Computational Logic

Moritz Müller

July 10, 2024

Logic is the calculus of computer science.

Contents

Preface

Logic

Modern mathematical logic emerged during the so-called foundational crisis of mathematics and Hilbert's program. From the very beginning work on this program tackled algorithmic questions, the most important of which was the question what an algorithm actually is. For example, Hilbert asked in 1928 whether there exists an algorithm deciding the problem

Entscheidung	
Input:	a sentence φ.
Problem:	is φ valid?

We explain these notions below. Maybe the most important aspect of this question is that the notion of *algorithm* had been informal at the time. Church and Turing formalized this notion in 1936 and gave a negative answer to Hilbert's question. Thereby logicians studied the reaches and limits of computers well before they had actually been built, including the today highly topical question concerning the possibility of AI. Thus, the historical roots of computer science lie in mathematical logic.

More importantly, logic continues to play an essential role in computer science. This has been repeatedly and prominently been pointed out.[1] In fact, "*Logic is for computer science what calculus is for physics*" is an often repeated slogan.[2] It has been said that computer science "*is a continuation of logic by other means*"[3] and even that it "*should be*

[1]M. Davis. Influences of mathematical logic on computer science. In: The universal Turing machine: a half-century survey (2nd ed.). Springer, pp. 289–299, 1995.

J. Y. Halpern, R. Harper, N. Immerman, P. G. Kolaitis, M. Vardi, V. Vianu. On the unusual effectiveness of logic in computer science. Bulletin of Symbolic Logic 7 (2): 213-236, 2001.

A. Blass. Symbioses between mathematical logic and computer science. Annals of Pure and Applied Logic167 (10): 868-878,2016.

[2]P. Kolaitis, M. Vardi. Logic as The Calculus of Computer Science. Talk at the NSF/CISE Workshop on The Unusual Effectiveness of Logic in Computer Science, National Science Foundation, Arlington, 2001. Available here: `https://www.cs.rice.edu/~vardi/logic/`

M. Genesereth, V. Chaudhri. Logic in Secondary School Education. Essay available at: `http://logic.stanford.edu/publications/genesereth/logic.pdf`

[3]G. Gottlob. Computer Science as the continuation of logic by other means. Talk at Simposio Internacional: El legado de Alan Turing, Madrid, 23-24-10-2012. Available here: `http://www.informatics-europe.org/images/ECSS/ECSS2009/slides/Gottlob.pdf`

viewed as a branch of applied logic."[4]

What is logic? Better: what is *a* logic? It is an artificial language, formally defined in order to avoid the usual ambiguities and vagueness of natural language. Viewed like this the essential role of logic for computer science should come as no surprise. The definition of a logic has three parts answering the questions:

1. How to talk?
2. What is truth?
3. How to reason?

Question 1 is answered by the definition of *syntax*: it is the definition of a set of formal objects called *sentences* (or *formulas*), often certain words over a certain alphabet. Usually sentences can be encoded by finite binary strings and can thus be processed by computers.

Question 2 is answered in two steps. First one decides what the sentences are supposed to talk about: one defines a collection of formal objects, let us call them *worlds*. Second one defines truth as a relation $\models$ between worlds W and sentences φ, usually denoted

$$W \models \varphi.$$

Such a definition automatically produces formal definitions of central semantic concepts:

– A sentence φ is *valid* if it is true in all worlds, i.e., $W \models \varphi$ for all worlds W.

– Two sentences φ, ψ are *logically equivalent* if they are true in the same worlds, i.e., for all worlds W: $W \models \varphi \iff W \models \psi$.

– A *theory* T (i.e., a set of sentences) *logically implies* a sentence φ if φ is true whenever T is true, i.e., $W \models \varphi$ for all worlds W with $W \models T$.

...

Question 3 is answered by defining a *calculus*. Usually a calculus is given by syntactically defined rules allowing to produce new sentences, namely *conclusions*, from given ones, namely *premisses*. Sequences of sentences produced by repeated rule applications are formal *proofs*. Being syntactically defined means that the applicability of a rule refers only to the syntactical form of the premisses and the conclusions, without reference to the semantics $\models$. Of course, this is key for the automatation of reasoning.

