equiv F = H$.

Inductive Step: Let F be a formula. Assume that the claim holds for all formulas smaller than F (in particular, for all subformulas of F except F itself).

Case 0: If $H = F$, then the result follows exactly as in the base case.

Case 1: If $H = \neg H_1$, then by the induction hypothesis, we have $H_1 \equiv H_1'$, so $H = \neg H_1 \equiv \neg H_1' = H'$.
More precisely: for every $\mathcal{A}$, it holds that $\mathcal{A}(H) = 1 - \mathcal{A}(H_1) = 1 - \mathcal{A}(H_1') = \mathcal{A}(\neg H_1') = \mathcal{A}(H')$.

Case 2: If $H = H_1 \vee H_2$, assume (WLOG[1]) that F is a subformula of H_1. (Otherwise, we rename them, making H_2 into H_1). By the induction hypothesis, $H_1 \equiv H_1'$, so $H = H_1 \vee H_2 \equiv H_1' \vee H_2 = H'$.

Case 3: If $H = H_1 \wedge H_2$, the proof follows completely analogously to Case 2. □

This proof illustrates a general phenomenon: most proofs in formal logic are technical, not elegant, and yield no deeper insight (except proofing formalities). Because of this we will show only the few nice proofs in the sequel, and omit the technical ones.

1.3 Normal forms

Truth tables are an initial algorithmic method to answer the question of whether a formula F is satisfiable (how?), or whether two formulas F and G are equivalent (how?). However, this is certainly not very efficient: if F contains n atomic formulas, then the number of rows in the corresponding truth table is 2^n.

In the following, we will learn three more efficient methods to answer the question "Is F satisfiable?" efficiently — one of which applies only to a specific class of formulas ("Horn formulas").

For the first two methods (the Horn formula algorithm and the resolution calculus), the formulas must be in a specific form, a "normal form." **Notation:** $\bigwedge_{i=1}^{n} F_i = F_1 \wedge F_2 \wedge \cdots \wedge F_n$; $\bigvee_{i=1}^{n} F_i = F_1 \vee F_2 \vee \cdots \vee F_n$.

Definition 1.13. A **literal** is a formula of the form A or $\neg A$, where A is some atomic formula.

A formula F is in **disjunctive normal form (DNF)** if

$$F = \bigvee_{i=1}^{n} \Big(\bigwedge_{j=1}^{m_i} L_{ij} \Big) \quad \text{where } L_{ij} \text{ are literals.}$$

A formula of the form $L_1 \wedge L_1 \wedge \cdots \wedge L_n$ (where L_i are literals) is called *conjunctive clause*.

A formula F is in **conjunctive normal form (CNF)** if

$$F = \bigwedge_{i=1}^{n} \Big(\bigvee_{j=1}^{m_i} L_{ij} \Big) \quad \text{where } L_{ij} \text{ are literals.}$$

A formula of the form $L_1 \vee L_1 \vee \cdots \vee L_n$ (where L_i are literals) is called *disjunctive clause*.

Example 1.14. The following formulas all have DNF:

$$(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F), \quad (A \wedge B) \vee C, \quad A \wedge B, \quad A.$$

[1] "Without loss of generality"

The following formulas do not have DNF:

$$\neg(A \vee B) \quad \text{(since an "or" sits inside a "not")},$$

$$A \vee (B \wedge (C \vee D)) \quad \text{(since an "or" sits inside an "and")}.$$

Theorem 1.15. *Each formula has some equivalent formula in DNF, and also some equivalent formula in CNF.*

The proof of this result is long and technical and uses induction on the construction of the formula. It essentially consists of showing that the following algorithm terminates and yields a CNF for F:

Algorithm 1.16. First convert all partial formulas of F of the form $G \Rightarrow H$ into $\neg G \vee H$, and all partial formulas of the form $G \Leftrightarrow H$ into $(G \wedge H) \vee (\neg G \wedge \neg H)$.

1. Replace in F each
   - $\neg\neg G$ by G
   - $\neg(G \wedge H)$ by $\neg G \vee \neg H$
   - $\neg(G \vee H)$ by $\neg G \wedge \neg H$, as long as possible.
2. Replace in F each
   - $G \vee (H \wedge J)$ by $(G \vee H) \wedge (G \vee J)$
   - $(G \wedge H) \vee J$ by $(G \vee J) \wedge (H \vee J)$, as long as possible.
3. Remove repetitions of clauses, if necessary.

The corresponding algorithm to construct the DNF of a formula F is obtained by replacing step 2. above by

2. Replace in F each

- $G \wedge (H \vee J)$ by $(G \wedge H) \vee (G \wedge J)$
- $(G \vee H) \wedge J$ by $(G \wedge J) \vee (H \wedge J)$, as long as possible. Then...

Example 1.17. Let $F = (A \wedge B \wedge C) \vee (D \wedge E)$. This has DNF. Let us apply the CNF-algorithm to transform F into CNF:

$$\begin{aligned}(A \wedge B \wedge C) \vee (D \wedge E) &= \big((A \wedge B \wedge C) \vee D\big) \wedge \big((A \wedge B \wedge C) \vee E\big)\big) \\ &= (A \vee D) \wedge (B \vee D) \wedge (C \vee D) \wedge (A \vee E) \wedge (B \vee E) \wedge (C \vee E)\end{aligned}$$

Remark 1.18. Here, we have strictly speaking used a generalization of the distributive law. If we calculate a few examples in detail, we quickly see that the distributive law from Theorem 1.10 generalizes to three, four, ... subformulas. (In the example above, we used "three"). The most general form of the distributive laws looks as follows:

$$(\bigvee_{i=1}^{n} F_i) \wedge (\bigvee_{j=1}^{m} G_j) \equiv \bigvee_{i=1}^{n} (\bigvee_{j=1}^{m} (F_i \wedge G_j)) \quad \text{resp.} \quad (\bigwedge_{i=1}^{n} F_i) \vee (\bigwedge_{j=1}^{m} G_j) \equiv \bigwedge_{i=1}^{n} (\bigwedge_{j=1}^{m} (F_i \vee G_j))$$

There is an alternative algorithm to produce CNF or DNF using truth tables. Assume that we would have already constructed the truth table of some formula F, containing atomic formulas $A_1, \ldots, A_n$.

Algorithm 1.19. In order to construct the DNF of F:

- Each row where $\mathcal{A}(F) = 1$ yields a clause $(L_1 \wedge \cdots \wedge L_n)$. If $\mathcal{A}(A_i) = 1$ in this row then set $L_i = A_i$, else set $L_i = \neg A_i$ $(i = 1, \ldots, n)$
- Connect the parts by $\vee$

In order to construct the CNF of F:

- Each row where $\mathcal{A}(F) = 0$ yields a clause $(L_1 \vee \cdots \vee L_n)$. If $\mathcal{A}(A_i) = 0$ in this row then set $L_i = A_i$, else set $L_i = \neg A_i$ $(i = 1, \ldots, n)$
- Connect the clauses by $\wedge$

Example 1.20. Consider $F = \neg(A \Rightarrow B) \vee \neg(A \vee B \vee C)$. The truth table is

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

We read off for the DNF

$$F \equiv (\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C).$$

We read off for the CNF

$$F \equiv (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$$

In general, CNF and DNF of a formula are not unique, there are usually several. The last algorithm provides a kind of standard CNF or standard DNF, in which each clause contains all atomic formulas. Except for the order, this is unique.

Relationship with P vs NP

28. Oct.

Remark 1.21. The question "is a propositional formula F satisfiable" is NP-complete (Cook's Theorem 1971, see also Levin 1973). Cook shows that the question of the satisfiability of a general formula (the problem is called SAT) can be traced back in polynomial time to the satisfiability of a formula in CNF (the problem is then sometimes called CNFSAT). This latter problem can in turn be reduced to the satisfiability of a formula in CNF in which each clause has at most three literals. This problem is famous and has the name 3SAT ("three-satisfiability"). 3SAT is therefore also NP-complete (Karp 1972).

In the next chapter we will get to know a class of formulas (so-called "Horn formulas") for which the question of satisfiability is efficiently decidable. The corresponding problem 2SAT — satisfiability of a formula in CNF, in which each clause has at most two literals — is decidable in polynomial time.

Remark 1.22. The corresponding problem of solving a formula in DNF is decidable in polynomial time. Therefore, we cannot expect that there is a general *and* efficient way to transform an arbitrary formula in CNF into an equivalent formula in DNF (with polynomial cost). In fact, the length of the formula can already grow exponentially in general. This worst case is achieved, for example, by

$$F = \bigwedge_{i=1}^{n} A_i \vee B_i = (A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \cdots \wedge (A_{n-1} \vee B_{n-1}) \wedge (A_n \vee B_n)$$

(this CNF has length $2n$ in order of magnitude). The (!) DNF of F is

$$\begin{array}{r} (A_1 \wedge A_2 \wedge \cdots \wedge A_{n-2} \wedge A_{n-1} \wedge A_n) \\ \vee (A_1 \wedge A_2 \wedge \cdots \wedge A_{n-2} \wedge A_{n-1} \wedge B_n) \\ \vee (A_1 \wedge A_2 \wedge \cdots \wedge A_{n-2} \wedge B_{n-1} \wedge A_n) \\ \quad \vdots \quad \vdots \\ \vee (B_1 \wedge B_2 \wedge \cdots \wedge B_{n-2} \wedge B_{n-1} \wedge A_n) \\ \vee (B_1 \wedge B_2 \wedge \cdots \wedge B_{n-2} \wedge B_{n-1} \wedge B_n) \end{array},$$

with a length of 2^n. (This length explosion also happens the other way around, from DNF to CNF, if we replace all $\vee$ with $\wedge$ and vice versa).

Theorem 1.23 ((3SAT))**.** *For each formula F there is a formula G in CNF where each disjunctive clause has at most three literals, such that F is satisfiable if and only if G is satisfiable.*

Proof. By Theorem 1.15 F has a CNF. We just need to show how to replace each disjunctive clause in the CNF that has more than three literals by one or more disjunctive clauses with at most three literals. The trick is to replace each clause with m literals by a partial formula that is the conjunction of four clauses with $m-1, 3, 2$ and 2 literals; and to continue this until there are no more clauses with $m \geq 4$ literals.

Assume there is a clause H with more than three literals: $H = A \vee B \vee G$. We replace H in the CNF by $H' = (A' \Leftrightarrow A \vee B) \wedge (A' \vee G)$, introducing a new atomic formula A' that does not occur in F. If H is satisfiable, any valuation $\mathcal{A}$ for H with $\mathcal{A}(H) = 1$ can be can be extended to H' by setting $\mathcal{A}(A') := \mathcal{A}(A \vee B)$.

SAT is NP-complete	(Cook 1971, Levin 1973)
CNFSAT is NP-complete	(Cook 1971)
3SAT is NP-complete	(Karp 1972)
2SAT is in P	(exercise)
DNFSAT is in P	(exercise)

Table 1: Overview of the difficulty of the satisfiability problems (for an explanation of explanation of P and NP see "Complexity" on page 58)

Why does this help? We claim that (a) $\mathcal{A}(H) = \mathcal{A}(H')$ and (b) H' can be written as a disjunction clause with $m-1, 3, 2$ and 2 literals.

(a) For the $\mathcal{A}$ above holds

$$\mathcal{A}(H) = \mathcal{A}(A \vee B \vee G) = \mathcal{A}(A' \vee G) = \mathcal{A}\big((A' \Leftrightarrow A \vee B) \wedge (A' \vee G)\big) = \mathcal{A}(H').$$

The second equation holds because $\mathcal{A}(A') := \mathcal{A}(A \vee B)$. (This is *not* true for all $\mathcal{A}$, only for this one). The third equation holds because $\mathcal{A}(A' \Leftrightarrow A \vee B) = 1$ (again, since we have defined $\mathcal{A}(A') = \mathcal{A}(A \vee B)$, compare the autologous rule in Theorem 1.10).

Thus H' is satisfiable if and only if H is.

(b) This is easy:

$$\begin{aligned} A' \Leftrightarrow A \vee B &\equiv (A' \Rightarrow A \vee B) \wedge (A \vee B \Rightarrow A') \equiv (\neg A' \vee A \vee B) \wedge (\neg(A \vee B) \vee A') \\ &\equiv (\neg A' \vee A \vee B) \wedge ((\neg A \wedge \neg B) \vee A') \equiv (\neg A' \vee A \vee B) \wedge (\neg A \vee A') \wedge (\neg B \vee A'), \end{aligned}$$

hence

$$H' \equiv (\neg A' \vee A \vee B) \wedge (\neg A \vee A') \wedge (\neg B \vee A') \wedge (A' \vee G),$$

the new formula has CNF, too. Moreover, the claim on the number of clauses follows. This completes the proof. □

Let us also count how the number of clauses grows during this replacement process: replacing H by H' means replacing a single clause with $m \geq 4$ literals by one clause with $m-1$ literals plus one more clause with three literals plus two clauses with two literals. This can be applied until all clauses have three literals or less. Hence if our general formula F in CNF has n clauses with at most m literals each, our counting shows that we can reduce each clause in $m-3$ steps to several small clauses with at most three literals. How many small clauses? In each step we trade one clause for four clauses, so in each step the number grows by three. So each long clause (of length m) is replaced by at most $3(m-3)+1 = 3m-2$ clauses of length three (or less). Altogether we end up with at most $n(3m-2)$ clauses. This is polynomial in n and m, resp. in $\max\{n, m\}$.

Remark 1.24. The question whether a formula G is satisfiable is NP-complete in general (Cook's Theorem 1971, see also Levin 1973). He showed that the question of the satisfiability of any formula in propositional logic can be reduced in polynomial time to the question about satisfiability of formulas in CNF (sometimes called CNFSAT). We just showed that, the more general problem CNFSAT can be reduced (in polynomial time) to the question whether a given formula in CNF where each clause has at most three literals is satisfiable. This latter problem is called 3SAT ("three-satisfiability"). Hence 3SAT is also NP-complete (Karp 1972). In contrast to this we will see in the next section a special case where the question for satisfiability is efficiently computable Btw: the corresponding problem 2SAT — satisfiablity of a formula in CNF where each clause has at most two literals — is decidable in polynomial time.

Now we get to know a class of formulas for which there is an efficient decision procedure for satisfiability.

1.4 Horn formulas

Definition 1.25. A formula F is called a *Horn formula*, if it is in CNF and if each disjunctive clause has at most one positive literal (i.e. one literal not containing $\neg$)

For example, $F = (\neg A \vee \neg B \vee C) \wedge (A \vee \neg D) \wedge (\neg A \vee \neg B \vee \neg C) \wedge D \wedge \neg E$ is a Horn formula, whereas $G = (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$ is not.

Each Horn formula can be written more intuitively using implications. For instance, the formula F above can be written as

$$(A \wedge B \Rightarrow C) \wedge (D \Rightarrow A) \wedge (A \wedge B \wedge C \Rightarrow 0) \wedge (1 \Rightarrow D) \wedge (E \Rightarrow 0),$$

where 1 stands for some arbitrary tautology and 0 stand for some arbitrary unsatisfiable formula.

Caution: Here we write "0" and "1" in the formulas as an exception. Normally this is a mistake (especially if you write $A = 0$ instead of $\mathcal{A}(A) = 0$).

There is an efficient algorithm checking for satisfiability of some Horn formula.

Algorithm 1.26. Input: some Horn formula F.

1. If F contains some partial formula $(1 \Rightarrow A)$, then mark all (!) literals in F containing the atomic formula A in F
2. **while** F contains some clause G with $G = (A_1 \wedge \cdots \wedge A_n \Rightarrow B)$ (with $n \geq 1$) where all A_i are marked (and B is not) **do**

   if $(B = 0)$ return "F is unsatisfiable" STOP

   else $B \neq 0$ (i.e. B is literal) **then** mark all (!) literals containing B in F,
3. Return "F is satisfiable" STOP

A valuation satisfying F is then given by setting $\mathcal{A}(A_i) = 1$ for all marked atomic formulas A_i and $\mathcal{A}(A_j) = 0$ for all unmarked atomic formulas A_j.

Theorem 1.27. *The algorithm above is correct for all Horn formulas and stops after at most k marking steps, where k denotes the number of atomic formulas contained in F.*
Moreover, the algorithm yields a minimal *satisfying valuation $\mathcal{A}$ for F. That is, for any other valuation $\mathcal{A}'$ with $\mathcal{A}' \models F$ holds that $\mathcal{A}(A_i) = 1$ implies $\mathcal{A}'(A_i) = 1$.*

The last statement can be seen as follows: when the algorithm starts all atomic formulas A_i are unmarked, i.e. their value is 0. All markings are forced, i.e. all marked A_i necessarily need to have value 1. If F is satisfiable the algorithm stops with a satisfying valuation in which all atomic formulas with value 1 are forced to have value 1. In this sense our obtained valuation $\mathcal{A}$ is minimal.

1.5 Compactness Theorem

Instead of individual formulas, we can also ask about the satisfiability of several formulas (finitely many or infinitely many) at once. That is, if $M = \{F_1, F_2, \ldots, F_n\}$ is a set of formulas, then we want to find an valuation $\mathcal{A}$ such that $\mathcal{A}(F_i) = 1$ for all $i = 1, \ldots, n$. If the set of formulas ist finite it is simple.

Remark 1.28. A set of formulas $\{F_1, F_2, \ldots, F_n\}$ is satisfiable if and only if $F = \bigwedge_{i=1}^{n} F_i$ is satisfiable.

It becomes interesting — and necessary for later purposes — when we consider infinite sets of formulas. In this context, the next result is essenttal.

Theorem 1.29 (compactness theorem)**.** *An infinite set M of formulas is satisfiable if and only if each finite subset of M is satisfiable.*

The precise proof is long and technical, see for instance UWE SCHÖNING: LOGIC FOR COMPUTER SCIENTISTS. A sketch of the proof is given below. However, the proof uses an important and interesting fact.

Theorem 1.30 (König's Lemma)**.** *Let T be a tree with infinitely many vertices such that each vertex has only finitely many neighbours. Then T contains an infinite path. (That is, a path $v_1 v_2 \cdots$ with infinitely many vertices such that $v_i \neq v_j$ for $i \neq j$.)*

Proof. (Entirely from Wikipedia:) Let v_i be the set of vertices. Since T is an infinite tree we know that this vertex set is infinite and the graph is connected (i.e., for any two vertices v_i, v_j there is a path in T from v_i to v_j).

Start with any vertex v_1. Every one of the infinitely many vertices of G can be reached from v_1 with a (simple) path, and each such path must start with one of the finitely many vertices adjacent to v_1. There must be one of those adjacent vertices through which infinitely many vertices can be reached without going through v_1. If there were not, then the entire graph would be the union of finitely many finite sets, and thus finite, contradicting the assumption that the graph is infinite. We may thus pick one of these vertices and call it v_2.

Now infinitely many vertices of G can be reached from v_2 with a simple path which does not include the vertex v_1. Each such path must start with one of the finitely many vertices adjacent to v_2. So an argument similar to the one above shows that there must be one of those adjacent vertices through which infinitely many vertices can be reached; pick one and call it v_3. Continue. □

König's Lemma implies the Compactness theorem essentially as follows: Draw a tree. One vertex is the root. Its children are all satisfying valuations $\mathcal{A}_1, \ldots, \mathcal{A}_k$ for F_1. The children of $\mathcal{A}_i$ are all satisfying valuations $\mathcal{A}'_1, \ldots, \mathcal{A}'_m$ for $F_1 \wedge F_2$ with $\mathcal{A}'_j(A_\ell) = \mathcal{A}_i(A_\ell)$ for all atomic formulas A_ℓ in F_1. (Hence $\mathcal{A}'_j$ is an extension from $\mathcal{A}_i$ to F_2).

We repeat the same for each $\mathcal{A}'_j$: Its nodes are all extensions of $\mathcal{A}'_j$ to all atomic formulas in F_3 that satisfy $F_1 \wedge F_2 \wedge F_3$. Due to the condition of the compactness theorem, there are an infinite number of satisfying valuations (corresponding to the nodes in the tree). By construction, each node has only finitely many children (possibly none), hence finitely many neighbors. Now it follows from König's lemma that there is an arbitrarily long path in the tree constructed in this way. This path corresponds to the valuation that fulfills all formulas.