The holy grail of such a definitorial adventure is a

Completeness Theorem *T logically implies φ if and only if T proves φ.*

Two philosophical comments. First, *Kreisel's sandwich argument*: whatever your informal notion of "implies" is it typically seems safe to assume that T "implies" φ implies that

[4]stated by Leivant in: K. Bruce, P. Kolaitis, D. Leivant and M. Vardi. Panel: logic in the computer science curriculum. ACM SIGCSE Bulletin 30: 376–377, 1998.

T logically implies φ. It is also typically seems more than safe to assume that T "implies" φ is implied by T proves φ. By the completeness theorem it is thus safe to assume that T proves φ if and only if T "implies" φ. This is a significant philosophical insight: the informal notion of "implication" is captured by the formal notion of proof.

Second: it is an often repeated line of argument that AI is not "real intelligence" because computers have no access to the "meaning" of sentences but "merely manipulate symbols" (e.g. *Searle's room* in the analytical philosophy of mind). Such arguments are dubious in the presence of a completeness theorem.

Algorithms

Formal definitions of computations can be given e.g. via the notion of a *Turing machine*. This is outside the scope of this course but assumed as background knowledge of the reader. However, the course is accessible already from an intuitive understanding of computations as follows.

An *input* is any finitary object, say encoded by a finite binary string. A *computation* is a sequence of simple modifications of the input string ending with an *output* string. The *algorithm* is a sequence of instructions, say program lines, that given any input produces a computation on it. A *(decision) problem* is formalized a set Q of binary strings. It represents the intuitive problem

Q	
Input:	a binary string x.
Problem:	is x in Q?

For example, the ENTSCHEIDUNGSPROBLEM is formally viewed as the set of binary strings encoding valid sentences φ. An algorithm *decides* Q if, given as input a binary string x, produces a computation with output 1 if $x \in Q$, and 0 if $x \notin Q$.

A function $t : \mathbb{N} \to \mathbb{N}$ *bounds the runtime* of an algorithm if, on an input of length n, the computation has length $\leqslant t(n)$.

What is an efficient algorithm? In this course we follow the historical suggestion of Cobham and Edmonds (1965) and view an *efficient* algorithm as one with polynomially bounded runtime, that is, the function t above can be taken to be a polynomial.

We are interested only in asymptotic runtime bounds and use the O-notation. Saying that a function $f : \mathbb{N} \to \mathbb{N}$ is $O(g)$ for another function $g : \mathbb{N} \to \mathbb{N}$ means that there is $c \in \mathbb{N}$ such that $f(n) \leqslant cg(n) + c$ for all $n \in \mathbb{N}$. Dually, we say f is $\Omega(g)$ if g is $O(f)$, equivalently, there is $c \in \mathbb{N}$ such that $f(n) \geqslant \lfloor g(n)/c \rfloor$ for all $n \in \mathbb{N}$. Often we write e.g. 2^n or n^3 to denote g. E.g. f is bounded by a polynomial if and only if f is $O(n^c)$ for some $c \in \mathbb{N}$.

Chapter 1

Propositional logic

This chapter introduces propositional logic and closely follows the outline in the Preface. It defines syntax and semantics and then semantic concepts in the canonical way. It proceeds giving two calculi for formal reasoning, first Gentzen's sequent calculus, then Resolution.

There are extra sections. Section 1.5 showcases how propositional logic is used in computer science to model basic computational problems. Such modeling is preliminary to employ powerful SAT solvers in software engineering. Section 1.7 showcases how Resolution is used for algorithm analysis. Section 1.8 is advanced material on Resolution lower bounds, implying lower bounds on the runtime of certain SAT solvers.

1.1 Syntax

For $n \in \mathbb{N}$ write $[n] := \{1, \ldots, n\}$, understanding $[0] = \emptyset$.

An *alphabet* A is a non-empty set of *letters*. We write $A^* := \bigcup_{n \in \mathbb{N}} A^n$ and refer to its elements as *words (over A)*. For $n \in \mathbb{N}$ and $w = (a_1, \ldots, a_n) \in A^n$ we call $|w| := n$ the *length of w*; we omit parentheses and write $w = a_0 \cdots a_n$. We say a letter $a \in A$ *occurs in* w if $a = a_i$ for some $i \in [n]$, and call such i an *occurrence* of a. There is exactly on word of length 0, the *empty word* ϵ. Given two words $w = a_1 \cdots a_n$ and $w' = a'_1 \cdots a'_m$ of lengths n and m we write ww' for the word $a_1 \cdots a_n a'_1 \cdots a'_m$ of length $n + m$. A *(proper) prefix* of a word w is a word w' such that $w = w'w''$ for some (non-empty) word w''.