Note that the proof does not tell us which numbers will do: it shows only the existence of such sequence, not the construction.

1.6 Consequences I

Definition 1.31. A formula G is a **consequence** (or *semantic consequence*) of the formulas $F_1, \ldots, F_n$, if: whenever $\mathcal{A}(F_1) = \cdots = \mathcal{A}(F_n) = 1$, then $\mathcal{A}(G) = 1$.

In this case, we write $\{F_1, \ldots, F_n\} \models G$ (see Definition 1.7).

The difference between "implies" ("$\Rightarrow$") and "is consequence of" ("$\models$") is subtle. The implication $\Rightarrow$ is on a syntactic level. $F \Rightarrow G$ just means $\neg F \vee G$. Whether it is "true" depends on the $\mathcal{A}$s: for some it may be true, for others not.

The consequence $\models$ lives on a higher level: If F holds then G holds for *each* valuation (respectively, in Section 2, for each "structure"). Whether $F \models G$ is "true" depends on *all* $\mathcal{A}$.

In addition, the following remark (and its proof) may be helpful.

Lemma 1.32. *The following statements are equivalent.*

1. $\{F_1, \ldots, F_n\} \models G$
2. $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ *is a tautology,*
3. $F_1 \wedge \cdots \wedge F_n \wedge \neg G$ *is unsatisfiable.*

Proof. For the equivalence of 1 and 2 we show the two implications separately.

1. implies 2.: Case 1: Let $\mathcal{A}(F_1) = \cdots = \mathcal{A}(F_n) = 1$. Since G is a consequence of F this implies $\mathcal{A}(G) = 1$. In this case $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ is true.

Case 2: Let not all $\mathcal{A}(F_i)$ equal one. Then $\mathcal{A}(F_1 \wedge \cdots \wedge F_n) = 0$, hence $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ is true. In both cases $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ is true, hence it is a tautology.

2. implies 1.: Let $H = F_1 \wedge \cdots \wedge F_n \Rightarrow G$ be a tautology.
Case 1: There is $\mathcal{A}$ such that $\mathcal{A}(F_i) = 0$. Then $\mathcal{A}(F_1 \wedge \cdots \wedge F_n) = 0$, hence this case does not matter.

Case 2: $\mathcal{A}(F_1) = \cdots = \mathcal{A}(F_n) = 1$. Now — since $\mathcal{A}(H) = 1$ — it follows $\mathcal{A}(G) = 1$. Thus altogether $\{F_1, \ldots, F_n\} \models G$ (since only Case 2 matters).

For the equivalence of 2. and 3. we need to compare whether $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ is a tautology with whether $F_1 \wedge \cdots \wedge F_n \wedge \neg G$ is unsatisfiable.

Let $F_1 \wedge \cdots \wedge F_n \Rightarrow G$ be a tautology. This means that $\neg(F_1 \wedge \cdots \wedge F_n \Rightarrow G)$ is unsatisfiable. Because of

$$\neg(F_1 \wedge \cdots \wedge F_n \Rightarrow G) \equiv \neg(\neg(F_1 \wedge \cdots \wedge F_n) \vee G) \equiv (F_1 \wedge \cdots \wedge F_n) \wedge \neg G$$

the latter term is unsatisfiable, too. □

It would be tempting to use truth tables. But this does not really work here. If we were to insist on the use of truth value tables: We must note that the following situation cannot happen:

$$\mathcal{A}(F_1) = \cdots = \mathcal{A}(F_n) = 1 \text{ and } \mathcal{A}(G) = 0.$$

In one direction, because G is a sequence of $\{F_1, \ldots, F_n\}$. In the other direction, because $F_1 \wedge \cdots \wedge F_n \Leftrightarrow G$ is a tautology. Therefore, we would only have to consider certain rows of the truth value table.

Remark 1.33. It is easy to see from the definitions that $F \equiv G$ holds if and only if $F \models G$ and $G \models F$. (Sure: let's assume the latter. So: whenever $\mathcal{A}(F) = 1$, then also $\mathcal{A}(G) = 1$ and vice versa. I.e. $\mathcal{A}(F) = 1$ exactly when $\mathcal{A}(G) = 1$; and that is $F \equiv G$).

Analogous to lemma 1.32 it also applies: $F \equiv G$ exactly if $F \Leftrightarrow G$ is a tautology.

$\{F, F \Rightarrow G\} \models G$	*(modus ponens)*
$\{\neg G, F \Rightarrow G\} \models \neg F$	*(modus tollens)*
$\{F \vee G, \neg F \vee H\} \models G \vee H$	*(resolution)*
$\{\neg(F \wedge G), F\} \models \neg G$	*(modus ponendo tollens)*
$\{F \Rightarrow G\} \models F \Rightarrow F \wedge G$	*(absorption)*
$\{F\} \models F \vee G$	*(disjuntion introduction)*
$\{F \Rightarrow G\} \models \neg G \Rightarrow \neg F$	*(contraposition)*

Table 2: Examples of consequences (aka inference rules)

The notion of consequence plays a central rule in several calculi. Another notation for $\{F_1, \ldots, F_n\} \models G$ is

$$\begin{array}{c} F_1 \\ F_2 \\ \vdots \\ F_n \\ \hline G \end{array} \quad \text{or} \quad \frac{F_1, F_2, \ldots, F_n}{G}$$

One of the best known examples of a semantic consequence is the *modus ponens*:

Lemma 1.34 (*modus ponens*)**.** $\{F, F \Rightarrow G\} \models G$

Using the two other notations above we may write it as

$$\frac{\begin{array}{c} F \\ F \Rightarrow G \end{array}}{G} \quad \text{, respectively} \quad \frac{F_1, F \Rightarrow G}{G}$$

In plain words this means: whenever F and $F \Rightarrow G$ are true (i.e., whenever $\mathcal{A} \models F$ and $\mathcal{A} \models F \Rightarrow G$) then G is also true under $\mathcal{A}$ (i.e., $\mathcal{A} \models G$). Why is this consequence rule true?

Proof. By Lemma 1.32 we need to show for instance that $F \wedge (F \Rightarrow G) \Rightarrow G$ is a tautology:

$$\begin{aligned} F \wedge (F \Rightarrow G) \Rightarrow G &\equiv \neg(F \wedge (\neg F \vee G)) \vee G \equiv \neg F \vee \neg(\neg F \vee G) \vee G \equiv \neg F \vee (F \wedge \neg G) \vee G \\ &\equiv \big((\neg F \vee F) \wedge (\neg F \vee \neg G)\big) \vee G \equiv (\neg F \vee \neg G) \vee G \equiv \neg F \vee \neg G \vee G \end{aligned}$$

The last expression is obviously a tautology. Hence $F \wedge (F \Rightarrow G) \models G$. □

1.7 Resolution

In logic, the generic term **calculus** is used to describe an algorithmic rule for deciding a question using logical formulas (for instance, $F \equiv G$, or F satisfiable). Truth tables are an example of a calculus, but an untypical one. A calculus in the narrower sense is a collection of transformation rules, and no transformation rules are used for truth value tables. (See also the definition 4.7 of a "formal system"). Other examples of calculi in the narrower sense are those that use **inference rule**s. The idea is to generate the semantic inferences from a given formula (or set of formulas). A particular calculus

uses the Ponens mode as an inference rule. In this section we present another such calculus that uses resolution as an inference rule.

Later we will write "G is inferred from F by means of a calculus $\mathcal{T}$", or "G is inferred from F by means of an inference rule $\mathcal{T}$" in short $F \vdash G$. If we want to emphasize the inference rule, we write $F \vdash_{\mathcal{T}} G$.

Remark 1.35. A calculus is particularly useful if it is **sound** and **complete**. The actual definition uses *axioms*, more on this later. Let us imagine here that we determine that the axioms should be true, and that we summarize the axioms (e.g., by ver-unden) in a single logical formula F. Then:

- The calculus is **sound** if from $F \vdash G$ always follows $F \models G$. (So for every G that we can derive from a true F with our rule, G is also "true").
- The calculus is **complete** if from $F \models G$ always follows $F \vdash G$. (So that every "true" statement can be derived with our rule).
- The calculus is **consistent**, if from $F \vdash G$ always follows $F \nvdash \neg G$. (This means that we cannot use our rule to derive a statement *and* derive its opposite).

In the context of this section, our goal is to decide whether F is is (un)satisfiable. Therefore, the terms here specifically mean something simpler:

- The calculus is **sound** if for every F for which it outputs "unsatisfiable", F is really unsatisfiable.
- The calculus is **complete** if for every unsatisfiable F outputs "unsatisfiable".

This is a subtle point: a calculus that is both sound and complete for the question "Is F unsatisfiable?" may not be complete for the question "Is F satisfiable?". For example, it may return nothing or runs forever if F is satisfiable.

15. Nov.

The resolution calculus uses just a single transformation rule to decide whether a formula F is satisfiable or unsatisfiable. (Later we will use the resolution calculus to decide whether a given formula in first-order logic is unsatisfiable. The effect described above occurs: if F is unsatisfiable, the answer is always correct. However, if F is satisfiable, it can happen that the resolution calculus does not give an answer).

Remark 1.36. A test for unsatisfiability of F answers several further questions, for instance

- Is F a tautology? (if and only if $\neg F$ is unsatisfiable)
- Is G a consequence of $F_1, \ldots, F_n$? (if and only if $F_1 \wedge \cdots \wedge F_n \wedge \neg G$ is unsatisfiable)

The first point is obvious: $\mathcal{A}(F) = 1$ for all $\mathcal{A}$ implies $\mathcal{A}(\neg F) = 1 - \mathcal{A}(F) = 0$ for all $\mathcal{A}$. The second point is Lemma 1.32.

In order to describe the resolution calculus for formulas in propositional logic we need some notation. Given some formula F in CNF:

$$F = (L_{1,1} \vee L_{1,2} \vee \cdots \vee L_{1,n_1}) \wedge (L_{2,1} \vee L_{2,2} \vee \cdots \vee L_{2,n_2}) \wedge \cdots \wedge (L_{k,1} \vee L_{k,2} \vee \cdots \vee L_{k,n_k})$$

where the L_{ij} are literals, we write F in **clause set notation** as

$$F = \big\{\{L_{1,1}, L_{1,2}, \ldots, L_{1,n_1}\}, \{L_{2,1}, L_{2,2}, \cdots, L_{2,n_2}\}, \cdots, \{L_{k,1}, L_{k,2}, \cdots, L_{k,n_k}\}\big\}.$$

Different formulas can have the same form in this notation. In fact the set notation reduces ambiguities. For instance the formulas

$$(A_1 \vee \neg A_2) \wedge A_3 \wedge A_3, \quad A_3 \wedge (\neg A_2 \vee A_1), \quad \text{and } A_3 \wedge (\neg A_2 \vee A_1 \vee \neg A_2)$$

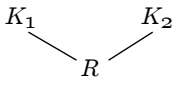
are all represented in set notation by $\big\{\{A_3\}, \{A_1, \neg A_2\}\big\}$.

Definition 1.37. Let K_1, K_2 be clauses, L a literal such that $L \in K_1$, $\neg L \in K_2$. Then the **resolvent** (of K_1 and K_2) is

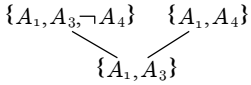
$$R = (K_1 \setminus \{L\}) \cup (K_2 \setminus \{\neg L\})$$

It may happen that $R = \{\}$ (for instance, if $K_1 = \{A\}$ and $K_2 = \{\neg A\}$). Since this means "unsatisfiable" in the sequel we write $R = \square$ in this case (rather than $R = \{\}$ or $R = \varnothing$, which could be confused with the empty clause).

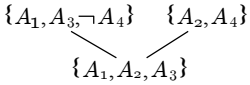
In a diagram we will write a resolvent R of K_1 and K_2 as



Example 1.38.



(using $L = A_4$) or



(also using $L = A_4$).

It is fruitful to think about whether it is OK to use two different literals (for instance, A_1 and $\neg A_1$ as well as A_4 and $\neg A_4$) at once in the same resolution step (see exercises).

Lemma 1.39. *Let F be a formula and let R be a resolvent. Then*

$$\{F \vee L, G \vee \neg L\} \models F \vee G$$

(The resolution $F \vee G$ is therefore a semantic consequence of $F \vee L$ and $G \vee \neg L$). Thus the following also applies: $F \equiv F \cup \{R\}$.

Hence the general idea of the resolution calculus is to determine all possible resolvents iteratively. F is unsatisfiable if and only if $\square$ appears at some point.

Notation: Let F be a formula in clause set notation.

- $\mathrm{Res}(F) = F \cup \{R \mid R \text{ resolvent of two clauses in } F\}$
- $\mathrm{Res}^0(F) = F$
- $\mathrm{Res}^{n+1}(F) = \mathrm{Res}(\mathrm{Res}^n(F))$
- $\mathrm{Res}^*(F) = \bigcup_{n \geq 0} \mathrm{Res}^n(F)$

Even though in general the questions "Is F unsatisfiable?" and "Is F satisfiable?" might be of different nature, in propositional logic they are the same with respect to resolution:

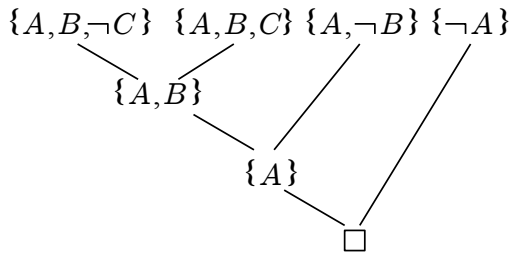
Theorem 1.40. *F is unsatisfiable if and only if $\square \in \mathrm{Res}^*(F)$. Since the procedure always terminates (see below) it follows that the resolution calculus is consistent and complete.*

The corresponding algorithm is clear now: compute iteratively $\mathrm{Res}^*(F)$. Stop if $\mathrm{Res}^k(F) = \mathrm{Res}^{k+1}(F)$ for some k (then $\mathrm{Res}^k(F) = \mathrm{Res}^*(F)$, see exercises). Return "F is unsatisfiable" if $\square \in \mathrm{Res}^*(F)$, return "$F$ is satisfiable" else.

The algorithm always terminates since we are dealing with finite formulas only. (In the next section we need to apply this algorithm to infinitely many formulas, hence it might not terminate if F is satisfiable.) We describe the algorithm by an example:

Example 1.41. Given $F = (A \vee B \vee \neg C) \wedge (A \vee B \vee C) \wedge (A \vee \neg B) \wedge \neg A$. We write F in clause set notation:

$$F = \big\{\{A,B,\neg C\},\{A,B,C\},\{A,\neg B\},\{\neg A\}\big\}.$$



Hence F is unsatisfiable.

In general, the diagram may not be a tree: sometimes one needs to use the same clause more than once, hence creating some loop.

The resolutions used in the example were chosen in some optimal way. The list of all resolvents is:

$$\mathrm{Res}^1(F) = F \cup \big\{\{A,B\},\{A,C\},\{A,\neg C\},\{B,\neg C\},\{B,C\},\{\neg B\}\big\}$$

$$\mathrm{Res}^2(F) = \mathrm{Res}^1(F) \cup \big\{\{A\},\{B\},\{C\},\{\neg C\}\big\}$$

A lot of work has been done on improving the algorithm by choosing resolvents in some optimal way. Nevertheless:

Theorem 1.42 (Haken 1984)**.** *There are formulas where each derivation of $\square$ needs exponentially many steps resolution in n, where n is the number of atomic formulas.*

Moreover, the CNF of some given formula F can be exponentially longer than F, see Remark 1.22.

Remark 1.43. The resolution calculus is efficient for Horn formulas: consider only those resolvents using clauses K_1 and K_2 where either K_1 has only one element, or K_2 has only one element. This can be seen by comparing the algorithms: This version of resolution simulates the Horn formula algorithm 1.3: Using clauses with one element covers all clauses in the Horn formula of the form $1 \Rightarrow A_i$. Marking means setting $\mathcal{A}(A_i) = 1$, hence implicitly adding a clause $\{A_i\}$. Marking all copies of A_i in $(A_i \wedge A_j \wedge \cdots \wedge A_m \Rightarrow B)$ means marking A_i in $(\neg A_i \vee \neg A_j \vee \cdots \vee \neg A_m \vee B)$ This corresponds to the clause $\{\neg A_i, \neg A_j, \cdots, \neg A_m, B\}$. The resolvent of the latter clause with $\{A_i\}$ is $\{\neg A_j, \cdots, \neg A_m, B\}$, and so on.

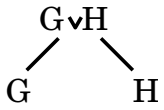
The resolution calculus is also efficient for formulas (in CNF) where each clause has at most two elements (see exercises).

There are several implementations of the resolution algorithm and of its refinements (Davis-Putnam-algorithm, Davis-Putnam-Logemann-Loveland algorithm, SAT-solvers...)

1.8 Tableau calculus

For later purposes we need a further formalism deciding the satisfiability of some formula in propositional logic that is *not* in CNF. It is assumed that the reader is familiar with the notion of a **binary tree** (root, vertex, child, leaf, path...). In the sequel "path" means a path ending in a leaf.

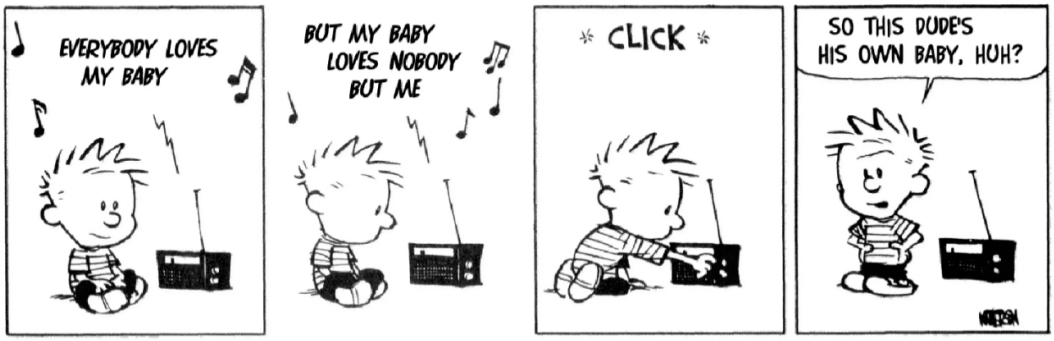
The idea is to break down a formula F into its sub-formulas with regard to its recursive structure. The sub-formulas gradually become nodes in the tree. The root of the tree is F. At the end, the paths in the tree are possible valuations of F. A formula of the form $G \vee H$ leads to a branch



meaning that either G or H (or both) must be true. A formula of the form $G \wedge H$ leads to a branch

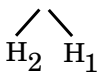
meaning that both G or H must be true. We continue to unravel the formula into a tree until all $\wedge$ and $\vee$ are translated into nodes of the tree. Then all leaves are literals. The finished tree is called **tableau**.

Then we check for satisfiability: If we see for instance A_i and $\neg A_i$ in some path, then this path does not yield a satisfying valuation (since in this path both A and $\neg A$ must be true). A formula is satisfiable if there is at least one path that is satisfiable in this sense. The general method is as follows.

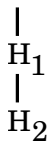


Algorithm 1.44. Input: some formula F in propositional logic (without $\Rightarrow$ and $\Leftrightarrow$).

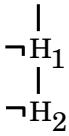
1. Start with F as the root.
2. Choose an unmarked vertex G that is not a literal. Mark G. Apply the following rules until all possibilities are exhausted.
   - If G is of the form $\neg\neg H$ then add a single vertex H to each path starting in G.
   - If G is of the form $H_1 \vee H_2$ then add to each path starting in G this:



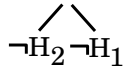
   - If G is of the form $H_1 \wedge H_2$ then add to each path starting in G this:



   - If G is of the form $\neg(H_1 \vee H_2)$ then add to each path starting in G this:



   - If G is of the form $\neg(H_1 \wedge H_2)$ then add to each path starting in G this:



3. If there is no further vertex to mark then return the tableau, STOP.

A path is **closed** if it contains A_i and $\neg A_i$ for some i. In this case we mark the leaf of this path by $\otimes$. A tableau is **closed** if all leaves are marked by $\otimes$. In this case F is unsatisfiable. Else F is satisfiable, and each path with a leaf having no $\otimes$ yields a satisfying valuation $\mathcal{A}$: for each occurrence of A_i set $\mathcal{A}(A_i) = 1$, for each occurrence of $\neg A_i$ set $\mathcal{A}(A_i) = 0$.