Example 1.1.1. For $A := \{0, 1\}$, the set $A^* = \{0, 1\}^*$ is the set of *binary strings*.

Definition 1.1.2 (Syntax)**.** Let $\mathit{Var} := \{X_0, X_1, \ldots\}$ be a set of (pairwise distinct) *propositional variables*. The set of *(propositional) formulas (* is the smallest set F of words over the alphabet

$$\wedge,\ \neg,\ (,\ ),\ X_0, X_1, \ldots$$

satisfying for all words φ, ψ:

(F1) $\mathit{Var} \subseteq F$;

(F2) if $\varphi \in F$, then $\neg\varphi \in F$;

(F3) if $\varphi, \psi \in F$, then $(\varphi \wedge \psi) \in F$.

Remark 1.1.3. Occasionally we shall use another set Var' of variables. The set of *(propositional) formulas with variables Var'* is defined in the same way replacing (F1) by $Var' \subseteq F$.

Remark 1.1.4. This is well-defined: let $\mathcal{F}$ be the collection of sets of words satisfying (F1)-(F3). Then $\mathcal{F} \neq \varnothing$ because it contains the set of all words. Then $\bigcap \mathcal{F} = \bigcap_{F \in \mathcal{F}} F$ satisfies (F1)-(F3). Clearly, it is contained in all $F \in \mathcal{F}$.

Lemma 1.1.5 (Unique readability). *For every formula φ exactly one of the following holds:*

1. *$\varphi \in Var$.*
2. *$\varphi = \neg\psi$ for some formula ψ; then φ is a* negation *(of ψ).*
3. *$\varphi = (\psi \wedge \chi)$ for some formulas ψ, χ; then φ is a* conjunction *(of ψ, χ).*

Moreover in case 2, ψ is uniquely determined, and in case 3, ψ, χ are uniquely determined.

Proof. At most one of these cases holds: clear, since first symbols are different. At least one of the cases holds: if F is the set of formulas and none of the cases holds for φ, then $F \setminus \{\varphi\}$ satisfies (F1)-(F3), contradiction.

Claim: No formula is a proper prefix of another.

This implies uniqueness in case 3 as follows: if $(\psi \wedge \chi) = (\psi' \wedge \chi')$ then $\psi = \psi'$ since otherwise one would be a prefix of the other; then also $\chi = \chi'$ follows.

To prove the claim, assume for contradiction that there exists a formula that has a proper prefix that is a formula too. Choose such a formula φ of minimal length. Let φ' be a formula that is a proper prefix of φ. We derive a contradiction showing that none of the 3 cases can happen for φ.

If $\varphi \in Var$, then $\varphi' = \epsilon$ but ϵ is not a formula since $F \setminus \{\epsilon\}$ satisfies (F1)-(F3).

Assume $\varphi = \neg\psi$ for some formula ψ. Then φ' starts with $\neg$, so there is a formula ψ' such that $\varphi' = \neg\psi'$. Then ψ' is a proper prefix of ψ. This contradicts the minimality of φ.

Assume $\varphi = (\psi \wedge \chi)$. Then φ' starts with (, so $\varphi' = (\psi' \wedge \chi')$ for some formulas ψ', χ'. By minimality of φ, neither ψ is a proper prefix of ψ', nor vice-versa. Then $\psi = \psi'$. Then χ' is a proper prefix of χ, in contradiction to the minimality of φ. □

Lemma 1.1.6 (Induction on syntax). *Let P be a set of formulas such that for all formulas φ, ψ:*

1. *$Var \subseteq P$;*
2. *if $\varphi \in P$, then $\neg\varphi \in P$;*
3. *if $\varphi, \psi \in P$, then $(\varphi \wedge \psi) \in P$.*

Then P is the set of all formulas.

Proof. Assume there exist formulas not in P. Let φ be a formula outside P of minimal length. Then φ is not a variable by 1, not a negation by 2 and not a conjunction by 3. This contradicts the previous lemma.