There are some obvious simplifications to the algorithm. For example, a formula of the form $A \vee B \vee C$ can be solved in one step, simply appending three children (A, B and C) to each path through this node instead of two (the tree is no longer a binary tree in this case, but that doesn't bother us, does it?). Or: you don't need to add any more nodes to an already closed path. This has already been done in Figure 1 with the nodes $\neg\neg A$ marked in the last figure.

There are also several refinements of this algorithm in order to improve the runtime. Two obvious examples are: paths can be marked closed as soon as they are detected to be closed (not just at the end). And: A partial formula $H_1 \wedge H_2 \wedge H_3$ can be unraveled in one step, resulting in three new vertices (see Example 1.46).

Theorem 1.45. *The Tableau algorithm of propositional logic is consistent and complete.*

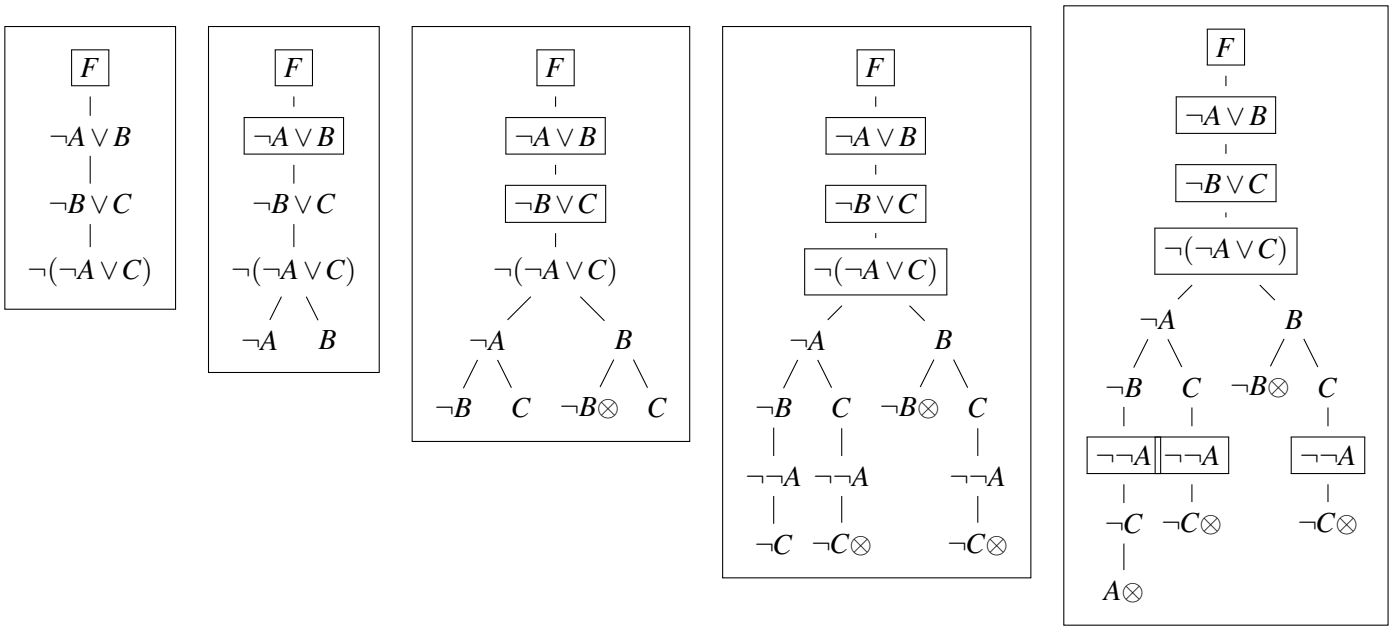


Figure 1: An example of the Tableau algorithm. For each new marking (boxed formula) there is a new diagram ((1)-(5)). (Thanks to Andreas Mazur)

A proof is contained in the book of Kreuzer-Kühling.

Example 1.46. We show that $A \Rightarrow C$ is a consequence of $\{A \Rightarrow B, B \Rightarrow C\}$. In other words, we show $\{A \Rightarrow B, B \Rightarrow C\} \models A \Rightarrow C$. Because of Remark 1.36 we can do this by showing that

$$F = (A \Rightarrow B) \wedge (B \Rightarrow C) \wedge \neg(A \Rightarrow C) \equiv (\neg A \vee B) \wedge (\neg B \vee C) \wedge \neg(\neg A \vee C)$$

is unsatisfiable. The resulting tableau is shown in Figure 1 is shown. All paths are closed, therefore F is unsatisfiable, therefore the statement holds.

Interlude: Relations

22. Nov

In the next section predicates will play a central role. Predicates are a generalisation of relations. (Predicates can have one, two, three... inputs, whereas relations have always two inputs.)

A relation is explained on some set W. Think of W as all students in the Faculty of Technology, or all integers, or all 0-1-words. In plain words a relation is some property that any pair of elements in W may have or not have. A relation on of the set W of all Techfakstudis can be "a knows 'b". For instance, on $W = \mathbb{Z}$ a relation may be $<$: $2 < 5$ is true, so the relation is true for the pair $(2,5)$. The relation is not true for the pair $(5,2)$, or the pair $(3,3)$.

Another relation on $W = \mathbb{Z}$ is "$a - b$ is odd". So the relation is true for $(2,5)$ and $(5,2)$, but not for $(3,3)$.

Recall that for two sets V, W the **Cartesian product** is

$$V \times W := \{(a,b) \mid a \in V, b \in W\}.$$

A convenient way to define a relation is to view it as a subset R of $W \times W$. For the $<$-example we obtain

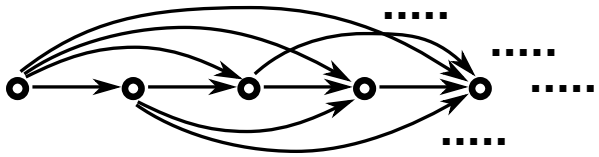
$$R = \{(a,b) \mid a,b \in \mathbb{Z}, a < b\}$$

So $(2,5) \in R$, $(5,2) \notin R$, $(3,3) \notin R$. For the "$a-b$ is odd"-example we obtain

$$R' = \{(a,b) \mid a-b \text{ is odd.}\}$$

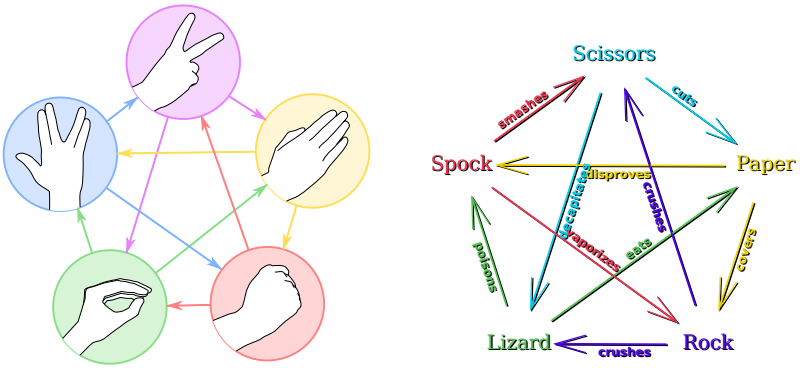
So $(2,5) \in R$, $(5,2) \in R$, $(3,3) \notin R$.

Example 1.47. A relation R on a set W can be visualized as a **directed graph**. The elements of W are the nodes of the graph, and an element $(m,n) \in R$ results in a directed edge from m to n. This is what the graph looks like for the relation $(\mathbb{N}_0, <)$ looks partly as follows:



In particular, this graph is an infinite graph. Furthermore an infinite number of edges extend from each node. A simpler example for a time frame would be the days of the week $\{1,2,\ldots,7\}$, ordered by using $<$, or also using "next day".

Another, simpler example is the rock-paper-scissors-Spock-lizard game. Here $W = \{$rock, paper, scissors, lizard, Spock$\}$, and $R = \{(n,m) \mid n,m \in W,\ n \text{ beats } m\}$. The corresponding diagram is (two versions)



A particular nice class of relations are equivalence relations.

Definition 1.48. Let R be a relation. $R \subseteq W \times W$ is called an **equivalence relation**, if for all $a,b,c \in W$ holds:

1. $(a,a) \in R$, *(reflexive)*
2. $(a,b) \in R$ if and only if $(b,a) \in R$, and *(symmetric)*
3. $(a,b) \in R$ and $(b,c) \in R$ implies $(a,c) \in R$. *(transitive)*

An equivalence relation partitions the set W into **equivalence classes**: By $[a]$ we denote the set of all $b \in W$ such that $(a,b) \in R$. A simple example of an equivalence relation is $=$ in $\mathbb{Z}$. Here each equivalence class $[a]$ consists of one element only, namely, a. The examples R, R' above are both not equivalence relations. (Why not?) Thus, they do not divide the set $\mathbb{Z}$ into cleanly separated subsets.

2 First-order logic

In propositional logic, we cannot further divide the statement "x is a Chinese" into sub-statements. Neither can the statement "for all $n \in \mathbb{N}$ applies: $n \geq 0$". Therefore, we now extend the language of propositional logic with further building blocks such as **variables** (like x). A statement "x is chinese" then is a **predicate**, its value being 0 or 1, depending on whether x is chinese or not. Furthermore adding **quantifiers** $\forall$ and $\exists$ and **functions** to the language of propositional logic yields the language of first-order logic. This allows for the formal logical treatment of statements like

$$\forall \varepsilon \, \exists \delta \, \forall x \, (|x-a| < \delta \Rightarrow |f(x)-f(a)| < \varepsilon)$$

Here $\forall$ and $\exists$ are quantifiers, f and $|\cdot|$ and $-$ are functions, ε, δ, x and a are variables, and $<$ is a predicate.

As in Chapter 1, the general structure of this chapter is (1.) formal definition of syntax and (2.) semantics, (3.) normal forms, (4.) algorithms for deciding on the (un)satisfiability of formulas. Unlike in Chapter 1, we will see here some problems that are algorithmically undecidable (i.e. not only not efficient, but not at all!).

2.1 Syntax of first-order logic

As in Section 1 we first declare what are valid formulas in a purely abstract way (syntax). Later we see how to fill this formulas with concrete meaning (semantics).

Definition 2.1 (syntax of first-order logic)**.** The building blocks of formulas are

- **Variables** denoted by $x_1, x_2, \ldots$ or u, v, w, x, y, z.
- **Predicate**-symbols denoted by $P_1, P_2, \ldots$ or P, Q, R.
- **Function**-symbols denoted by $f_1, f_2, \ldots$ or f, g, h.
- **Terms** are denoted $t_1, t_2, \ldots$ and are defined inductively:
  - Each variable is a term.
  - If f is a function (with k inputs) and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term.
- **Formulas** are defined inductively:
  - If P is a predicate (with k inputs) and $t_1, \ldots, t_k$ are terms, then $P(t_1, \ldots, t_k)$ is a formula. (Sometimes these are called *atomic* formulas)
  - If F is a formula, then $\neg F$ is.
  - If F and G are formula, then $F \vee G$ and $F \wedge G$ are.
  - If x is a variable and F is a formula then $\forall x F$ and $\exists x F$ are formulas.

All intermediate steps of this inductive construction are called **partial formulas**. An occurrence of some variable x in some partial formula G of some formula F is **bound** if G is of the form $\forall x H$ or $\exists x H$ such that H contains x; otherwise this occurrence of x is called **free**. Note that a variable can be both bound and free in the same formula, see the example below. A formula is **closed** if all occurrences of all variables are bound. The **matrix** of F is obtained by deleting all $\forall x$ and $\exists x$. Functions with no inputs are allowed. Such functions are also called **constants**.

Remark 2.2. Because "being equal" is such a natural and frequently occurring concept, it is sometimes useful to allow a further symbol: $=$. The definition of the syntax has to be adjusted appropriately: If t_1 and t_2 are terms, then $t_1 = t_2$ is a formula.

Example 2.3. $F = \exists x P(x, f_1(y)) \vee \neg\forall y Q(y, f_7(f_2, f_3(z)))$ is a formula. All partial formulas of F are

$$F,\ \exists x P(x, f_1(y)),\ P(x, f_1(y)),\ \neg\forall y Q(y, f_7(f_2, f_3(z))),\ \forall y Q(y, f_7(f_2, f_3(z))),\ Q(y, f_7(f_2, f_3(z))).$$

All terms in F are $x, y, f_1(y), f_7((f_2, f_3(z)), f_2, f_3(z), z$. All occurrences of x in F are bound. The first occurrence of y in F is free, the second is bound. The occurrence of z in F is free. The matrix of F is

$$P(x, f_1(y)) \vee \neg Q(y, f_7(f_2, f_3(z))).$$

Remark 2.4. In order to use fewer brackets we agree on the rule that $\forall x$ and $\exists x$ binds stronger than $\vee$ and $\wedge$. Hence $\forall x\, P(x) \vee \forall y\, P(y)$ means $(\forall x\, P(x)) \vee (\forall y\, P(y))$, and not $\forall x\, (P(x) \vee \forall y\, P(y))$.

2.2 Semantics of first-order logic

In order to interpret formulas in first-order logic as "true" or "false" one needs to give the symbols a meaning. More precisely, one needs to define a basic set in which the variables and functions take their values (the universe), and an interpretation of any predicate-symbol as a predicate (relation) in this universe, any function symbol as a function, and so on. More precisely:

Definition 2.5. A **structure** (aka world) for a formula F is a pair $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, where

- $U_\mathcal{A}$ is a nonempty set (the **universe**), and
- $I_\mathcal{A}$ is a map (the **interpretation**) such that
  - $I_\mathcal{A}(x)$ is an element of $U_\mathcal{A}$ for all free variable-symbols x in F,
  - $I_\mathcal{A}(f)$ is a function over $U_\mathcal{A}$ for all function-symbols f in F,
  - $I_\mathcal{A}(P)$ is a predicate over $U_\mathcal{A}$ for all predicate-symbols P in F.

Here "element" has the usual meaning: some element of $U_\mathcal{A}$. "Function" means a function from $(U_\mathcal{A})^k \to U_\mathcal{A}$, where f has k inputs. "Predicate" is to be thought of as some property. E.g., for a predicate P with one input $P(x)$ may mean "x is odd", or "x is a prime number" (if $U_\mathcal{A} = \mathbb{N}$), or $P(x)$ may mean "x is chinese" if $U_\mathcal{A}$ is the set of all humans. For a predicate P with two inputs $P(x,y)$ can mean $x < y$, or x divides y (if $U_\mathcal{A} = \mathbb{N}$), or $P(x,y)$ can mean "x knows y" if $U_\mathcal{A}$ is the set of all humans.

Remark 2.6. . More formally a predicate P is defined as a subset of $U_\mathcal{A}$ (if P has one input), or a subset of $U_\mathcal{A} \times U_\mathcal{A}$ (if P has two inputs), or a subset of $U_\mathcal{A} \times U_\mathcal{A} \times \cdots \times U_\mathcal{A}$ (n times) if P has n inputs. Then P is for instance

$$\{x \in \mathbb{N} \mid x \text{ is odd }\}, \quad \{x \in \mathbb{N} \mid x \text{ is prime }\},$$

respectively

$$\{(n,m) \in \mathbb{N} \times \mathbb{N} \mid n < m\}, \quad \{(n,m) \in \mathbb{N} \times \mathbb{N} \mid n \text{ divides } m\}$$

for the examples mentioned above.

In the sequel we will write shortly $x^\mathcal{A}$, $f^\mathcal{A}$, $P^\mathcal{A}$ rather than $I_\mathcal{A}(x)$, $I_\mathcal{A}(f)$, $I_\mathcal{A}(P)$.

Example 2.7. Consider the formula $F' = (\forall x P(x, f(x))) \wedge Q(g(h,x))$. Here x is both a bound variable and a free variable, f is a function with one input, g is a function with two inputs, and h is a function with no input; hence h is a constant. P is a predicate with two inputs, Q is a predicate with one input.

In order to avoid confusion (because of x free versus x bound) let us consider $F = (\forall x P(x, f(x))) \wedge Q(g(h,z))$. A possible structure for F is

$$\begin{aligned} U_{\mathcal{A}} &= \{0,1,2,\ldots\} = \mathbb{N}_0 \\ P^{\mathcal{A}} &= \{(x,y) \in \mathbb{N}_0 \times \mathbb{N}_0 \mid x < y\} \\ Q^{\mathcal{A}} &= \{x \in \mathbb{N}_0 \mid x \text{ is prime } \} \\ f^{\mathcal{A}} &: \mathbb{N}_0 \to \mathbb{N}_0,\ f(x) = x+1 \\ g^{\mathcal{A}} &: \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0,\ g(x,y) = x+y \\ h^{\mathcal{A}} &= 2 \quad \text{and} \quad z^{\mathcal{A}} = 3. \end{aligned}$$

This structure does not only match F (i.e.: it explains everything, that appears in F), but it makes F true! Indeed, in this universe F means

$$(\forall x \in \mathbb{N}_0\ x < x+1) \wedge 2+3 \text{ is prime.}$$

Since both statements are true, F is true in the structure $\mathcal{A}$. We will write shortly $\mathcal{A} \models F$.

If we change the structure above into $\mathcal{A}'$ by changing $z^{\mathcal{A}} = 3$ to $z^{\mathcal{A}'} = 4$ then F is false in the new structure $\mathcal{A}'$ (since $2+4=6$ is not prime).

In the sequel we will focus on the question "Given some formula F, is there some structure making F true?" (satisfiability); respectively "Given some formula F, is F true for all structures (matching F)?" (compare tautology in Section 1, here this will be denoted as "valid") In the spirit of Def. 1.3 we will define what it means that $\mathcal{A}$ makes F true. 29. Nov.

Definition 2.8. Let F be a formula and $\mathcal{A}$ be a structure for F. $\mathcal{A}(F)$ is defined inductively by Definition 1.3 together with the following points

1. If $F = P(t_1, \ldots, t_k)$ then
$$\mathcal{A}(F) = \begin{cases} 1 & \text{if } (t_1^{\mathcal{A}}, \ldots, t_k^{\mathcal{A}}) \in P^{\mathcal{A}}, \\ 0 & \text{else} \end{cases}$$
2. If $F = \forall x G$ then
$$\mathcal{A}(F) = \begin{cases} 1 & \text{if for all } x^{\mathcal{A}} \in U_{\mathcal{A}} \text{ holds } \mathcal{A}(G) = 1 \\ 0 & \text{else} \end{cases}$$
3. If $F = \exists x G$ then
$$\mathcal{A}(F) = \begin{cases} 1 & \text{if there is } x^{\mathcal{A}} \in U_{\mathcal{A}} \text{ with } \mathcal{A}(G) = 1 \\ 0 & \text{else} \end{cases}$$

Given a formula F and a structure $\mathcal{A}$ for F.

- If $\mathcal{A}(F) = 1$ then $\mathcal{A}$ **satisfies** F, short: $\mathcal{A} \models F$ We say also in this case: $\mathcal{A}$ is a **model** for F.

- If for all $\mathcal{A}$ for F holds $\mathcal{A} \models F$ then F is **valid**, short: $\models F$ (this is the analogue concept of tautology in propositional logic).
- If there is $\mathcal{A}$ for F such that $\mathcal{A} \models F$ then F is **satisfiable**.

Example 2.9. Let $F = \forall x \exists y\, P(x,y)$. A structure $\mathcal{A}$ with $\mathcal{A} \models F$ is

$$U_{\mathcal{A}} = \mathbb{N}, \quad P^{\mathcal{A}} = \{(x,y) \mid x < y, x \in \mathbb{N}, y \in \mathbb{N}\}.$$

("For all $x \in \mathbb{N}$ exists $y \in \mathbb{N}$ such that $x < y$", bingo)

A structure $\mathcal{A}$ with $\mathcal{A} \not\models F$ is for instance

$$U_{\mathcal{A}} = \{0,1,2,\ldots,9\} \quad P^{\mathcal{A}} = \{(x,y) \mid x,y \in U_{\mathcal{A}},\, x = 2y,\}.$$

In this case F is false for $x = 1$.

In order to illustrate that a structure is not necessarily a mathematical object: Let $U_{\mathcal{A}}$ be the set of all humans, and let $P^{\mathcal{A}}(x,y)$ mean "x loves y". Consider all combinations of quantifier-variable-quantifier-variable $P(x,y)$. Then

- $\forall x\, \exists y\, P(x,y)$ is "everybody loves someone",
- $\forall y\, \exists x\, P(x,y)$ is "everybody is loved by someone",
- $\exists x\, \forall y\, P(x,y)$ is "there is someone loving everybody" (Jesus?),
- $\exists y\, \forall x\, P(x,y)$ is "there is someone loved by everybody" (Elvis??),
- $\forall x\, \forall y\, P(x,y)$ is "everybody loves everybody" (Woodstock???),
- $\exists x\, P(x,x)$ is "somebody loves himself".

The truth of each formula depends on $\mathcal{A}$, but all these formulas are satisfiable (in a universe where everybody loves everybody), and none of these formulas is valid (= true in all possible universes): we can just define a world where no one loves no one by choosing P to be the empty predicate.

Remark 2.10. Again, for first-order logic with identity, the definition of the semantics needs to be adjusted. This is done by giving the predicate $=$ the usual meaning: being equal as elements in $U_{\mathcal{A}}$.

Remark 2.11. In analogy to Remark 1.36 one can show here that

- F is valid if and only if $\neg F$ is unsatisfiable.
- G is a consequence of $F_1 \wedge \cdots \wedge F_n$ if and only if $F_1 \wedge \cdots \wedge F_n \wedge \neg G$ is unsatisfiable.
- $F \equiv G$ if and only if $F \models G$ and $G \models F$ if and only if $F \Leftrightarrow G$ is a tautology.

Remark 2.12. First-order logic does indeed contain propositional logic as a special case. In fact each formula in first-order logic without quantifiers can be interpreted as a formula in propositional logic: all variables are free, hence treated as constants. All predicates contain only constants. Hence variables as such vanish. For instance,

$$F = (P(g) \vee \neg Q(f(g),h)) \wedge R(a,b)$$

corresponds in each interpretation to constants and predicates of constants. The latter are the atomic formulas, so F becomes

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

The latter is satisfiable (resp. a tautology) if and only if the former is satisfiable (resp. valid).

Remark 2.13. Not every statement can be expressed in first-order logic. Up to now we need to deal with one universe: we cannot express "for all students there is an integer" (at least not easily). Moreover, we cannot express statements like "for all functions", or "it exists a predicate". Allowing for these possibilities yields *second-order logic*. The latter is beyond the scope of this course.

2.3 Normal forms

All laws of calculation in propositional logic (see Theorem 1.10) do also apply to first-order logic. We need some further laws for treating quantifiers. These laws tell us when two formulas are equivalent. Being equivalent here means essentially the same as in Section 1: $F \equiv G$ means that for all $\mathcal{A}$ holds that $\mathcal{A}(F) = \mathcal{A}(G)$. The definition is the same only the meaning of $\mathcal{A}$ has changed.

Theorem 2.14. *Let F and G be formulas. Then the following rules hold.*

1. $\neg\forall x\, F \equiv \exists x\, \neg F$
   $\neg\exists x\, F \equiv \forall x\, \neg F$
2. *If x does not occur freely in G, then*
   $(\forall x\, F) \wedge G \equiv \forall x\, (F \wedge G)$
   $(\forall x\, F) \vee G \equiv \forall x\, (F \vee G)$
   $(\exists x\, F) \wedge G \equiv \exists x\, (F \wedge G)$
   $(\exists x\, F) \vee G \equiv \exists x\, (F \vee G)$
3. $(\forall x\, F) \wedge (\forall x\, G) \equiv \forall x\, (F \wedge G)$
   $(\exists x\, F) \vee (\exists x\, G) \equiv \exists x\, (F \vee G)$
4. $\forall x\, \forall y\, F \equiv \forall y\, \forall x F$
   $\exists x\, \exists y\, F \equiv \exists y\, \exists x F$

For first-order logic with identity we have three further laws. But they are not very exciting. E.g. applies $x = x$, or if $x = y$, then $f(x) = f(y)$. etc.

Proof. The proofs 1, 3 and 4 are standard, technical and do not provide any deeper insights. We therefore only show the proof of the first formula in point 2 of the theorem. With definition 2.8 we get:

$$\begin{aligned}
\mathcal{A}((\forall x\, F) \wedge (\forall x\, G)) = 1 \text{ iff } & \mathcal{A}(\forall x\, F) = 1 \text{ and } \mathcal{A}(G) = 1 \\
\text{iff } & (\text{ for all } x \in U_{\mathcal{A}} :\ \mathcal{A}(F) = 1) \text{ and } \mathcal{A}(G) = 1 \\
\text{iff } & \text{for all } x \in U_{\mathcal{A}} :\ (\mathcal{A}(F) = 1 \text{ and } \mathcal{A}(G) = 1) \\
& \qquad (\text{because } x \text{ does not occur freely in } G) \\
\text{iff } & \text{for all } x \in U_{\mathcal{A}} :\ \mathcal{A}(F \wedge G) = 1 \\
\text{iff } & \mathcal{A}(\forall x(F \wedge G)) = 1.
\end{aligned}$$

Here, "iff" means: "if and only if". (Again, the subtle reason is: the sign $\Leftrightarrow$ is purely on the syntactic level, but here however, we are arguing on a different level). All the rules above can be proved in this way. □

Analogous to chapter 1, we use the rules above to create a normal form of a given formula F. However, this is a bit longish here. First, we bring all the quantifiers to the front.

Example 2.15.

$$\begin{aligned}\neg(\exists x\, P(x,y) \vee \forall z\, Q(y)) \wedge \exists w\, Q(w) &= \neg\exists x\, P(x,y) \wedge \neg\forall z\, Q(y) \wedge \exists w\, Q(w)\\ &= \forall x\, \neg P(x,y) \wedge \exists z\, \neg Q(y) \wedge \exists w\, Q(w)\\ &= \forall x\, \big(\neg P(x,y) \wedge \exists z\, \neg Q(y)\big) \wedge \exists w\, Q(w)\\ &= \forall x\, (\exists z(\neg P(x,y) \wedge\ \neg Q(y))) \wedge \exists w\, Q(w)\\ &= \exists w\, \forall x\, \exists z\, (\neg P(x,y) \wedge \neg Q(y) \wedge Q(w))\end{aligned}$$

Note that the order of the quantifiers above depends on the transformations used. In particular, the order ambiguous. In this case the quantifiers can be arranged to any order, but this is not always the case.

A problem occurs if we want to use rule 2: $\forall x\, F \wedge G \equiv \forall x\, (F \wedge G)$ and if a variable x is free in G (and bound in $\forall x\, F$, e.g. $\big(\forall x\, P(x)\big) \wedge Q(x)$).

Lemma 2.16. *Let $F[x/y]$ denote the formula obtained by replacing each occurrence of x in F by y. Let G be some formula not containing y. Then*

$$\forall x\, G \equiv \forall y\, G[x/y] \quad \textit{and} \quad \exists x\, G \equiv \exists y\, G[x/y]$$

Using this lemma we may transform each formula into one where no two quantifiers have the same variable, and where no variable occurs both bound and free. This is the requirement in order to construct the following normal form.

The next definition describes an intermediate state: the first thing we want to ensure is that all quantifiers are as far left as possible. Let us make this precise.

Definition 2.17. A formula F has **prenex normal form** (PNF) if

$$F = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n G,$$

where the Q_i are quantifiers, and G contains no further quantifiers.

In other words, all variables in G (!) are free, respectively G is the matrix of F. For instance

$$F = \exists x\, \forall y\, \exists z\, ((P(f(x)) \wedge P(y)) \vee Q(z))$$

has PNF, whereas

$$F = \exists x\, \exists y\, (P(f(x)) \wedge P(y)) \vee \forall z\, Q(z)$$

has not.

Theorem 2.18. *Each formula in first-order logic has an equivalent formula in PNF.*

Again, this theorem is proven using the inductive definition of a formula. Lemma 2.16 provides enough freedom to rename the variables in order to achieve this.

For the algorithmic treatment of (satisfiability of) formulas we need a more special normal form. Maybe the exercises gave an idea already on the fact that it makes no big difference whether a variable is free, or is bound by an $\exists$. So the next step is to delete all $\exists$ in some appropriate manner.

Definition 2.19. Given a formula in PNF its corresponding **Skolem normal form** (SNF) is the output of the following algorithm:

1. while F contains $\exists$ do
   - Is $F = \forall y_1\, \forall y_2\, \cdots \forall y_n\, \exists x\, G$
     then choose a function f not contained in G and set $F := \forall y_1\, \forall y_2\, \cdots \forall y_n\; G[x/f(y_1,\ldots,y_n)]$.
2. STOP

Note that G can contain $\forall$. For instance, if $F = \forall x \exists y \forall z\, P(x,y,z)$, then $G = \forall z\, P(x,y,z)$, and F becomes $\forall x \forall z\, P(x, f(x), z)$.

The case $n = 0$ is possible, that is, F is of the form $\exists x G$. In this case f has no arguments, hence f is a constant. Consequently, F is replaced by $G[x/f]$ in this case. For a further illustration see Example 2.23 below.

Theorem 2.20. *For each formula F in first-order logic there is a formula G in SNF, such that F is satisfiable if and only if G is.*

Definition 2.19 provides us with the recipe on how to get the G. After this step, we bring the matrix of the formula to CNF (see Chapter 1, Def. 1.13).

Remark 2.21. Obviously, constants play a particular role here. Up to now we used function symbols $f, g, h...$ for constants (where $f, g, h...$ had no inputs). For the sake of clarity, from now on we will use $a, b, c...$ or $a_1, a_2, ...$ for constants.

An important insight in this chapter is that a constant is the same as a free variable in practically every respect, and both are almost the same as a variable bound by an existential quantifier. This is because they behave in principle the same under any structure $\mathcal{A}$, see Def. 2.5: a function f with no input (i.e. a constant a) must be bound by a structure $\mathcal{A}$ must be explained by a constant value in $U_\mathcal{A}$ (number or person or...). So $f^\mathcal{A} \in U_\mathcal{A}$ (or $a^\mathcal{A} \in U_\mathcal{A}$).

A free variable x must be explained by a value in $U_\mathcal{A}$, so $x^\mathcal{A} \in U_\mathcal{A}$. And for a variable y bound by $\exists$, $\mathcal{A}(\exists y\, F) = 1$ if there is at least one value in $y \in U_\mathcal{A}$ with $\mathcal{A}(F) = 1$. In all three cases, if $\mathcal{A}$ is a model, then there is an element in $U_\mathcal{A}$ that makes the respective formula true. This allows you to roughly understand why theorem **??** applies.

For reasons of space, a cartoon is shown here.

The following list summarizes the steps needed to transform some arbitrary formula into the normal form we need later. This normal form does not seem to have a particular name in the literature, even though it is the most important one in the sequel. So we call it THE normal form.

Algorithm 2.22 (THE normal form (TNF)).

0. Replace all partial formulas $F \Rightarrow G$ by $\neg F \vee G$, and all $F \Leftrightarrow G$ by $(F \wedge G) \vee (\neg F \wedge \neg G)$.
1. Rename bound variables in F until no two quantifiers have the same variable, and where no variable occurs both bound and free.
2. Let $y_1, \ldots, y_n$ be all free variables in F. Replace each y_i by a constant a_i.
3. Transform F into PNF (by pushing quantifiers to the front).
4. Delete all $\exists$ by transforming F into Skolem normal form.
5. Transform the matrix of F into CNF.

Output: the resulting formula (with the all quantifiers, i.e. not just the matrix)

Note that only steps 0, 1, 3 and 5 preserve the equivalence of formulas. In general, the output at the end will not be a formula that is equivalent to the original formula. However, the result is satisfiable if the original formula is.

Example 2.23. A full example illustrating all the steps needed to establish TNF: Consider

$$F = \exists x \, \forall y \, \big(P(x) \Rightarrow Q(y)\big) \wedge P(x) \wedge \forall y \, \neg Q(y).$$

Applying steps 0-5 above yields

$$\begin{aligned}
F &\overset{0.}{\equiv} \exists x \forall y \, \big(\neg P(x) \vee Q(y)\big) \wedge P(x) \wedge \forall y \, \neg Q(y) \\
&\overset{1.}{\equiv} \exists w \forall y \, \big(\neg P(w) \vee Q(y)\big) \wedge P(x) \wedge \forall z \, \neg Q(z) \\
&\overset{2.}{\simeq} \exists w \forall y \, \big(\neg P(w) \vee Q(y)\big) \wedge P(a) \wedge \forall z \, \neg Q(z) \\
&\overset{3.}{\equiv} \forall z \exists w \forall y \, \Big(\big(\neg P(w) \vee Q(y)\big) \wedge P(a) \wedge \neg Q(z)\Big) \\
&\overset{4.}{\simeq} \forall z \forall y \, \Big(\big(\neg P(f(z)) \vee Q(y)\big) \wedge P(a) \wedge \neg Q(z)\Big)
\end{aligned}$$

The matrix of the last formula already has CNF, so we don't need to do anything for step 5. We use the character $F \cong G$ here provisionally for "F satisfiable exactly when G is satisfiable".

2.4 Resolution calculus of first-order logic

We will see in Section 4 that we cannot hope for anything better than a test for unsatisfiability of a formula F that returns "Yes" whenever F is unsatisfiable, and may return nothing (runs forever) if F is satisfiable. Moreover, we may wait for a positive answer arbitrarily long: if there would be some bound on the waiting time this test could be turned into some test on satisfiability ("wait long enough, if algorithm did not terminate, return 'satisfiable'"). This test would just not be efficient.

The problem is essentially that we need to test infinitely many structures, partly because of the following result:

Theorem 2.24. *There are satisfiable formulas in first-order logic possessing only infinite models; that is, models $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ such that $U_\mathcal{A}$ is an infinite set.*

Proof. (Exercise) □

The solution to the problem is to use a certain standard structure.

Definition 2.25. The **Herbrand universe** $H(F)$ of some formula F is the set of all variable-free terms that can be constructed out of partial formulas of F. That is, $H(F)$ consists of

1. All constants $a, b, \ldots$ occurring in F are in $H(F)$.
2. If $t_1, t_2, \ldots t_n \in H(F)$ and f occurs in F then $f(t_1, t_2, \ldots, t_n) \in H(F)$.

If F contains no constant then add one constant a to $H(F)$ and proceed as above.

Note the twist here: the elements of the universe *are* the symbols in F. This means we define the semantics to be the syntax. Consequently, any two different symbols are different elements of $H(F)$.

Example 2.26. Let $F = \forall x \, \forall y \, \forall z \, P(x, f(y), g(z,x))$. F contains no constant, so we add a to $H(F)$ and get

$$H(F) = \{a, f(a), g(a,a), f(f(a)), f(g(a,a)), g(a, f(a)), g(f(a), a), g(f(a), f(a)), f(g(a, f(a))), \ldots\}$$

Let $G = \forall x \, \forall y \, Q(a, f(z), h(y,b))$. G contains two constants a, b, and we get

$$\begin{aligned} H(G) = \{&a, b, f(a), f(b), h(a,a), h(a,b), \\ &h(b,b), h(b,a), f(f(a)), f(f(b)), \\ &f(h(a,a)), f(h(a,b)), h(f(a), b), \\ &h(f(b), b), h(a, f(a)), h(f(a), f(a)), \ldots\} \end{aligned}$$

Note in the last example that $h(a,a)$, for example, also belongs to $H(G)$, although h only occurs as $h(y,b)$ in G. To obtain a structure $\mathcal{A}$, we still need an interpretation. All terms are already defined (e.g. $a^\mathcal{A} = a, f^\mathcal{A}(a) = f(a)$ and so on). In principle we still need to define interpretation of the predicates. But in the sequel we proceed by considering all possible interpretations of some formula F. Explicitly this means that we will iterate over all instances of (the matrix of) F — for instance

$$P^\mathcal{A}(a, f(a), g(a,a)), P^\mathcal{A}(b, f(a), g(a,a)), P^\mathcal{A}(a, f(b), g(a,a)), P^\mathcal{A}(a, f(a), g(b,a)), \ldots$$

in the example above — and consider all these instances as atomic formulas, until we obtain a contradiction. Any structure $\mathcal{A} = (H(F), I_\mathcal{A})$ with $\mathcal{A} \models F$ is called a **Herbrand model** for F.

Theorem 2.27. *Let F be in Skolem normal form (or in TNF). Then F is satisfiable if and only if F has a Herbrand model.*

A proof is contained in Schöning. The proof is given for the Skolem normal form, for the TNF it is then clear (because the matrix of the formula is equivalent to its CNF).

Definition 2.28. Let $F = \forall y_1 \cdots \forall y_n F^*$ be a formula in Skolem normal form, and let F^* be the matrix of F. Let $E(F)$ be the set of all instances of F, that is:

$$E(F) = \{F^*[y_1/t_1][y_2/t_2]\cdots[y_n/t_n] \mid t_1, t_2, \ldots, t_n \in H(F)\}.$$

$E(F)$ is called **Herbrand expansion**.

Theorem 2.29. *For each formula F in Skolem normal form (or TNF) holds: F is satisfiable if and only if $E(F)$ is satisfiable (in the sense of propositional logic).*

Note that $E(F)$ is infinite, so this is the point where the Compactness Theorem (Thm. 1.29) is needed. Now we are able to provide a test that answers whether a given formula is unsatisfiable.

Algorithm 2.30. Input: a formula F in Skolem normal form such that the matrix of F has CNF. Let $E(F) = \{F_1, F_2, F_3, \ldots\}$ be an enumeration of the Herbrand expansion of F. Let $n = 0$, $M := \{\}$.

while $\square \notin M$ do

- $n := n + 1$
- $M := M \cup F_n$
- $M := \mathrm{Res}^*(M)$.

Return "unsatisfiable".

Note that this test runs forever if F is satisfiable. (Strictly speaking, it is not an algorithm in this sense).

Since $\neg F$ is unsatisfiable exactly when F is valid, this also provides a method for testing whether a given formula is valid (but not whether it is invalid, analogous to above). In the same way, we get a test for $F \models G$ (i.e. G is a consequence of F), see remark **??**.

Example 2.31. Consider $F = \forall x\,(P(x) \wedge \neg P(f(x))$. The matrix of F in clause set notation is

$$\{\{P(x)\}, \{\neg P(f(x))\}\}.$$

The Herbrand universe is $H(F) = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$. The Herbrand expansion of F therefore is $E(F) = \Big\{\{\{P(a)\}, \{\neg P(f(a))\}\}, \{\{P(f(a))\}, \{\neg P(f(f(a)))\}\}, \ldots\Big\}$. We get

$$\begin{array}{cccc} \{P(a)\} & \{\neg P(f(a))\} & \{P(f(a))\} & \{\neg P(f(f(a)))\} \\ & \diagdown \quad \diagup & & \\ & \square & & \end{array}$$

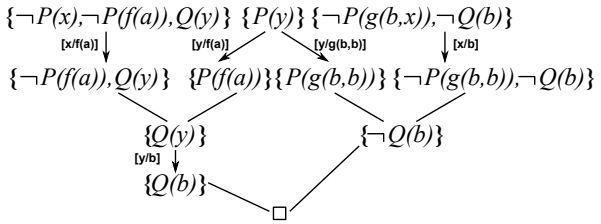
This shows that F is unsatisfiable.

In the example above we needed only two F_i, hence four clauses, and only one resolution step. Still, two of the clauses generated are superfluous. In bigger examples it is profitable to produce only clauses that are assumed to lead to $\square$ pretty quickly.