One can argue without Lemma 1.1.5: let F be the set of formulas, then the assumptions state $P \subseteq F$ and P satisfies (F1)-(F3). As F is smallest, $F \subseteq P$. □

Exercise 1.1.7. Show that every formula has the same number of occurrences of (as of).

Lemma 1.1.5 also enables definitions by recursion.

Example 1.1.8. There is exactly one function *sub* defined on the set of formulas that satisfies for all formulas φ, ψ and $X \in Var$:

1. $sub(X) := \{X\}$;
2. $sub(\neg\varphi) = \{\neg\varphi\} \cup sub(\varphi)$;
3. $sub((\varphi \wedge \psi)) = \{(\varphi \wedge \psi)\} \cup sub(\varphi) \cup sub(\psi)$.

Elements of $sub(\varphi)$ (distinct from φ) are *(proper) subformulas of* φ.

Proof. We show by induction on $n \in \mathbb{N}$ that there exists exactly one function s_n whose domain is the set of formulas of length $\leqslant n$ and that satisfies (1)-(3). For $n := 0$, let s_0 be the empty function. For $n > 0$, let s_n agree with s_{n-1} on formulas of length $< n$, and for formulas of length n, define s_n as follows. If φ is a variable X (and $n = 1$), then $s_n(X) := \{X\}$. If $\varphi = \neg\psi$, then $s_n(\varphi) := \{\varphi\} \cup s_{|\psi|}(\psi)$. If $\varphi = (\psi \wedge \chi)$, then $s_n(\varphi) := \{\varphi\} \cup s_{|\psi|}(\psi) \cup s_{|\chi|}(\chi)$. Observe that s_n is well-defined by Lemma 1.1.5. It is clear that this is the only possibility in order to satisfy (1)-(3).

Define $sub(\varphi) := s_{|\varphi|}(\varphi)$. Then (1)-(3) are obvious. □

Exercise 1.1.9. Show that every formula φ has at most $|\varphi|$ many subformulas.

1.2 Semantics

Roughly, the "worlds" propositional logic is supposed to talk about are sequences of 0s and 1s, with variables denoting bits – formally:

Definition 1.2.1. A *(total) assignment* is a function $\beta : Var \to \{0, 1\}$. For $V \subseteq Var$ we let $\beta \upharpoonright V$ denote the restriction of β to V: it has domain V and maps every $X \in V$ to $\beta(X)$; conversely, β is an *extension of* $\beta \upharpoonright V$. We refer to $\beta \upharpoonright V$ as a *partial assigment* or an *assignment to* V.

Truth is determined by *Tarski's T-conditions*:

Definition 1.2.2 (Semantics)**.** For a (total) assignment β and a formula φ we define $\beta \models \varphi$ by recursion on φ:

(T1) if $\varphi \in Var$, then: $\beta \models \varphi$ if and only if $\beta(\varphi) = 1$;

(T2) if $\varphi = \neg\psi$ for some formula ψ, then: $\beta \models \varphi$ if and only if $\beta \not\models \psi$;

(T3) if $\varphi = (\psi \wedge \chi)$ for some formulas ψ, χ, then: $\beta \models \varphi$ if and only if both $\beta \models \psi$ and $\beta \models \chi$.

We read $\beta \models \varphi$ as φ is *true under* β or β *satisfies* φ. We read $\beta \not\models \varphi$ as φ is *false under* β or β *falsifies* φ. The *truth value of* φ *under* β is 1 or 0 depending on whether $\beta \models \varphi$ or not.

Exercise 1.2.3. Show that there exists exactly one relation $\models$ between assignments and formulas satisfying (T1)-(T3).

Definition 1.2.2 should be read as a description of a recursive algorithm:

Proposition 1.2.4. *There is an efficient algorithm that given a formula φ and a partial assignment β defined on all variables occuring in φ, outputs the truth value of φ under β.*

Proof. The algorithm checks whether φ is a variable X or a negation $\neg\psi$ or a conjunction $(\psi \wedge \chi)$. In the first case it outputs $\beta(X)$. In the second it recurses on ψ and outputs $1 - b$ where b is the bit returned by the recursive call. In the third case, it recurses on ψ. If this recursive call returns 0, it outputs 0. Else it recurses on χ and outputs the bit this recursive call returns.