Example 2.32. Consider $F = \forall x\,\forall y\,\big((\neg P(x) \vee \neg P(f(a)) \vee Q(y)) \wedge P(y) \wedge (\neg P(g(b,x)) \vee \neg Q(b))\big)$. The matrix of F in clause set notation is

$$\{\{(\neg P(x), \neg P(f(a)), Q(y)\}, \{P(y)\}, \{\neg P(g(b,x)), \neg Q(b)\}\}$$

We use a more clever approach: start with the matrix of F and substitute variables in a prudent manner:



Hence F is unsatisfiable.

Further questions now may be:

1. How can we refine this algorithm in order to make it more efficient? E.g. using prudent substitutions, or delayed substitutions (like delaying $[y/b]$ in the example above)
2. Which classes of formulas are efficiently decidable? (compare Horn formulas)
3. Can we apply the resolution calculus to prove mathematical theorems automatically?

Remark 2.33. In order to illustrate the last one consider this formula:

$$F = \forall x\, \forall y\, \forall z\, f(x, f(y,z)) = f(f(x,y),z) \wedge \forall x\, f(x,e) = x \wedge \forall x\, \exists y\, f(x,y) = e$$

Each model for F is a group (see Maths I, or Wikipedia). Hence every consequence derived from F is a (true) statement about groups. If there is a machine (or algorithm) printing all true statement about groups a lot of mathematicians (namely, group theorists) would be unemployed. This is undergoing work, but it is not clear whether it will succeed. Right now the assumption is "no". One problem is that a machine cannot detect which result is "important" and which one is not. The tendency goes in direction "computer assisted proofs" and "computer aided proofs", see e.g. the very interesting survey Formal Proof by Thomas Hales. (Please note also the advertisement at the end of the paper.)

3 Modal logic

In some contexts it is desirable to generalize the notions of "true" and "false". E.g. "I am hungry" or "I wear a blue shirt" depends on the circumstances, e.g. on the time of the day or week. With respect to chronological order we may distinguish 10. Jan

- A is always true vs
- A is sometimes true

Or with respect to consequence we may distinguish

- A is necessarily true vs
- A is possibly true

For this purpose we introduce two new operators: $\Box$ and $\diamond$.

3.1 Syntax and semantics

The modal logic we discuss here is an extension of propositional logic. There is also a first-order modal logic, but it is not part of this course.

Definition 3.1 (Syntax)**.** A formula in modal logic is defined inductively as follows:

1. Each formula in propositional logic is a formula in modal logic.
2. If F and G are formulas in modal logic, then $\neg F$, $F \vee G$, $F \wedge G$, $\Box F$ ("necessary F") and $\diamond F$ ("possibly F") are formulas in modal logic.

For instance $\diamond A \Rightarrow \neg\Box(B \wedge \diamond\neg C)$ is a formula (in modal logic; in the remainder of this section "formula" means always "formula in modal logic" if not explicitly stated that not).

In order not to use too many brackets we agree that $\diamond$ and $\Box$ are stronger than $\wedge$ and $\vee$ (hence stronger than $\Rightarrow$ and $\Leftrightarrow$).

Example 3.2. Three football experts make three statements:

1. "If Schalke wins the championship ever again then I will eat my hat."
2. "At some time your statement will become true."
3. "Statement 2 being correct is the same as saying that Schalke will always be champion from now on."

If A="Schalke will be champion" and B="expert 1 will eat his hat" then 1. can be stated as $\diamond A \Rightarrow \diamond B$, 2. becomes $\diamond(\diamond A \Rightarrow \diamond B)$, and 3. becomes $\diamond(\diamond A \Rightarrow \diamond B) \Leftrightarrow \Box A$. (It is not clear yet whether this statements are true, or even make sense.)

In order to define the semantics of modal logic we need to specify a collection W of different states ("worlds") in which a formula holds, or does not hold. For each state in W there are further points in W that can be reached from this state ("possible futures"), others not (e.g. "the past", or "impossible futures"). This is realised as follows.

Definition 3.3 (Semantics)**.** A **structure** (in modal logic) is a triple $\mathcal{A} = (W, R, \alpha)$, where

- (W, R) is pair such that W is a nonemtpy set and R is a relation on W (therefore is $R \subset W \times W$, compare the text before definition **??**
- Let M be a set of atomic formulas. A map $\alpha : M \times W \to \{0,1\}$ is a **valuation** of M. In plain words: α assigns to each pair (A,s) — where $A \in M$ and $s \in W$ — true or false.

The pair (W,R) is called a **frame** for F. (W,R) can be viewed as a *directed graph*, see below.

Now we can define truth values for a formula F inductively as follows.

1. Is F an atomic formula A then $\mathcal{A}(F,s) = \alpha(A,s)$.
2. If F is a formula then $\mathcal{A}(\neg F, s) = \begin{cases} 1 & \text{if } \mathcal{A}(F,s) = 0 \\ 0 & \text{else} \end{cases}$

3. If F and G are formulas then $\mathcal{A}(F \wedge G, s) = \begin{cases} 1 & \text{if } \mathcal{A}(F,s) = \mathcal{A}(G,s) = 1 \\ 0 & \text{else} \end{cases}$

4. If F and G are formulas then $\mathcal{A}(F \vee G, s) = \begin{cases} 1 & \text{if } \mathcal{A}(F,s) = 1 \text{ or } \mathcal{A}(G,s) = 1 \\ 0 & \text{else} \end{cases}$

5. Is F a formula then $\mathcal{A}(\Box F, s) = \begin{cases} 1 & \text{if } \mathcal{A}(F,t) = 1 \text{ for all } t \in W \text{ with } (s,t) \in R \\ 0 & \text{else} \end{cases}$

6. Is F a formula then $\mathcal{A}(\diamond F, s) = \begin{cases} 1 & \text{if } \mathcal{A}(F,t) = 1 \text{ for some } t \in W \text{ with } (s,t) \in R \\ 0 & \text{else} \end{cases}$

As always we implicitly assume that $\mathcal{A}$, resp. M, contains all atomic formulas of F.

Example 3.4. (Ex. 3.6 cont.) A possible structure for $F = \diamond(\diamond A \Rightarrow \diamond B) \Leftrightarrow \Box A$ is, for example, $\mathcal{A} = (W, R, \alpha)$, where $W = \mathbb{N}_0$ (i.e. 0 stands for the current championship, 1 for the next championship, etc.), the relation R is $<$ (i.e. $R = \{(n,m) \mid n,m \in \mathbb{N}_0, n < m)\}$). An valuation α that makes F true is

$$\alpha(A,s) = 1 \text{ for all } s \in \mathbb{N}_0, \quad \alpha(B,s) = 1 \text{ for } s \text{ even}, \quad \alpha(B,s) = 0 \text{ otherwise}$$

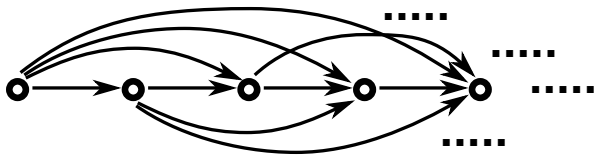
(”Schalke will always be champions from now on‘’, ”Expert 1 eats a broom in season broom in season 0, 2, 4, 6...”)

Another structure for F — with the same frame — is $\mathcal{A}' = (W, R, \beta) = (\mathbb{N}_0, <, \beta)$, with

$$\beta(A,s) = 1 \text{ for } s = 2,\ \beta(A,s) = 0 \text{ else}, \quad \beta(B,s) = 1 \text{ for } s = 3,\ \beta(B,s) = 0 \text{ else}$$

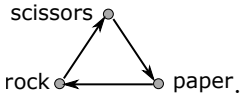
In this case $\mathcal{A}'(\diamond(\diamond A \Rightarrow \diamond B), s) = \mathcal{A}'(\diamond(\neg\diamond A \vee \diamond B), s) = 1$ for all $s \geq 2$ (since $\mathcal{A}'(\diamond A, s) = 0$ for $s \geq 2$, thus $\mathcal{A}'(\neg\diamond A, s) = 1$ for $s \geq 2$, thus $\mathcal{A}'(\diamond(\neg\diamond A \vee \diamond B), t) = 1$ for all t). On the other hand, $\mathcal{A}'(\Box A, s) = 0$ for all s. Therefore, $\mathcal{A}'(F, s) = 0$ for all $s \geq 2$ in this structure.

Example 3.5. A frame (W, R) can be visualized as a **directed graph**. The elements of W are the vertices of the graph, and an element $(n,m) \in R$ yields a directed edge from n to m. For instance the graph for the relation in the example above, i.e. for $(W, R) = (\mathbb{N}_0, <)$ looks in part as follows:



In particular the graph above is an infinite graph. Moreover, from each vertex infinitely many edges are leaving. A simpler example are the days of the week $\{1, 2, \ldots, 7\}$, ordered with respect to $<$.

Another simple example is the game rock-paper-scissors. Here $W = \{\text{rock, paper, scissors}\}$, and $R = \{(n,m) \mid n,m \in W,\ n \text{ beats } m\} = \{\text{(rock, scissors), (scissors, paper), (paper, rock)}\}$. The corresponding graph is simply



Example 3.6. (Ex. 4.1 cont.) A possible structure for $F = \diamond(\diamond A \Rightarrow \diamond B) \Leftrightarrow \Box A$ is $\mathcal{A} = (W, R, \alpha)$, where $W = \mathbb{N}$ (0 stands for the current championship 2022/23, 1 for the next championship and so on), the accessibility relation R is $<$ (that is, $R = \{(n,m) \mid n,m \in \mathbb{N}, n < m)\}$). A valuation α that makes F true is

$$\alpha(A,s) = 1 \text{ for all } s \in \mathbb{N}, \quad \alpha(B,s) = 1 \text{ for } s \text{ odd}, \quad \alpha(B,s) = 0 \text{ else.}$$

("Schalke will always be champion from now on", "expert 1 eats his hat in season 2022/23, 2024/25, 206/27...")

Another structure for F — using the same frame — is $\mathcal{A}' = (\mathbb{N}_0, <, \beta)$ is for instance given by

$$\beta(A,s) = 1 \text{ for } s = 2,\ \beta(A,s) = 0 \text{ else }, \quad \beta(B,s) = 1 \text{ for } s = 3,\ \beta(B,s) = 0 \text{ else.}$$

In this case $\mathcal{A}'(\diamond(\diamond A \Rightarrow \diamond B), s) = \mathcal{A}'(\diamond(\neg\diamond A \vee \diamond B), s) = 1$ for all $s \geq 2$ (since $\mathcal{A}'(\diamond A, s) = 0$ for $s \geq 2$, hence $\mathcal{A}'(\neg\diamond A, s) = 1$ for $s \geq 2$, hence $\mathcal{A}'(\diamond(\neg\diamond A \vee \diamond B), t) = 1$ for all t). On the other hand we have $\mathcal{A}'(\Box A, s) = 0$ for all s. Hence $\mathcal{A}'(F) = 0$ in this structure.

In particular the truth of a formula F depends on the point s in which we evaluate F. Therefore we will adjust the terms "model" and "valid" now to the context of modal logic.

The option to choose different frames for a given formula F, as well as different valuations for each frame, leads to a diversity of options for F "being true". The following definition lists all possibilities.

Definition 3.7. Let F be a formula in modal logic and $\mathcal{A} = (W, R, \alpha)$ a structure for F.

- If $\mathcal{A}(F,s) = 1$, then F **holds** in s. (One writes short $s \Vdash_{\mathcal{A}} F$, or $s \Vdash F$ if $\mathcal{A}$ is clear from the context). If there is $\mathcal{A} = (W, R, \alpha)$ and $s \in W$ with $s \Vdash_{\mathcal{A}} F$ then F is **satisfiable**.
- If $\mathcal{A}(F,s) = 1$ for all $s \in W$ we call $\mathcal{A}$ a **model** for F, short: $\mathcal{A} \models F$.
- Let a frame (W,R) be given. If for all structures $\mathcal{A} = (W, R, \alpha)$ holds $\mathcal{A} \models F$ then F is called **valid** in (W,R). In this case we write shortly $(W,R) \models F$.
- F is a **tautology** if F is valid in each frame for F.

With respect to Example 3.6 above: with $\mathcal{A} = (\mathbb{N}_0, <, \alpha)$ we have that $\mathcal{A}(F,s) = 1$ for all $s \in \mathbb{N}_0$, hence $\mathcal{A}$ is a model for F, short: $\mathcal{A} \models F$.

With $\mathcal{A}' = (\mathbb{N}_0, <, \beta)$ we saw that $\mathcal{A}'(F,s) = 0$ for all s. Hence $\mathcal{A}' \not\models F$ (but $\mathcal{A}' \models \neg F$). Therefore we also see $(N_0, <) \not\models F$, that is, F is not valid in $(\mathbb{N}_0, <)$. Because of this, F is in particular no tautology.

$\mathcal{A}$ and s given, $\mathcal{A}(F,s) = 1$	$s \Vdash_{\mathcal{A}} F$	F holds in s
There is $\mathcal{A}, s$ with $\mathcal{A}(F,s) = 1$	—	F is satisfiable
$\mathcal{A}$ given, for all $s \in W:\ \mathcal{A}(F,s) = 1$	$\mathcal{A} \models F$	$\mathcal{A}$ is model for F
(W,R) given, for all $\alpha, s \in W:\ \mathcal{A}(F,s) = 1$	$(W,R) \models F$	$\mathcal{A}$ is valid (under (W,R))
For all $\mathcal{A} = (W,R,\alpha), s \in W:\ \mathcal{A}(F,s) = 1$	$\models F$	F is a tautology.

Table 3: An overview of how "true" a formula F in modal logic can be, depending on which data out of $\mathcal{A} = (W,R,\alpha), s \in W$ is given (compare Def. 3.7).

$\mathcal{A}$ and s given, $\mathcal{A}(F,s)=1$	$s \Vdash_{\mathcal{A}} F$	F *holds* in s
There exist $\mathcal{A}, s$ with $\mathcal{A}(F,s)=1$	—	F is *satisfiable*
$\mathcal{A}$ given, for all $s \in W:\ \mathcal{A}(F,s)=1$	$\mathcal{A} \models F$	$\mathcal{A}$ is *model* for F
(W,R) given, for all $\alpha, s \in W:\ \mathcal{A}(F,s)=1$	$(W,R) \models F$	F is *valid* (in (W,R))
For all $\mathcal{A}=(W,R,\alpha), s \in W:\ \mathcal{A}(F,s)=1$	$\models F$	F is a *tautology*.

Table 4: An overview of the ways in which a formula F can be "true" in modal logic, depending on what exactly is defined in $\mathcal{A}=(W,R,\alpha), s \in W$ (cf. def. 3.7).

3.2 Calculation rules and (no) normal forms

As in Sections 1 and 2 we also have some rules of calculation for modal logic. In order to state these we need to define equivalence ($F \equiv G$) and consequence ($F \models G$). 10. Jan

Remark 3.8. Like in Lemma 1.32 we have that $F \models G$ if and only if $F \Rightarrow G$ is a tautology if and only if $F \wedge \neg G$ is unsatisfiable.
Furthermore, the following applies again: $F \equiv G$ if and only if $F \models G$ and $G \models F$ if and only if $F \Leftrightarrow G$ is a tautology.

Theorem 3.9. *Let F, G be two formulas. In addition to all calculation rules for formulas in propositional logic (see Theorem 1.10) the following rules apply:*

1. $\neg\Box F \equiv \Diamond\neg F$
2. $\neg\Diamond F \equiv \Box\neg F$
3. $\Box(F \Rightarrow G) \ \models\ \Box F \Rightarrow \Box G$
4. $\Box(F \Rightarrow G) \ \models\ \Diamond F \Rightarrow \Diamond G$
5. $\Diamond(F \Rightarrow G) \equiv \Box F \Rightarrow \Diamond G$
6. $\Box(F \wedge G) \equiv \Box F \wedge \Box G$
7. $\Diamond(F \vee G) \equiv \Diamond F \vee \Diamond G$
8. *If F is valid then $\Box F$ is valid.*

All statements can be proven by using the inductive definition of truth values. For instance, Rule 1 can be seen as follows: 17. Jan

$$\begin{aligned}
\mathcal{A}(\neg\Box F,t)=1 &\text{ if and only if } \mathcal{A}(\Box F,t)=0\\
&\text{precisely when not for all } s \text{ with } (t,s)\in R \text{ holds}:\ \mathcal{A}(F,s)=1\\
&\text{exactly when there is a } s \text{ with } (t,s)\in R \text{ and } \mathcal{A}(F,s)\neq 1\\
&\text{exactly when there is a } s \text{ with } (t,s)\in R \text{ and } \mathcal{A}(\neg F,s)=1\\
&\text{if and only if } \mathcal{A}(\Diamond\neg F,t)=1.
\end{aligned}$$

More on this in the exercises.

Recall: because of the definitions of $\Diamond$ and $\Box$, the truth value of a formula F depends on 18.1 Jan

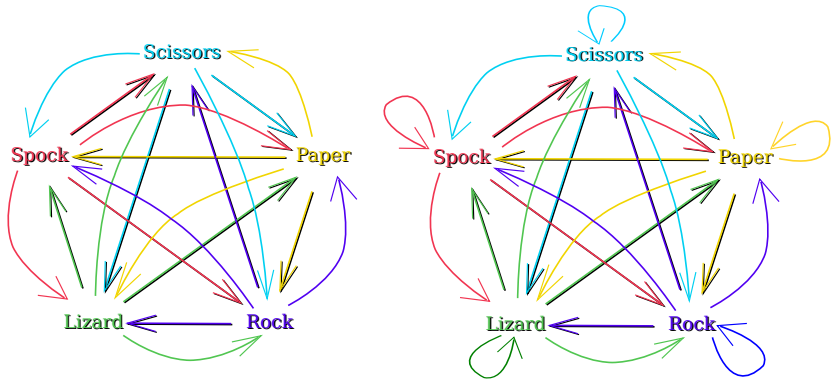
- the frame (W,R) in which we consider F, and
- the particular reference point $s \in W$ in which we evaluate $\mathcal{A}(F,s)$,
- as well as on all points that are accessible from s,
- and of α, of course.

Assume for now we have fixed some frame (W,R). The following result answers how far we need to look from s on, to be able to decide "true" or "false". The following definitions allows us to measure "far" in W with respect to F.

Definition 3.10. Let F be a formula, and (W,R) a frame for F.
A point $t \in W$ is **accessible from** s **in** n **steps**, if there are $t_0, t_1, \ldots, t_n \in W$ such that $s = t_0$, $t = t_n$, and $(t_i, t_{i+1}) \in W$ for $i = 0, 1, \ldots, n-1$. The n-th iterate of R is

$$R^n := \{(s,t) \in W \times W \mid t \text{ is accessible from } s \text{ in } \leq n \text{ steps.}\}$$

Example 3.11. (Example 3.5 cont.) The graphs of (W,R^2) and (W,R^3) look as follows (they are almost identical: all distinct points are accessible in two steps from each other, only a path from a vertex x back to x requires 3 steps):



Note that the graphs for (W,R^n) in the first example are all equal to the graph of (W,R) (since the relation $<$ is *transitive*). More examples are found on the exercise sheets.

Definition 3.12. Let F be a formula. The **modal rank** $\mathrm{MR}(F)$ of F is defined inductively as follows.

- If F is an atomic formula, then $\mathrm{MR}(F)$=0.
- If $F = \neg G$, then $\mathrm{MR}(F)$=$\mathrm{MR}(G)$.
- If $F = G \wedge H$ or $F = G \vee H$, then $\mathrm{MR}(F)$=$\max\{\mathrm{MR}(G),\mathrm{MR}(H)\}$.
- If $F = \Box G$ or $F = \Diamond G$, then $\mathrm{MR}(F)$=$\mathrm{MR}(G)$+1.

For example, for $F = \Diamond(\Diamond A \Rightarrow \Diamond B) \Leftrightarrow \Box A$ the modal rank is $\mathrm{MR}(F) = 2$.

Theorem 3.13 (coincidence lemma)**.** *Let F be a formula with $MR(F) = m$, let (W,R) be a frame for F and let $s \in W$. Furthermore, let $\mathcal{A} = (W,R,\alpha)$ and $\mathcal{A}' = (W,R,\beta)$ be two structures for F that are identical on all t that can be reached from s in at most m steps. Then $\mathcal{A}(F,s) = 1$ if and only if $\mathcal{A}'(F,s) = 1$.*

In plain language this means: Whether a formula F is true under $\mathcal{A}$ at point s depends only on the value of $\mathcal{A}$ at those points $t \in W$ that can be reached from s in at most $\mathrm{MR}(F)$ steps.

In reference to the above example 3.6, this means that if we want to determine the value of $\mathcal{A}(A,s)$, i.e. $\alpha(A,s)$, then we must consider all points in the diagram that can be reached in a maximum of two steps. can be reached. Unfortunately, this means that we have to consider all years in the future, since every year in the future can be reached in just one step is achievable: $(n,m) \in R$ whenever $n < m$.

Remark 3.14. There are normal forms for formulas in modal logic. The procedure is analoguous to Algorithm 1.19 together with pulling $\Box$s and $\diamond$s directly in front of the literals (analogous to dragging the $\neg$ in front of the literals). However, since we do not have both analogs of deMorgan's rules here, but only one each (see theorem 3.9 rules 6. and 7.), there is no clean normal form, only a "quasi" CNF. This generally consists of nested CNFs (a CNF within a CNF within a CNF within..., for more information see the book by Kreuzer-Kühling). We will therefore not do this here, as we don't need normal forms in the following, as we will use the tableau calculus.

3.3 Tableau calculus for modal logic

In the following, we will extend the tableau calculus from section 1.8 to modal logic. First, we present the main result of this section. In plain language, it says that the satisfiability of formulas in modal logic is always decidable.

Theorem 3.15. *A formula F in modal logic is unsatisfiable if and only if the completed tableau is closed. The number of nodes in each tableau for F is at most $O(m)$, where m is the number of subformulas of F.*

For proof, see Kreuzer-Kühling.

Note that the number of subformulas of a formula of length n (n symbols of any type, such as $\neg$, $\Box$, A, $\vee$...) is $O(n)$ in the worst case.

The following algorithm is an extension of 1.44. For the sake of completeness, we list all old and new rules here. This is the point at which the notation $s \Vdash F$ is practical. (This is is the abbreviation for $\mathcal{A}(F,s) = 1$).

In the following, "path" always means a path downwards (towards the leaves), either from the current node or from the root.

Algorithm 3.1 (Table calculation for modal logic) Input: a formula F in modal logic.

1. Start with $s \Vdash F$ as the root.

2. Choose an unmarked vertex $u \Vdash G$ where G is not a literal (in the sense of propositional logic). Mark $u \Vdash G$. Apply the following rules until all possibilities are exhausted.

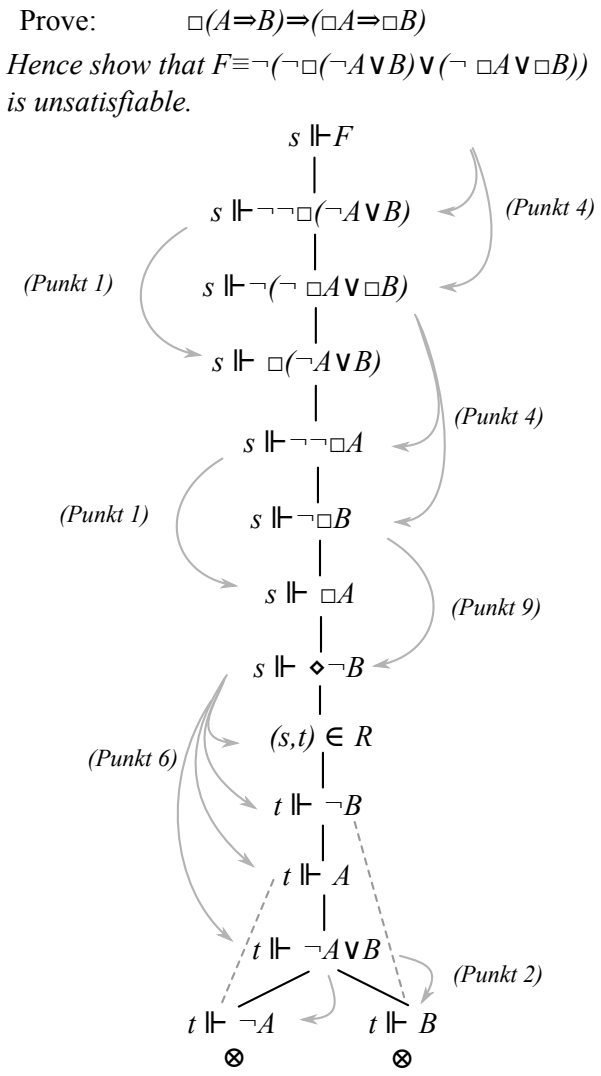
   - If G is of the form $u \Vdash \neg\neg H$ then add a single vertex $u \Vdash H$ to each path starting in $u \Vdash G$.
   - If G is of the form $H_1 \vee H_2$ then add $\begin{array}{ccc} & \diagup\ \diagdown & \\ u \Vdash H_2 & & u \Vdash H_1 \end{array}$ to each path starting in $u \Vdash G$.
   - If G is of the form $H_1 \wedge H_2$ then add $\begin{array}{c} \mid \\ u \Vdash H_1 \\ \mid \\ u \Vdash H_2 \end{array}$ to each path starting in $u \Vdash G$.
   - If G is of the form $\neg(H_1 \vee H_2)$ then add $\begin{array}{c} \mid \\ u \Vdash \neg H_1 \\ \mid \\ u \Vdash \neg H_2 \end{array}$ to each path starting in $u \Vdash G$.
   - If G is of the form $\neg(H_1 \wedge H_2)$ then add $\begin{array}{ccc} & \diagup\ \diagdown & \\ u \Vdash \neg H_2 & & u \Vdash \neg H_1 \end{array}$ to each path starting in $u \Vdash G$.
   - If G is of the form $\Diamond H$ then add $\begin{array}{c} \mid \\ (u,t) \in R \\ \mid \\ t \Vdash H \end{array}$ to each path starting in $u \Vdash G$, where $t \in W$ is a point not occurring in the tableau yet. For every vertex $u \Vdash \Box H'$ in the path from the root through $u \Vdash G$ add $\begin{array}{c} \mid \\ t \Vdash H' \end{array}$ at each path starting in $u \Vdash \Box H'$ that contains $u \Vdash G$.
   - If G is of the form $\Box H$ then add to each path starting in $u \Vdash G$, where $t_1, \ldots, t_k$ $\begin{array}{c} \mid \\ t_i \Vdash H \end{array}$ are the points that occur in the form $(u,t_i) \in R$ contained in this path.
   - If G is of the form $\neg\Diamond H$ then add $\begin{array}{c} \mid \\ u \Vdash \Box\neg H \end{array}$ to each path starting in $u \Vdash G$.
   - If G is of the form $\neg\Box H$ then add $\begin{array}{c} \mid \\ u \Vdash \Diamond\neg H \end{array}$ to each path starting in $u \Vdash G$.

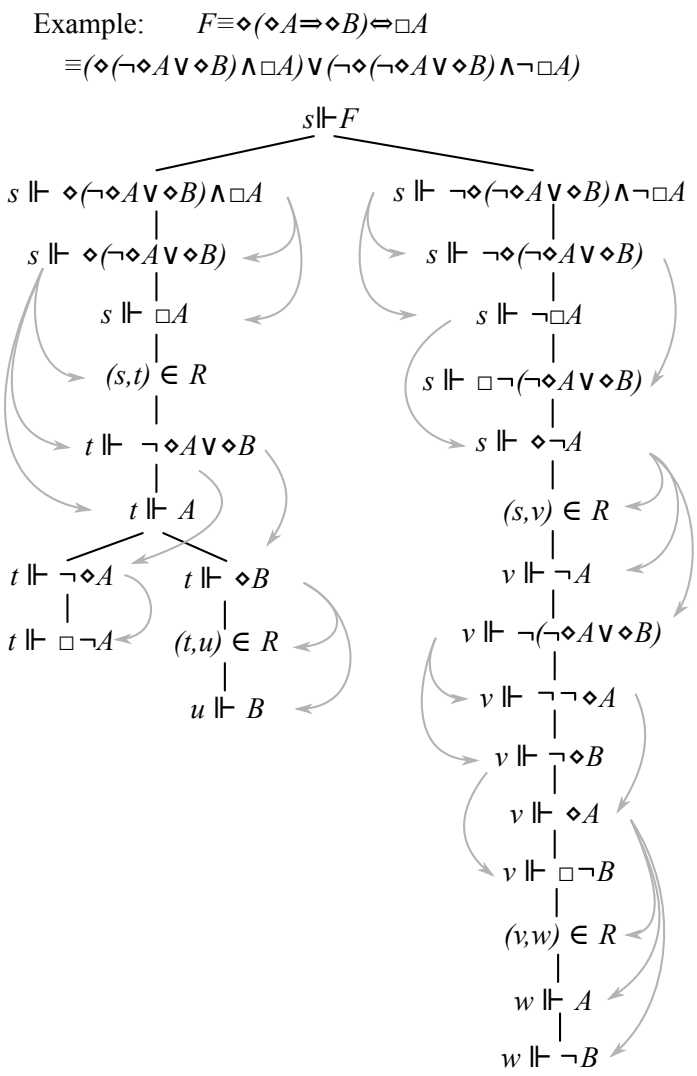
3. If there is no further vertex to mark then return the tableau, STOP.

A path is **closed** if it contains $u \Vdash A_i$ and $u \Vdash \neg A_i$ for some i. In this case we mark the leaf of this path by $\otimes$. A tableau is **closed** if all leaves are marked by $\otimes$. In this case F is unsatisfiable. Else F is satisfiable.

The $\Box$ rule and the $\Diamond$ rule make this tableau calculus much trickier than the one for propositional logic. Please note in particular: With the $\Box$ rule, sometimes there is nothing to do at all, for example if there is no $(u,t) \in R$ node yet. The second part of the $\Diamond$ rule is basically the $\Box$ rule, because: If nothing has been done with a $\Box$ rule, but a $(u,t) \in R$ node appears later for which the $\Box$ rule would have applied, this second part ensures that this is done later.

Next we show two examples: First we prove Rule 3 of Theorem 3.9 by showing that $\Box(F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G)$ is a tautology; hence by showing that $\neg\big(\Box(F \Rightarrow G) \Rightarrow (\Box F \Rightarrow \Box G)\big)$ is unsatisfiable. (Recall that we amy as well use A, B rather than F, G, see Example 1.46). Then we show that our master example $\diamond(\diamond A \Rightarrow \diamond B) \Leftrightarrow \Box A$ is satisfiable.





3.4 Different flavours of modal logic

In general frames can have pretty general forms, see above or the exercises for examples. A lot of work has been dedicated to study modal logic under certain restrictions for frames. Three such properties that we already know are

- For all $s \in W$ applies: $(s,s) \in R$ (reflexive)
- For all $s,t \in W$ applies: $(s,t) \in R \Rightarrow (t,s) \in R$ (symmetrical)
- For all $s,t,u \in W$ applies: $\big((s,t) \in R \wedge (t,u) \in R\big) \Rightarrow (s,u) \in R$ (transitive)

Requiring one or more of these may make additional rules become true that are false under modal logic in general. For example, $F \models \diamond F$ is true if (W,R) is reflexive (see also the exercises). Because:

It holds $s \Vdash A \Rightarrow \diamond A$ if and only if $s \Vdash \neg A \vee s \Vdash \diamond A$ is a tautology. The latter means: $s \Vdash \neg A$ holds, or there is t with $(s,t) \in R$ and $t \Vdash A$. Since R is reflexive, with $t = s$ yes $s,s) \in R$, and the statement becomes $s \Vdash \neg A$ or $s \Vdash A$, i.e. $s \Vdash \neg A \vee A$. This is obviously a tautology.

In general the following flavours of modal logic are studied in detail:

- K No conditions on (W,R)
- D serial, i.e. $\forall s \in W\ \exists t \in W :\ (s,t) \in R$ (no dead ends)
- T reflexive
- S4 reflexive and transitive
- S5 reflexive, symmetric and transitive

It is easy to see that a hierarchy applies to these five points: each of the logics contains all those below it. (Because, for example, every relation that is reflexive *and* transitive is also reflexive; or because serial follows from reflexive).

A further particular application of modal logic uses time as a frame. This is **temporal logic**: it uses the same elements as modal logic, but for the frame (W,R) one requires additionally that it is transitive and *irreflexive*: $\forall s \in W :\ (s,s) \notin R$. So a frame $(\mathbb{N}_0, <)$ realizes temporal logic.

The tableau calculus from above (i.e. also theorem refthm:modalperfect) applies to modal logic K. For the others you need a tableau calculus with more rules; or you no longer get a perfect test, but only "if all paths are closed, then then unsatisfiable".

It is known that the modal logics K, T, S4, S5 and temporal logic are all decidable, even though the tableau algorithm in the form above might not terminate in all cases (e.g. in temporal logic the tableau calculus will not terminate in all cases).

Interlude: Infinite cardinalities

20. Dec

For a finite set M it is clear how to define the number of its elements: the set $M = \{2,3,5,7\}$ has four elements, for instance. This number is called the **cardinality** of M, denoted by $|M|$ (sometimes also $\#M$ or $\text{card}(M)$). If M has infinitely many elements it becomes more tricky. Then the following concept becomes helpful.

Definition 3.16. Let M,N be two sets. It holds $|M| = |N|$ (speaking: M and N have the same same **cardinality** if there is a bijective mapping from M to N exists.
It holds $|M| \leq |N|$ if there is an injective mapping from M to N exists.

Recall:
A map $f : M \to N$ is called *bijective* if for each $m \in M$ there is exactly one n in N such that $f(m) = n$. Think of a mass wedding: M consists of a bunch of men, N consists of a bunch of women. Then a bijective f assigns to each man exactly one woman and vice versa. Everyone gets married. For finite sets M and N this is clearly possible only if they have the same number of elements.

A map $f : M \to N$ is called *injective* if for each $m \in M$ there is at most one n in N such that $f(m) = n$. Regarding the mass wedding: Every woman gets at most one man, possibly none. But no woman gets two or more men.

We can now extend the notion of cardinality to infinite sets like $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$..., or to their power sets.

Definition 3.17. The **power set** of some set M is the set of all its subsets, including M and $\varnothing$. It is denoted by $\mathcal{P}(M)$.

For instance, the power set of $\{1,2,3\}$ is

$$\mathcal{P}(\{1,2,3\}) = \big\{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\big\}$$

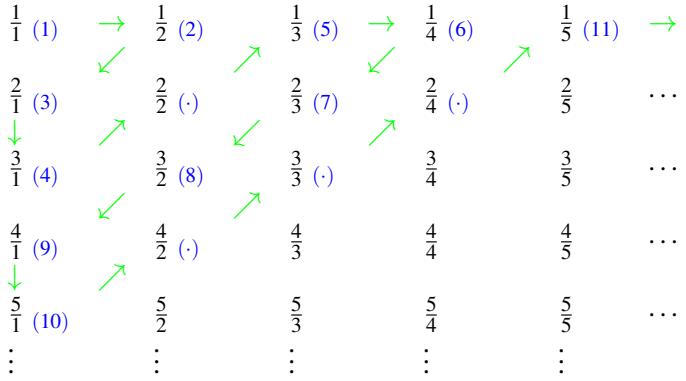
The first maybe surprising fact is the following. (Surprising, since for finite sets it is not possible that a proper subset of M has the same cardinality as M.)

Theorem 3.18. $|\mathbb{N}| = |\mathbb{Q}|$.

Proof. We need to construct a bijective map f from $\mathbb{Q}$ to $\mathbb{N}$. Write all elements of $\mathbb{Q}$ in an infinite two-dimensional array:

$$\begin{array}{ccccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} & \cdots \\
\frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} & \frac{3}{5} & \cdots \\
\frac{4}{1} & \frac{4}{2} & \frac{4}{3} & \frac{4}{4} & \frac{4}{5} & \cdots \\
\frac{5}{1} & \frac{5}{2} & \frac{5}{3} & \frac{5}{4} & \frac{5}{5} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots &
\end{array}$$

It is clear that this array contains each positive element of $\mathbb{Q}$. Some of them occur in multiple copies (e.g. $\frac{1}{1} = \frac{2}{2} = \frac{3}{3} = \cdots$) Now walk through this array in the way indicated here in the following diagram, numbering each element of $\mathbb{Q}$ with consecutive integer numbers that is not numbered yet (e.g., don't mark $\frac{2}{2}, \frac{3}{3}, \ldots$ since $\frac{1}{1}$ is numbered already).



It is clear that each number in $\mathbb{Q}$ in the array will be marked: it will be reached after finitely many steps. This yields a bijection between $\mathbb{N}$ and the positive rationals $\mathbb{Q}^+$:

$$\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & \cdots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
1 & \frac{1}{2} & 2 & 3 & \frac{1}{3} & \frac{1}{4} & \frac{2}{3} & \frac{3}{2} & 4 & 5 & \frac{1}{5} & \cdots
\end{array}$$

Insert the 0 at the beginning, and after each positive rational number its negative:

$$\begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & \cdots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & -1 & \frac{1}{2} & -\frac{1}{2} & 2 & -2 & 3 & -3 & \frac{1}{3} & -\frac{1}{3} & \frac{1}{4} & -\frac{1}{4} & \frac{2}{3} & -\frac{2}{3} & \cdots \end{array}.$$

This is the desired bijective map, hence $|\mathbb{N}| = |\mathbb{Q}|$. □

This proof shows a general trick: To show $M| = |\mathbb{N}|$, it is sufficient to number all elements in M with $1,2,3,\ldots$ so that each element in M gets exactly one number $n \in \mathbb{N}$. A second, somewhat surprising fact is this:

Theorem 3.19 (Cantor 1874)**.** $|\mathbb{R}| > |\mathbb{N}|$.

The following proof is not the original one, but one found by Cantor in 1891.

Proof. by contradiction. We show that there is no bijection from $\mathbb{N}$ into the half-open interval $[0;1[$ (i.e., all numbers x such that $0 \leq x < 1$). If we succeed this shows that that there is no bijection between $\mathbb{R}$ and $\mathbb{N}$, since $\mathbb{R}$ is even larger than $[0,1[$.

So let us assume there is a bijection from $\mathbb{N}$ into $[0;1[$. Then we can write the elements $a_1, a_2, \ldots$ of $[0;1[$ in a list. We use the decimal representation $0.a_{i,1}a_{i,2}a_{i,3}\ldots$.

$$\begin{array}{llllllll} a_1 = 0. & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \ldots \\ a_2 = 0. & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \ldots \\ a_3 = 0. & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \ldots \\ a_4 = 0. & a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \ldots \\ a_5 = 0. & a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \ldots \\ & & & \cdots & & & \end{array}$$

If there is an ambiguity (like $0.09999\cdots = 0.1$) we choose the finite version, ending in ...0000..., and omit the infinite version, ending in ...99999...

By our assumption this list contains all elements of $[0,1[$. We will now find a contradiction by constructing an element that is not in the list: define $s = 0.s_1s_2s_3\cdots$ by

$$s_i = \begin{cases} 7 & \text{if } a_{ii} = 3 \\ 3 & \text{if } a_{ii} \neq 3 \end{cases}$$

This number s is clearly in $[0,1[$, and it does not end in ...99999... (since its digits are all either 3 or 7). The first digit of s after the period differs from the first digit of a_1 (by construction: if the first digit of a_1 is 3, then the first digit of s is 7; and if the first digit of a_1 is not 3, then the first digit of s is 3). In a similar fashion, the ith digit of s after the period differs from the ith digit of a_i.

Hence s is not equal to any number a_i in the list. □

The last result leads to the following terminology.

Definition 3.20. A set M with $|M| = |\mathbb{N}|$ is called **countable** set. A set M with $|M| > |\mathbb{N}|$ is called **uncountable** set.

Now that we know that there are different levels of "infinitely many", can we describe this hierarchy? Again Georg Cantor can help.