To see that this algorithm is efficient observe every recursive call is on a subformula of φ, so there are at most $|sub(\varphi)| \leqslant |\varphi|$ many recursive calls. Each recursive call involves a check and the computation of certain subformulas, plus possibly the retrieval of a value of β from the input; this is clearly efficient. $\square$

Remark 1.2.5. For formulas φ, ψ we use the following abbreviations:

$$\begin{array}{rcl} (\varphi \vee \psi) & := & \neg(\neg\varphi \wedge \neg\psi) \\ (\varphi \to \psi) & := & \neg(\varphi \wedge \neg\psi) \\ (\varphi \leftrightarrow \psi) & := & ((\varphi \to \psi) \wedge (\psi \to \varphi)) \end{array}$$

We call $(\varphi \vee \psi)$ is a *disjunction (of φ and ψ)*. Then for all assignments β:

$\beta \models (\varphi \vee \psi)$ if and only if: $\beta \models \varphi$ or $\beta \models \psi$.

$\beta \models (\varphi \to \psi)$ if and only if: if $\beta \models \varphi$, then $\beta \models \psi$.

$\beta \models (\varphi \leftrightarrow \psi)$ if and only if: φ, ψ have the same truth value under β.

Example 1.2.6. Let β be an assignment. The following are equivalent:

$$\begin{array}{l} \beta \models ((\neg X \wedge Y) \vee Z) \\ \beta \models (\neg X \wedge Y) \text{ or } \beta \models Z \\ \beta \models \neg X \text{ and } \beta \models Y, \text{ or } \beta(Z) = 1 \\ \beta \not\models X \text{ and } \beta(Y) = 1, \text{ or } \beta(Z) = 1 \\ \beta(X) = 0 \text{ and } \beta(Y) = 1, \text{ or } \beta(Z) = 1. \end{array}$$

Exercise 1.2.7. Let $(\varphi \mid \psi)$ abbreviate $(\neg\varphi \wedge \neg\psi)$, Show every formula is equivalent to one built from variables and $\mid$ alone.

For φ a formula with variables in V and an assignment β, we say $\beta \models \varphi$ if $\gamma \models \varphi$ for some total assignment γ extending β (i.e., $\gamma{\restriction}V = \beta$). We now check the obvious fact that this notation is well-defined in that it does not depend on the choice of γ:

Lemma 1.2.8 (Coincidence)**.** *Let φ be a formula* with variables in *$V \subseteq Var$, i.e., only variables from V occur in φ. Let β, γ be assignments with $\beta{\restriction}V = \gamma{\restriction}V$. Then*

$$\beta \models \varphi \iff \gamma \models \varphi.$$

Proof. Induction on syntax. The claim holds for all variables X:

$$\beta \models X \iff \beta(X) = 1 \iff \gamma(X) = 1 \iff \gamma \models X.$$

If the claim holds for φ, then also for $\neg\varphi$:

$$\beta \models \neg\varphi \iff \beta \not\models \varphi \iff \gamma \not\models \varphi \iff \gamma \models \neg\varphi.$$

If the claim holds for φ, ψ, then also for $(\varphi \wedge \psi)$:

$$\beta \models (\varphi \wedge \psi) \iff \beta \models \varphi \text{ and } \beta \models \psi \iff \gamma \models \varphi \text{ and } \gamma \models \psi \iff \gamma \models (\varphi \wedge \psi). \qquad \square$$

1.2.1 Truth tables

The following table shows how basic formulas evaluate:

X	Y	$\neg X$	$(X \wedge Y)$	$(X \vee Y)$	$(X \to Y)$	$(X \leftrightarrow Y)$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Here e.g. the second row considers an assignment β with $\beta(X) = 1$ and $\beta(Y) = 0$. The other entries state that the truth value under β of $\neg X$ is 0, of $(X \wedge Y)$ is 0, and so on.