Theorem 3.21. *Let M be a set. Then $|\mathcal{P}(M)| > |M|$.*

For a finite set $M = \{m_1, m_2, \dots, \}$ this is a simple observation: $\mathcal{P}(M)$ contains at least $\{m_1\}, \{m_2\}, \dots$, but also $\{m_1, m_2\}$, or M. Note that $|\varnothing| = 0$, but

$$|\mathcal{P}(\varnothing)| = |\{\varnothing\}| = 1.$$

In particular the theorem implies $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$, $|\mathcal{P}(\mathcal{P}(\mathbb{N}))| > |\mathcal{P}(\mathbb{N})|$ etc. There is a notation for these growing infinite cardinalities.

Definition 3.22. Let $\beth_0 := |\mathbb{N}|$. If $|M| = \beth_n$ for $n \geq 0$, let $\beth_{n+1} := |\mathcal{P}(M)|$.

Remark 3.23. One can show that $|\mathbb{R}| = |\mathcal{P}(\mathbb{N})| = \beth_1$.

Usually the simplest way to show such equalities is the Schröder-Bernstein Theorem.

Theorem 3.24. *Let A and B be sets. If there exist injective functions $f : A \to B$ and $g : B \to A$ between the sets A and B, then $|A| = |B|$.*

The $\beth$ ("beth") is the second letter in the Hebrew alphabet. It is (was) a big question whether these $\beth_n$s are the *only* infinite numbers, or whether there are more infinite numbers in between. In particular the continuum hypothesis asks whether there is some infinite cardinality between $\beth_0$ and $\beth_1$. Click the link for more information (Wikipedia). The Wikipedia article uses terms like "independent from ZFC" etc. These are explained in Section 5.

4 Undecidability

24. Jan

A celebrated result in the area of undecidability is due to the legendary (and tragic) figure of Kurt Gödel. There are two distinct senses of the word "undecidable" in the context of logic, or computability. The first of these is used in relation to computability theory. It does not apply to individual statements ("this sentence is true"), but to decision problems that have (countably) *infinitely many* inputs (e.g. "Is n a prime number?" for $n \in \mathbb{N}$), each of which must be answered with yes or no. More precisely: we call such a problem **not computable** if there is no computer program (more precisely: no Turing machine) that answers the question correctly with "Yes" or "No" for each n.

The second sense of this term is the sense used in relation to Gödel's theorems, that of a statement being neither provable nor refutable in a specified deductive system. Here we use the word "decidable" in the first sense only.

4.1 Undecidable problems

Definition 4.1. A **decision problem** is a yes-no question with infinitely many possible inputs. Formally it can be modeled as a pair (I, M) where I is the set of all possible inputs, and $M \subset I$ is the set of inputs with answer "yes".

A problem is **decidable** (aka computable, berechenbar, entscheidbar) if there is a Turing machine that answers correctly "yes" or "no" on any input. (If you don't know what a Turing machine is, replace "'Turing machine"' by "'algorithm"' and think of a haskell or python algorithm.) A problem is **semi-decidable** (aka semi-computable) if there is a Turing machine that answers correctly "yes" for every input with answer "yes", and does not answer "yes" for any input with answer "no".

Usually the inputs can be enumerated, hence in several cases one has $I = \mathbb{N}$ or $I = \mathbb{N}^k$ and $M \subset \mathbb{N}$. Then one can state the same question using a function $f : \mathbb{N}^k \to \{0,1\}$, with $f(n) = 1$ if $n \in M$, $f(n) = 0$ else. The corresponding names for f are then **recursive** (corr. to decidable=computable) and **recursively enumerable** (corr. to semi-decidable).

A famous problem that is undecidable is the question "does Turing machine number i ever stop on input j?" This question is known as the halting problem. Since there are countably infinitely many Turing machines we may enumerate all Turing machines by numbers in $\mathbb{N}$. All inputs (finite starting configurations on the tape) can be enumerated as well (by some appropriate scheme).

Theorem 4.2 (Turing 1937)**.** *The halting problem is undecidable.*

Proof. Phrased as a function the problem asks now for a *computable* $f : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ where $f(i,j) = 1$ if Turing machine i stops on input j, $f(i,j) = 0$ else. Assume that f is computable. Consider

$$g : \mathbb{N} \to \{0,1,\text{undefined}\}, \quad g(i) = \begin{cases} 0 & \text{if } f(i,i) = 0 \\ \text{undefined} & \text{else} \end{cases}$$

For instance, g can be realized as a Turing machine (or as a program):

$$\text{If } f(i,i) = 0 \text{ then return 0 else run forever}$$

Clearly g is computable if f is. Hence there is a Turing machine t_g that computes g. What is $f(t_g,t_g)$?

Case 0: If $f(t_g,t_g) = 0$ then $g(t_g) = 0$. In particular (since the Turing machine t_g computes g) the t_g stops on input t_g, and $f(t_g,t_g) = 1$. Contradiction.

Case 1: If $f(t_g,t_g) = 1$ then $g(t_g)$ is not defined, hence t_g does not stop, thus $f(t,t) = 0$. Contradiction.

The assumption that f is computable leads to a contradiction in all (two) cases, hence f is not computable. □

Theorem 4.3 (Turing 1937)**.** *The halting problem is semi-decidable. I.e. there is a Turing machine that returns* 1 *on input* (i,j) *if Turing machine i stops on input j, and returns nothing if not.*

Proof. The rough idea is: just let a given Turing machine i run on input j. If it stops return 1. As usually we rely on the results in Theoretische Informatik and formulate just an algorithm without implementing it as an actual Turing machine.

All you have to do is run all possible Turing machines with all possible inputs one after the other in a sensible sequence. (or more precisely: to simulate them. This is the "universal" Turing machine). To do this, we (somehow) calculate the k-th step for the Turing machine i with input j in this order:

$$(i,j,k) = (1,1,1),(1,1,2),(1,2,1),(2,1,1),(1,1,3),(1,2,2),(1,3,1),(2,1,2),(2,2,1),(3,1,1),(1,1,4),\ldots$$

In plain words: list all triples (i,j,k) with $i,j,k \in \mathbb{N} \setminus \{0\}$ in ascending order with respect to $i+j+k$ and then with respect to lexicographical order. It is clear that all possibilities in $\mathbb{N}^3$ are covered in finite time (!).

Our universal Turing machine therefore takes (i,j) as input, simulates at some point every k-th step of the Turing machine i with input j, and returns 1 when the Turing machine i finally stops. □

Some undecidable problems The following problems are undecidable (usually, semi-decidable)

1. Is a given formula in first-order logic valid? (Church 1936, Turing 1937)

Theorem 4.4 (Church 1936, Turing 1937)**.** *The problem whether a formula in first-order logic is satisfiable is undecidable.*
Since F is unsatisfiable if and only $\neg F$ is valid, the problem whether a formula in first-order logic is valid is undecidable as well.

One strategy of the proof is the following (the original proof was different):

1. Show that: If the problem above is decidable then the *Post correspondence problem* below is decidable.
2. Show that: If the Post correspondence problem is decidable then the halting problem is decidable.

Part I can be found in Schöning's book, part II in Sipser's book (both in the bibliography at the end of this script). This yields a contradiction to Theorem 4.2. Hence the problem above must be undecidable. It remains to show 1 and 2.

2. The **Post correspondence problem (PCP)**: The input of the problem consists of two finite lists $u_1,\ldots,u_n$ and $v_1,\ldots,v_n$ of words over the alphabet $\{0,1\}$. A solution to this problem is a sequence of indices $i_1,i_2,\ldots,i_k$ with $k \geq 1$ such that

$$u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}.$$

Here u_1u_2 just means concatenation: if $u_1 = 10$ and $u_2 = 01$ then $u_1u_2 = 1001$. The decision version of the PCP problem then is to decide whether such a solution exists or not.

A nice illustration of this problem is: assume we have finitely many distinct domino tiles, with some 0-1-words on top and on bottom. With an arbitrary large amount of copies of these, can we place them in a row such that the top row shows the same string as the bottom row?

For instance assume that we have $u_1 = 1, u_2 = 10, u_3 = 011$, and $v_1 = 101, v_2 = 00, v_3 = 11$. The tiles thus look as follows:

$$\left[\begin{smallmatrix}1\\101\end{smallmatrix}\right], \quad \left[\begin{smallmatrix}10\\00\end{smallmatrix}\right], \quad \left[\begin{smallmatrix}011\\11\end{smallmatrix}\right].$$

A solution is $(1,3,2,3)$, that is,

$$\left[\begin{smallmatrix}1\\101\end{smallmatrix}\right]\left[\begin{smallmatrix}011\\11\end{smallmatrix}\right]\left[\begin{smallmatrix}10\\00\end{smallmatrix}\right]\left[\begin{smallmatrix}011\\11\end{smallmatrix}\right].$$

The word in the top row, as well as in the bottom row, is 101110011.

3. The **Wang tile problem**: Given a set of squares whose edges are colored with different colors ("Wang tiles"). Can we pave the plane with copies of these tiles so that adjoining edges have the same

color? “Paving the layer” means placing the tiles so that they do not overlap and leave no gaps. The tiles should always be placed corner to corner. It is not permitted to rotate or mirror the tiles. (Again, it is not a requirement that all types of tiles must be used).

Translated with DeepL.com (free version)

For instance, the set of 11 tiles below can tile the plane (but it is veeeeery tricky to see how)

No subset of 10 tiles can tile the plane. In general this problem is undecidable for more than three colours and more than 10 tiles.

4. **Mortal matrix problem**: Given some matrices $A_1, \ldots, A_n$ in $\mathbb{Z}^{d\times d}$. Is there any combination of them (say, $(i_1, i_2, \ldots, i_m)$ with $1 \leq i_j \leq n$) such that the product equals the zero matrix? That is, is there $i_1, \ldots, i_m$ such that

$$A_{i_1} \cdot A_{i_2} \cdots A_{i_m} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix} ?$$

This problem is undecidable for $n \geq 6$.

5. **Conway’s Game of Life** (see Wikipedia): given some start pattern and another pattern, can the latter ever be the result of the first one?

6. **Diophantine equations:** Given a polynomial with several variables, but integer coefficients, does it have an integer solution? (Diophantine means we are only interested in integer solutions). For example, the Diophantine equation $3x^2 - 2xy - y^2z - 7 = 0$ has an integer solution: $x = 1,\ y = 2,\ z = -2$. By contrast, the Diophantine equation $x^2 + 4y^2 - 3 = 0$ has no such solution (only non-integer ones, like $x = 1, y = \frac{\sqrt{2}}{2}$, or $x = \sqrt{2}, y = \frac{1}{2}$).

In a similar manner as the halting problem all problems above can be shown to be semi-decidable.

4.2 Computable numbers

The paper in which Turing proved Theorem 4.4 also introduced Turing machines. It was titled ”On computable numbers with an application to the Entscheidungsproblem”. Turing defined a number to be computable as follows: all integers are computable. A number $x \notin \mathbb{Z}$ is computable if there is a Turing machine that computes the n-th digit of its fractional part (”die n-te Nachkommastelle”) on input n. This is in essence equivalent to the modern definition:

Definition 4.5. A number $x \in \mathbb{R}$ is **computable** if for each $\varepsilon \in \mathbb{Q}, \varepsilon > 0$ there is a computable function $f : \mathbb{Q} \to \mathbb{Q}$ such that $|x - f(\varepsilon)| < \varepsilon$.
The set of all computable numbers is denoted by $\mathbb{B}$.

The modern definition solves the problem of a Turing machine approximating the number 2, for instance, by computing n digits of $1,9999999\ldots$. It holds $2 = 1,9999999\ldots$. But the seventh digit of 2 is 0, not 9. The definition above repairs this problem. Turing already realized that not all numbers are computable. The argument is simple: There are only countably many Turing machines (because

each Turing machine can be encoded as some finite 0-1-word) hence there are at most countably many computable numbers. Since $\mathbb{R}$ is uncountable:

Almost all numbers in $\mathbb{R}$ are not computable

Nevertheless, Turing already showed that

1. All rational numbers are computable. In other words/symbols: $\mathbb{Q} \subset \mathbb{B}$
2. All algebraic numbers are computable (this is, the roots of some polynomial with integer coefficients).
3. π and e are computable.
4. The limit of a computable sequence that is computably convergent is computable.

In order to explain point 4: It is not true that the limit of any convergent sequence of any computable number is computable. We need to make sure that: (a) the sequence itself is computable, and (b) the fact that it converges is computable.

Definition 4.6. A sequence of computable numbers a_n is called **computably convergent** if there is a computable function $N : \mathbb{B} \to \mathbb{N}$ such that for all $\varepsilon \in \mathbb{B}, \varepsilon > 0$ and for all $m, n \in \mathbb{N}$ such that $m > N(\varepsilon)$ and $n > N(\varepsilon)$ holds $|a_n - a_m| < \varepsilon$.

Compare this definition with the definition of a Cauchy sequence.

Now point 3 ensures that

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots) \quad \text{and} \quad e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

are computable. Moreover, point 4 ensures that all algebraic numbers are computable, because there are efficient methods to approximate roots of polynomials. In fact, almost *all* numbers we can describe in words are computable. Exceptions are constructed as follows:

A non-computable number Consider the halting problem for Turing machines. Let us assume we did agree already on an enumeration of all Turing machines. Let $0 < a < 1$ such that the nth digit of a is 0 if Turing machine number n stops on input 0, and 1 else. It is clear that $a \in \mathbb{R}$. If a would be computable we would have a solution to the halting problem. This contradicts Theorem 4.3.

4.3 Consequences II

We are now in a position to understand Gödel's results precisely (and not just by means of superficial illustrations). Let us recall the two types of inference, but now put them in a different framework (namely, there are a few formulas that we take to be true).

Definition 4.7. A **formal system** is a pair $(\mathcal{F}, S)$, where $\mathcal{F}$ is a set of **axioms** (formulas that we assume to be true, they are our starting point) and a set S of rules for drawing syntactic consequences (like resolution, or modus ponens, or both together).

A formula G is a **semantic consequence** of F, if for all $\mathcal{A}$ we have that whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$. (see definition 1.31). In this case we write shortly $F \models G$.

A formula G is a **syntactic consequence** of F, if we can deduce G from F using the rules S. In this case we write shortly $F \vdash G$ (or $F \vdash_S G$, if we want to emphasize that we use rules of S). See also the text in note 1.35).

In the sequel we think of $\mathcal{F}$ usually as a set of formulas in first-order logic. One gets an idea of "using S" by thinking of resolutions: resolutions are consequences, see Lemma 1.39.

One of Turing's achievements was that he gave us a better understanding of what "drawing consequences" can mean: namely, using a Turing machine. Today this is clear: using a computer. But programmable computers had not yet been invented at that time. If you like, Turing invented his machine in order to concretize the term "syntactic inference".

For propositional logic there is no difference between the two kinds of consequences: everything that is true in propositional logic can be proven in propositional logic. ("'Proven"' means: deduced by using truth tables, or resolution, or...) Moreover, everything that can be proven in propositional logic is true in propositional logic.

Now we are prepared to formulate Gödel's famous results in a precise way.

4.4 Gödel's completeness theorem

To better understand the incompleteness theorems, it is worth looking at another of Gödel's celebrated results. To do this, we must first establish a meaningful calculus so that the $\vdash$ is meaningfully explained. Gödel's original proof proved something like "there is a meaningful calculus $\vdash$ such that..." In a more modern view, one says: $\vdash$ is given by a Hilbert calculus. This is (to my knowledge) not sharply defined, but the idea is: there are a few axioms and as few inference rules as possible. (In contrast to *systems of natural inference*, which manage without axioms). For a concrete example, see the Wikipadia "Hilbert system". (Once again: the internet knows little about formal logic. The list "Hilbert systems" in the English Wikipedia only lists those for propositional logic.)

Theorem 4.8. *Let $\mathcal{F}$ be a set of formulas in first-order logic, and G be a formula.*

$$\mathcal{F} \models G \text{ implies } \mathcal{F} \vdash G$$

Of course, we want $\vdash$ to be correct (see def. 1.35). Then "$\mathcal{F} \vdash G$ implies $\mathcal{F} \models G$" is *per definitionem* fulfilled, and the theorem becomes "$\mathcal{F} \models G$ if and only if $\mathcal{F} \vdash G$". Very naively one might be confused by "everything can be proven" (completeness theorem) versus "not everything can be proven" (incompleteness theorems). A more careful formulation is this one: "All valid theorems can be proven" (completeness theorem) versus "There are theorems that are neither true nor false" (incompleteness theorem). This is a clumsy interpretation, but one might get some better idea about the difference.

Indeed, by Definition 4.7 $\mathcal{F} \models G$ means: whenever $\mathcal{A} \models \mathcal{F}$ then $\mathcal{A} \models G$ (G is a semantic consequence.) And $\mathcal{F} \vdash G$ means that we can deduce G from $\mathcal{F}$ in the formal system. So as well as in propositional logic and in modal logic, also for first order logic holds:

Everything that is true in first-order logic can be proven in first-order logic.

But the next section shows that not every statement in first-order logic is either true or false (in the sense of "is semantic consequence from the axioms")

4.5 Gödel's incompleteness theorems

Gödel's first inclompeteness theorem states that any formal system $(\mathcal{F}, S)$ cannot have all four of the following properties:

1. $(\mathcal{F}, S)$ is recursively enumerable,
2. $(\mathcal{F}, S)$ contains, or implies, integer arithmetic,
3. $(\mathcal{F}, S)$ is negation-complete,
4. $(\mathcal{F}, S)$ is consistent.

Number 1. means that the set of all syntactic consequences (from $\mathcal{F}$ using S) is recursively enumerable (see Section 4.1), and that $\mathcal{F}$ (and S) themselves are. I.e., the problem whether any given G is a syntactic consequence of $\mathcal{F}$ using S is semi-decidable, compare Definition 4.1.

Number 2. see below.

Number 3.: **Negation complete** means, that for each F (expressible in the language of $\mathcal{F}$) either $\mathcal{F} \vdash F$ or $\mathcal{F} \vdash \neg F$. (This is a *different* meaning of "complete" than in theorem 4.8). That's why we call it differently here. (In the literature or on wikipedia, unfortunately, often the two meanings are both referred to as "complete").

Number 4: **Consistent** means that there is no F (expressible in the language of $\mathcal{F}$), so that both $\mathcal{F} \vdash F$ and $\mathcal{F} \vdash \neg F$. We have already seen and used in Section 1, see Remark 1.35.

Number 2. means that $(\mathcal{F}, S)$ is rich enough to contain the axioms for the natural numbers $\mathbb{N}$ together with the rules for addition and multiplication. One way to achieve this in first-order logic are the **Peano axioms**. Usually this requires the usage of infinitely many formulas in first-order logic. One possibility uses the identity, one constant (0), one function with one argument (S, "successor", i.e. $S(x) = x+1$) and two functions with two arguments: $f(x,y)$ (to be understood as +) and $g(x,y)$ (to be understood as $x \cdot y$). There are six particular formulas stated as axioms:

1.	$\forall x\, 0 \neq S(x)$	(read: $0 \neq x+1$)
2.	$\forall x \forall y\, S(x) = S(y) \Rightarrow x = y$	(read: $x+1 = y+1 \Rightarrow x = y$)
3.	$\forall x\, f(x,0) = x$	(read: $x+0 = x$)
4.	$\forall x \forall y\, f(x,S(y)) = S(f(x,y))$	(read: $x+(y+1) = (x+y)+1$)
5.	$\forall x\, g(x,0) = 0$	(read: $x \cdot 0 = 0$)
6.	$\forall x \forall y\, g(x,S(y)) = f(g(x,y),x)$	(read: $x \cdot (y+1) = x \cdot y + x$)

Furthermore, for every predicate P with $k+1$ arguments we add the formula

$$\forall y_1, \ldots, y_k \left(\left(P(0, y_1, \ldots, y_k) \wedge \forall x \left(P(x, y_1, \ldots, y_k) \Rightarrow P(S(x), y_1, \ldots, y_k) \right) \right) \Rightarrow \forall x\, P(x, y_1, \ldots, y_k) \right)$$

to the axioms. Hence there are countably infinitely many axioms altogether. These latter formulas realize the **principle of induction**.

There are several ways to formulate the next result. One common way is this.