For the formula $((\neg X \wedge Y) \vee Z)$ of Example 1.2.6 we produce the table

X	Y	Z	$\neg X$	$(\neg X \wedge Y)$	$((\neg X \wedge Y) \vee Z)$
1	1	1	0	0	1
1	1	0	0	0	0
1	0	1	0	0	1
1	0	0	0	0	0
0	1	1	1	1	1
0	1	0	1	1	1
0	0	1	1	0	1
0	0	0	1	0	0

E.g. the last column is produced applying the above table for $\vee$ to the 5th and 3rd column. The satisfying partial assignments are exactly those with a 1 in the last column, the first is $\beta(X) = \beta(Y) = \beta(Z) = 1$, the second $\beta(X) = \beta(Z) = 1, \beta(Y) = 0$, and so on. The falsifying partial assignments are exactly those with a 0 in the last column.

1.3 Semantic concepts

Definition 1.3.1. Let φ, ψ be formulas and T a set of formulas.

1. φ is *satisfiable* if there exists an assignment satisfying φ.
2. φ is *valid* or *tautological* if every assignment satisfies φ.
3. φ, ψ are *(logically) equivalent*, symbolically $\varphi \equiv \psi$, if $(\varphi \leftrightarrow \psi)$ is valid.
4. T is *satisfiable* if there exists an assignment β with $\beta \vDash T$, i.e., $\beta \vDash \chi$ for all $\chi \in T$.
5. T *(logically) implies* φ, symbolically $T \vDash \varphi$, if $\beta \vDash \varphi$ for all assignments β with $\beta \vDash T$.

Example 1.3.2. These properties can be read-off truth tables. E.g. $((\neg X \wedge Y) \vee Z)$ is satisfiable (not valif) because the last column contains a 1 (a 0).

Remark 1.3.3.

1. φ is valid if and only if $\neg\varphi$ is unsatisfiable.
2. $T \cup \{\varphi\} \vDash \psi$ if and only if $T \vDash (\varphi \to \psi)$.
3. $T \vDash \varphi$ if and only if $T \cup \{\neg\varphi\}$ is unsatisfiable.

Examples 1.3.4. For all formulas φ, ψ, χ:

$$\begin{aligned}
&(\varphi \wedge (\psi \wedge \chi)) \equiv ((\varphi \wedge \psi) \wedge \chi),\ (\varphi \vee (\psi \vee \chi)) \equiv ((\varphi \vee \psi) \vee \chi)\\
&(\varphi \wedge (\psi \vee \chi)) \equiv ((\varphi \wedge \psi) \vee (\varphi \wedge \chi)),\ (\varphi \vee (\psi \wedge \chi)) \equiv ((\varphi \vee \psi) \wedge (\varphi \vee \chi))\\
&\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi),\ \neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi)\\
&(\varphi \to \psi) \equiv (\neg\varphi \vee \psi),\ (\varphi \to \psi) \equiv (\neg\psi \to \neg\varphi)\\
&(\varphi \to (\psi \to \chi)) \equiv ((\varphi \wedge \psi) \to \chi)\\
&(\varphi \to \psi) \wedge (\varphi \to \chi)) \equiv (\varphi \to (\psi \wedge \chi)),\ ((\varphi \to \chi) \wedge (\psi \to \chi)) \equiv ((\varphi \vee \psi) \to \chi).
\end{aligned}$$

The first line states *associativity of* $\wedge, \vee$, the second the *distributive laws*, the third the *de Morgan laws.*

Definition 1.3.5. Let $n > 0$ and $\Phi := \{\varphi_1, \ldots, \varphi_n\}$ a set of formulas. Their disjunction and conjunction are

$$\bigvee \Phi := \bigvee_{i=1}^{n} \varphi_i := \varphi_1 \vee \cdots \vee \varphi_n \quad \text{and} \quad \bigwedge \Phi := \bigwedge_{i=1}^{n} \varphi_1 := \varphi_1 \wedge \cdots \wedge \varphi_n,$$

where we omit parentheses (recall associativity); the φ_i are *disjuncts* and *conjuncts* of the respective formulas. Write $\neg^1\varphi$ for $\neg\varphi$, and, $\neg^0\varphi$ or for φ. A *literal* has the form $\neg^b X$ for a variable X and $b \in \{0,1\}$. A *clause* is a disjunction of literals, and a *term* is a conjunction of literals. A *disjunctive normal form (DNF)* is a disjunction of terms. A *conjunctive normal form (CNF)* is a conjunction of clauses. That is, DNFs and CNFs have the forms

$$\bigvee_{i=1}^{n}\bigwedge_{j=1}^{n_i}\lambda_{ij} \quad \text{and} \quad \bigwedge_{i=1}^{n}\bigvee_{j=1}^{n_i}\lambda_{ij}$$

respectively, where $n, n_i \in \mathbb{N}_{>0}$, and the λ_{ij} are literals. If $k \in \mathbb{N}_{>0}$ and all $n_i \leqslant k$, we speak of a *k-DNF* and *k-CNF*, respectively.