Theorem 4.9 (Gödel's first incompleteness theorem, 1931)**.** *Any consistent and recursively enumerable formal system $(\mathcal{F}, S)$ that contains integer arithmetic is not negation-complete.*

In this form it means that there are formulas G (expressable in the language of $(\mathcal{F}, S)$) where neither $\mathcal{F} \vdash_S G$ nor $\mathcal{F} \vdash_S \neg G$ holds. Using the enumeration from the list above it says in this form "if we have 1,2 and 4 we cannot have 3". Of course there are now equivalent ways to state the same result:

"if we have 1,2, and 3 we cannot have 4"; or "if we have 2,3 and 4 we cannot have 1". It is helpful to consider an example for each instance.

Not negation-complete. This was the "classical" case, surprising experts like David Hilbert or John von Neumann: using the Peano axioms we cannot expect to prove or disprove any given statement as a syntactic consequence. There are some statements F that are neither "true" nor "false", at least not viewed from within the system. (Because otherwise we could prove it because of Theorem 4.8.) In fact, in such a case we could add F to the axioms without losing the freedom from contradiction. And we could as well add $\neg F$ to the axioms instead without losing freedom of contradiction (but not both, of course).

A famous example of such a statement is the continuum hypothesis in the Z ermelo-Frenkel axioms (for the latter, see section 5).

Not consistent. Take the Peano axioms above as our $\mathcal{F}$ and define some (very liberal) rule S that says "for every formula G we have $\mathcal{F} \vdash G$". Hence every formula is a consequence of $\mathcal{F}$ under S. This means in particular that $\mathcal{F} \vdash_S G$ and $\mathcal{F} \vdash_S \neg G$ for any G. This example is very far from being consistent.

Not recursively enumerable. It is a very different thing to draw consequences in a formal system (which can have several different models) in contrast to draw consequences in some particular concrete model $\mathcal{A}$. The definition of $\mathcal{A}$ allows us to decide whether $\mathcal{A} \models G$ or not. So let us assume the **true arithmetic**, that is, the structure $\mathcal{A}$ for the Peano axioms where $U^{\mathcal{A}} = \mathbb{N}$, the 0 symbol means the actual 0, $S^{\mathcal{A}}(x) = x+1$, $f^{\mathcal{A}}(x,y) = x+y$ and so on. (Each symbol means what it is intended to mean.) Now for every statement we can express in the language of the Peano axioms it can be decided (in principle[2]) whether it is true or false in this model. Moreover, no statement can be both true and false. So we have integer arithmetic (2), negation-completeness (3), and consistency (4). By Theorem 4.9 there is no formal system for this true integer arithmetic that is semi-computable.

Not containing integer arithmetic. It is an interesting (and deep) fact there are also negation-complete (and consistent) theories. A simple example is propositional logic. More sophisticated examples include **Presburger arithmetic** (integer artihmetic without multiplication), or the first-order theory of Euclidean geometry, or monadic first-order logic (all predicates have only one argument, compare exercises).

Theorem 4.10 (Gödel's second incompleteness theorem, 1931)**.** *The consistency of a consistent and recursively enumerable formal system $(\mathcal{F}, S)$ that contains integer arithmetic can be formulated as a formula G in $(\mathcal{F}, S)$, and $\mathcal{F} \nvdash_S G$.*

5 Zermelo-Fraenkel, axiom of choice and the Banach-Tarski paradox

In the beginning of the 20th century a lot of researchers tried to develop an axiomatization of the foundations of mathematics. A milestone was the book *Principia Mathematicae* by *Bertrand Russell* and *Alfred North Whitehead*, see for instance the interesting comic book *Logicomix*. It became clear that the most basic notion is that of a set. Once a set is properly defined one can easily define numbers, relations, functions etc.

[2]Some statements that eluded an answer so far are "is each even number larger than 2 the sum of two primes?", or "are there infinitely many twin primes?". (Twin primes are primes of the form p, $p+2$, like 11 and 13.) Nevertheless, these statements are either true or false in $(\mathbb{N}, +, \cdot)$.

For instance, once sets are defined, the natural numbers can be defined as follows:

$$0 = \{\}, \quad S(n) = n \cup \{n\},$$

where S is the successor function (think: "$n+1$"). This means the first natural numbers are $0 = \{\} = \varnothing, 1 = \{0\} = \{\varnothing\}, 2 = \{0,1\} = \{\varnothing, \{\varnothing\}\}, 3 = \{0,1,2\} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$.... Once the successor function S is defined we may define addition by

$$f(n,m) = n+m = S^n(m) = S(S(\cdots S(m)\cdots)$$

and multiplication by

$$n \cdot 0 = g(n,0) = 0, \quad n \cdot S(m) = g(n, S(m)) = S^n(g(n,m)).$$

The latter means for instance (note that $2 = S(1)$)

$$3 \cdot 2 = S^3(3 \cdot 1) = S^3(S^3(1 \cdot 0)) = S^3(S^3(0)) = 6$$

One can easily check that these definitions satisfy the Peano axioms.

Relations on a set W are just subsets of $W \times W$, see the previous sections. Functions $f : X \to Y$ are an valuation of elements $f(x) \in Y$ for each $x \in X$. And so on. It remains to define what a set is.

Earlier approaches to define sets were given in a non-formal way, e.g. Georg Cantor's *naïve set theory*. It went along the lines of "a set is a collection of objects".

By a set we mean any grouping of certain well-differentiated objects of our perception or our thinking into a whole.

Hence a set is everything that can be defined using language. Even though several aspects of Cantor's work are milestones of logic and mathematics (Cantor's diagonal argument, uncountable sets...) the insufficiency of a naive definition of sets was revealed by **Russell's paradox**:

Let R be the set of all sets that are not members of themselves. If R is not a member of itself, then its definition implies that it must contain itself. If R contains itself, then it contradicts its own definition. Symbolically:

$$\text{Let } R = \{x \mid x \notin x\}, \text{ then } R \in R \Leftrightarrow R \notin R \equiv \neg(R \in R)$$

Russell's paradox (also found earlier by Zermelo) and Hilbert's demand to find an axiomatization for all of mathematics lead people to develop an axiomatization of set theory.

Maybe the most standard definition today is called the **Zermelo-Fraenkel-axioms** (ZF), developed by Ernst Zermelo in 1908 and later improved by Fraenkel and Skolem in order to allow to prove certain concepts ("cardinal numbers") and avoid certain improperly defined terms. Nowadays these axioms can be (and usually are) stated in first-order logic. There are several equivalent formulations. One is the following:

5.1 Zermelo-Fraenkel axioms

ZF uses identity $=$ and one further binary predicate $P(x,y)$, namely "x is element of y", short (as usual) $x \in y$. Hence we will write $x = y$ and $x \in y$ in the sequel rather than $P(x,y)$. Note that this

allows us to define a further predicate "subsets": a set z is a subset of a set x if and only if every element of z is also an element of x: $z \subseteq x$ means: $\forall q\ (q \in z \Rightarrow q \in x)$.

In the sequel, keep in mind that the intended universe is "all sets": the axioms are tailored in a way such that each model behaves like its universe being sets.

Axioms number 1 to 5 are the "intuitive" axioms that we would expect from the properties of sets that we are used to. Numbers 6 to 9 are rather less intuitive, but needed in order to avoid certain problems (all axioms are accompanied by a short explanation of their respective meaning). This section makes strong use of the corresponding Wikipedia article, but it is not exactly the same.

1. Axiom of extensionality Two sets are equal (are the same set) if they have the same elements.

$$\forall x\, \forall y\, \big(\forall z(z \in x \Leftrightarrow z \in y) \Rightarrow x = y\big).$$

2. Empty set axiom There is a set without elements.

$$\exists x\, \neg \exists y\, y \in x$$

Notation: this x is called $\varnothing$.

3. Axiom of pairing If x and y are sets, then there exists a set which contains x and y as elements.

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

For instance, the pairing of $x = \{1,2\}$ and $y = \{2,3\}$ is $z = \{\{1,2\},\{2,3\}\}$.

4. Axiom of union The union over the elements of a set exists. More precisely, for any set of sets x there is a set y containing every element of every element of x.

$$\forall x\, \exists y\, \forall u\, \big(u \in y \Leftrightarrow (\exists v\, (v \in x \wedge u \in v))\big)$$

For example, the union over the elements of the elements of $x = \{\{1,2\},\{2,3\}\}$ is $y = \{1,2,3\}$.

5. Axiom of power set This axiom states that for any set x, there is a set y that contains every subset of x (compare Def. 3.17):

$$\forall x\, \exists y\, \forall z\, \big(z \subseteq x \Rightarrow z \in y\big).$$

Ensures the existence of the set of all subsets of x, the *power set* (notation: $\mathcal{P}(x)$) of any set x. For instance if $x = \{1,2,3\}$ then $\mathcal{P}(x) = \{\varnothing,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.

6. Axiom of specification Subsets are often denoted by something like $\{n \in \mathbb{N} \mid n \text{ prime}\}$, or $\{n \in \mathbb{N} \mid n \equiv 2 \bmod 5\}$, or so. This axiom ensures that such subsets always exist.

Let F be any formula in the language of ZF with some free variable x (y is not free in F). Then:

$$\forall z\, \exists y\, \forall x\, \big(x \in y \Leftrightarrow (x \in z \wedge F)\big).$$

Since this axiom is of the form "for any F" it is in fact an infinite collection of axioms in first-order logic (compare the Peano axiom of induction).

This axiom ensures that we can write $y = \{n \in \mathbb{N} \mid n \text{ prime}\}$ using $z = \mathbb{N}$ and $F = P(n) =$"n is prime".

This axiom prevents the "set" R of the Russell paradox from being a set!

Let us assume the set M of all sets is a set. Let $P(x) = x \notin x = \neg(x \in x)$. Then by this axiom $R = \{x \in M \mid P(x)\}$ is a set. But we know that it cannot be a set, since its existence leads to a contradiction. Hence our assumption is wrong and the set of all sets does not belong to our universe, that is, M is not a set (and neither is R).

7. Axiom of replacement The axiom of replacement asserts that the image of a set under any definable function will also fall inside a set.

Formally, let F be a formula in the language of ZF whose free variables are x and y. Then:

$$\forall x\, \exists y\, \forall u \big(u \in y \Leftrightarrow \exists z\, (z \in x \wedge F(z,u))\big).$$

Spelled out this means: For each set x exists a set y consisting of all elements u for which there is $z \in x$ such that $F(z,u)$ holds.

In particular this means that the image of a set under some function f is again a set: choose $F(z,u)$ as $u = f(z)$. Then y contains all u such that $f(z) = u$, where z takes all values in x.

Again this axiom is of the form "for any F", hence it is in fact an infinite collection of axioms in first-order logic (compare the Peano axiom of induction).

8. Axiom of infinity There are infinite sets. More precisely:

$$\exists x\, \big(\varnothing \in x \wedge \forall y\, (y \in x \Rightarrow y \cup \{y\} \in x)\big).$$

This axiom ensures the existence of infinite sets like the natural numbers in the set theoretic definition above: $\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}, \cdots,\}$. Together with the power set axiom it ensures also the existence of uncountably infinite sets.

9. Axiom of foundation Also known as "axiom of regularity". Every non-empty set x contains a member y such that x and y are disjoint sets.

$$\forall x\, \big(x \neq \varnothing \Rightarrow \exists y\, (y \in x \wedge \neg \exists z\, (z \in y \wedge z \in x))\big)$$

This implies, for example, that no set is an element of itself. More generally, it prevents the existence of cyclic chains of subsets: $x_1 \subset x_2 \subset \cdots \subset x_n \subset x_1$.

5.2 Axiom of choice

Now every model for the ZF-axioms has the properties that we require for the notion of "sets", without leading to some non-desired effects (as mentioned above: e.g. a set cannot be its own element, the set of all sets is not a set, ...). Still it was found that one rule is missing:

Axiom of choice For any set x of nonempty sets x_i there is a function that chooses one element from each x_i.

$$\forall x\big(\varnothing \notin x \Rightarrow \exists f\colon x \to \bigcup x \quad \forall u \in x\ f(u) \in u\big).$$

It seems rather intuitive that such a rule must indeed be true: For instance, given the set

$$\{\{1,4,5\},\{12,31,44,77\},\{101,202,303\}\}$$

we can choose one element from each set, for instance 1, 12, 101. Sometimes the axiom of choice is illustrated as follows:

Assume you are the king of n provinces with finitely many citizens each. You need to choose a governour for each province. How do you proceed? Simple: just choose the oldest citizen to be governour.

What if you have infinitely many provinces with finitely many citizens each? The same procedure works. What if you have infinitely many provinces with infinitely many citizens each? There may not longer be an oldest citizen... can you still formulate a general rule for this case?

Now consider an even worse case: you are supposed to choose one element from each subset of the real numbers. There is no known rule how to achieve this. But the axiom of choice states that such a rule exists. Even though it seems so intuitive this rule does not follow form the ZF axioms 1-9:

Theorem 5.1 (Gödel 1940, Cohen 1963)**.** *The axiom of choice is not a consequence of the Zermelo-Fraenkel axioms. Neither is its negation a consequence of the Zermelo-Fraenkel axioms.*

Hence nowadays **ZFC**, that is, ZF together with the axiom of choice, are regarded as the (best known?) standard axiomatization of set theory.

Given the ZF axioms there are several equivalent formulations of the axiom of choice. Here are two of them:

Well-ordering Theorem For any set X there is a linear order, that is: there is a relation $\leq$ on X that is

- antisymmetric; that is if $a \leq b$ and $b \leq a$ then $a = b$.
- transitive; that is if $a \leq b$ and $b \leq c$ then $a \leq c$.
- total; that is $a \leq b$ or $b \leq a$ for all a, b.

Intuitively one may doubt that this can be true: think of the set $\mathbb{C}$, or of $\mathbb{R}^2$. How can there possibly be a well defined relation like $\leq$?

Zorn's Lemma Let $(M, \leq)$ be a partially ordered set (i.e. $(M, \leq)$ is reflexive, antisymmetric and transitive). If every subset of M that has a linear order has an upper bound in M then the set M contains at least one maximal element.

The fact that the axiom of choice (resp. its equivalent formulations) are controversial (see Wikipedia) is reflected in the following quotation:

The Axiom of Choice is obviously true, the well-ordering principle obviously false, and who can tell about Zorn's lemma?

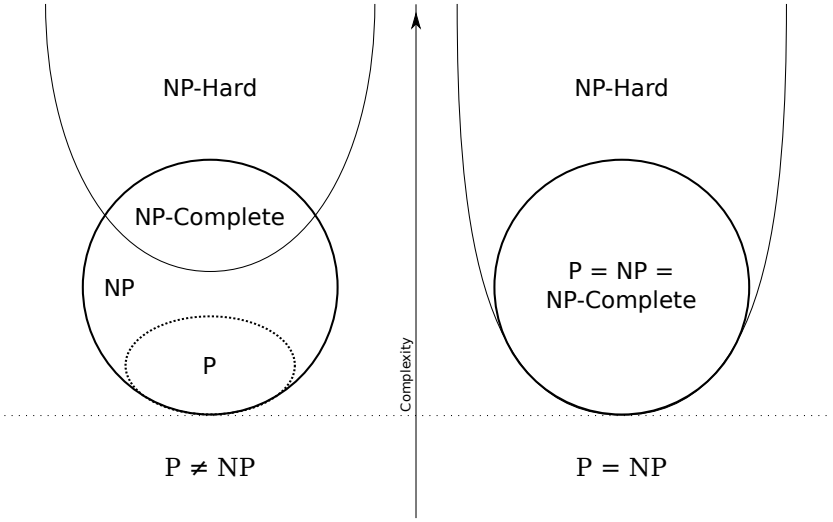


Figure 2: The situation if P≠NP (left) and if P=NP (right).

Complexity

In Theoretical Computer Science there are several levels of complexity of a (class of) problem(s): P, NP,... The question whether a formula in first-order logic is satisfiable is harder than any of them. Let us briefly sketch this hierarchy. A problem is here a problem that can have infinitely many, and arbitrary large, inputs, like an integer, or a graph, or a tree, or a list of integers. The size of the input is measured by some reasonable measure, like the number of the digits of an integer, or the number of vertices of a graph or tree, or the number of the entries of some list.

For some purposes the problem is required to have only the answers 'yes' or 'no'.

P A problem is in P if there is an algorithm that solves the problem for any input of size n in polynomial time; i.e., the number of the required steps is in $O(p(n))$ where p is some polynomial (or in general, something that is smaller than some polynomial). E.g. sorting a list of n numbers can be done in $O(n^2)$ steps (quicksort), or even in $O(n \log n)$ (mergesort). Finding the minimal number in a list of integers can be done trivially in time $O(n)$. The problem PRIME, i.e., testing whether a number x with n binary digits is a prime number ('yes' or 'no') requires naively $O(2^{n/2})$ steps (test all odd numbers between 3 and $\sqrt{x}$ if they divide x). If we couldn't do better problem this would not be in P, but there are algorithms known that need $O(n^{12})$ steps. Hence PRIME is in P.

NP A problem is in NP if each correct solution (aka "witness", or "certificate") of it can be checked in polynomial time (with respect to the size n of the input). For instance, the question "does a given graph G with n vertices have a Hamiltonian cycle" is in NP. (A graph has a Hamiltonian cycle if it contains a cycle that contains each vertex of G exactly once.) In this example a solution is such a Hamiltonian cycle, and it is easily checked whether a given solution is correct. On the other hand, finding a Hamiltonian cycle can be pretty hard in general.

Similarly, the prime factorization of integers with n digits is in NP: it might be hard to tell what the

prime factors of 1003 are, or of 1007 (are they prime numbers or composite numbers?) But a witness for the factorization of 1007 is $(19,53)$, since $19 \cdot 53 = 1007$. (And $1003 = 17 \cdot 59$.)

It is a big open problem whether P=NP. It is generally assumed that NP is a bigger class than P. A problem is called **NP-hard** if a solution of this problem can be translated into a solution for any problem in NP in polynomial time. A problem that is both in NP and NP-hard is called **NP-complete**. Figure 2 illustrates the situation (for both cases, P=NP and P$\neq$NP).

It is known that the Hamiltonian cycle problem is NP-complete, but prime factorization of integers is not. Usually such results are shown by reducing a problem to SAT, the question whether a formula in propositional logic in CNF is satisfiable, compare Remark 1.24. SAT was the first problem known to be NP-complete.

Beyond NP Are there problems that are even harder than problems in NP? Yes. For instance, a class of problems that is harder than NP problems is the class NEXP. This is the class of problems having instances where all witnesses are of exponential size. It is known that NP$\neq$NEXP (more precisely, NP is a proper subset of NEXP). The examples for this look somehow artificial. For instance, the question whether a non-deterministic Turing machine will stop after n steps is in NEXP, since each witness is already of size $O(2^n)$. In a similar manner, it is known that certain graphs with 2^n vertices can be encoded into logical circuits with $O(n)$ logic gates ("succinct circuits"). If one asks for a Hamiltonian cycle in such a graph this problem is in NEXP, since each witness is of size $O(2^n)$. In Section 4 we see undecidable problems. These are problems where *no* algorithm can answer each instance of the problem. At all. One is the question whether a given formula in first-order logic is satisfiable.

Literature

- Uwe Schöning: Logic for Computer Scientists *(covers most of Sections 1 and 2 in a very compact but comprehensive manner)*
- Uwe Schöning: Logik für Informatiker (same in German)
- Martin Kreuzer, Stefan Kühling: Logik für Informatiker (German) *(one of the very few textbooks covering modal logic)*
- H.-D. Ebbinghaus, J. Flum, W. Thomas: Mathematical Logic *(THE classic textbbok on formal logic, contains a lot more than this lecture)*
- Wolfgang Rautenberg: A Concise Introduction to Mathematical Logic *(another comprehensive textbook)*
- M. Sipser: Introduction to the theory of computation *(contains the complete proof of Theorem 4.4)*
- A.K. Doxiades, C.H. Papadimitriou, A. Papadatos: Logicomix *(Just for fun: a comic book telling the story of Russell (and how he met Gödel and Whitehead and Cantor...)*