Proposition 1.3.6 (Expressive completeness). *Let $n > 0$. For every set $P \subseteq \{0,1\}^n$ there exists a formula φ in the variables $X_1, \ldots, X_n$ that* defines *P:*

$$P = \big\{\beta(X_1)\cdots\beta(X_n) \in \{0,1\}^n \mid \beta \vDash \varphi\big\}. \tag{1.1}$$

Moreover, φ can be chosen both as a DNF and a CNF.

Proof. Let $x = x_1\cdots x_n \in \{0,1\}^n$. For all assignments β:

$$\begin{aligned}\beta \vDash T_x &:= \neg^{1-x_1}X_1 \wedge \cdots \wedge \neg^{1-x_n}X_n &&\iff \beta(X_1)\cdots\beta(X_n) = x,\\ \beta \vDash C_x &:= \neg^{x_1}X_1 \vee \cdots \vee \neg^{x_n}X_n &&\iff \beta(X_1)\cdots\beta(X_n) \neq x.\end{aligned}$$

Thus, for all assignments β:

$$\begin{aligned}\beta \vDash \bigvee_{x\in P} T_x &\iff \text{there is } x \in P\text{: } \beta \vDash T_x &&\iff \beta(X_1)\cdots\beta(X_n) \in P,\\ \beta \vDash \bigwedge_{x\in\{0,1\}^n\setminus P} C_x &\iff \text{for all } x \in \{0,1\}^n \setminus P\text{: } \beta \vDash C_x &&\iff \beta(X_1)\cdots\beta(X_n) \in P.\end{aligned}$$ □

Corollary 1.3.7. *For every n, there are up to logical equivalence exactly 2^{2^n} many formulas in the variables $X_1, \ldots, X_n$.*

Proof. For every $P \subseteq \{0,1\}^n$ choose φ_P defining it. These are 2^{2^n} many pairwise non-equivalent formulas. Given any formula φ in the variables $X_1, \ldots, X_n$ let $P \subseteq \{0,1\}^n$ be the set it defines. Then φ is logically equivalent to φ_P. □

Corollary 1.3.8. *Every formula is equivalent both to a CNF and to a DNF.*

Example 1.3.9. DNFs and CNFs can be read-off truth tables. E.g. for $((\neg X \wedge Y) \vee Z)$ (is already a DNF but) the equivalent DNFs and CNFs read-off the truth table are

- "row 1" $\vee$ "row 3" $\vee$ "row 5" $\vee$ "row 6" $\vee$ "row 7"
  $= (X \wedge Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (\neg X \wedge Y \wedge Z) \vee (\neg X \wedge Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z)$,
- "not row 2" $\wedge$ "not row 4" $\wedge$ "not row 8"
  $= (\neg X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee Z) \wedge (X \vee Y \vee Z)$.

1.3.1 Huge DNFs

We show that the move from a formula to a equivalent DNF sometimes necessarily increases the length of the formula enormously.

Proposition 1.3.10. *For every $\ell \in \mathbb{N}$ there is a formula φ_ℓ such that*

1. $|\varphi_\ell| \leqslant 10 \cdot 2^{2\ell}$;
2. *every DNF equivalent to φ_ℓ has length at least $2^{2^\ell - 1}$.*

Proof. 1: For $n > 0$ let $\textsc{Par}_n \subseteq \{0,1\}^n$ consist of the binary strings with an odd number of occurrences of 1. Observe $\textsc{Par}_{2^0}$ is defined by $\varphi_0 := X_1$ and, for $\ell > 0$, $\textsc{Par}_{2^\ell}$ is defined by

$$\varphi_\ell \ := \ (\varphi_{\ell-1} \wedge \neg\varphi'_{\ell-1}) \vee (\neg\varphi_{\ell-1} \wedge \varphi'_{\ell-1}),$$

where $\varphi'_{\ell-1}$ is $\varphi_{\ell-1}$ with variables $X_1, \dots, X_{2^{\ell-1}}$ replaced by $X_{2^{\ell-1}+1}, \dots, X_{2^\ell}$. Hence

$$\begin{aligned} |\varphi_\ell| &\leqslant 4 \cdot |\varphi_{\ell-1}| + 9 \leqslant 4(4|\varphi_{\ell-2}| + 9) + 9 = 4^2|\varphi_{\ell-2}| + 4 \cdot 9 + 9 \leqslant \cdots \\ &\leqslant 4^\ell |\varphi_0| + 4^{\ell-1} \cdot 9 + \cdots + 4 \cdot 9 + 9 \leqslant 4^\ell + 9 \cdot 4^\ell. \end{aligned}$$

2: Let ψ_n be a DNF defining $\textsc{Par}_n$ with a minimal number of terms. Then all terms are satisfiable. Assume there is a term T in ψ_n such that not all of $X_1, \dots, X_n$ occur in T, say, X_i does not occur. Choose an assignment β satisfying T. Then $\beta \models \psi_n$, so $\beta(X_1)\cdots\beta(X_n) \in \textsc{Par}_n$. Let γ be as β but with the value on X_i flipped, i.e., $\gamma(X_j) = \beta(X_j)$ for all $j \in [n] \setminus \{i\}$ and $\gamma(X_i) := 1 - \beta(X_i)$. Then $\gamma \models T$ by the coincidence lemma, so $\gamma \models \psi_n$. But $\gamma(X_1)\cdots\gamma(X_n) \notin \textsc{Par}_n$, a contradiction.

Hence each term is satisfied by at most one assignment to $X_1, \dots, X_n$. But there are 2^{n-1} such assignments β with $\beta(X_1)\cdots\beta(X_n) \in \textsc{Par}_n$. □

Remark 1.3.11. A DNF determining the parity of 267 bits (i.e., defining $\textsc{Par}_{267}$) has more disjuncts than there are atoms in the observable universe.

Exercise 1.3.12. There does not exist an efficient algorithm that, given a formula φ, outputs a equivalent DNF. Same for CNFs.

Proposition 1.3.13. *There is an efficient algorithm that, given a formula φ, outputs an* equisatisfiable *CNF φ', i.e., φ' is satisfiable if and only if φ is satisfiable.*

Proof. For every $\psi \in \mathit{sub}(\varphi)$ let X_ψ be a variable. For every $\psi \in \mathit{sub}(\varphi)$ the algorithm computes clauses as follows:

- if $\psi = X \in \mathit{Var}$ compute: $(\neg X_\psi \vee X), (\neg X \vee X_\psi)$;
- if $\psi = \neg\chi$ compute: $(\neg X_\psi \vee \neg X_\chi), (X_\chi \vee X_\psi)$;
- if $\psi = (\psi_0 \wedge \psi_1)$ compute: $(\neg X_\psi \vee X_{\psi_0}); (\neg X_\psi \vee X_{\psi_1}), (\neg X_{\psi_0} \vee \neg X_{\psi_1} \vee X_\psi)$.

Let χ be the conjunction of these clauses. Set $\varphi' := (\chi \wedge X_\varphi)$. It is clear that φ' can be efficiently computed (recall $|\mathit{sub}(\varphi)| \leqslant |\varphi|$). To see φ' is equisatisfiable to φ it suffices to show for all assignments β:

$$\beta \models \chi \iff \text{for all } \psi \in \mathit{sub}(\varphi)\text{: } \beta(X_\psi) \text{ is the truth value of } \psi \text{ under } \beta.$$

This is proved by a straightforward induction on syntax. □

1.3.2 Decision trees

Assume you have a (question, formally a) formula and want to know its truth value under an assignment hidden in the world. How many values of variables do you need to find out? A strategy intended to minimize this number is a decision tree.

Definition 1.3.14. A *binary tree* is a non-empty finite set $T \subseteq \{0,1\}^*$ of *nodes*, closed under prefixes and such that $t0 \in T$ whenever $t1 \in T$ for all $t \in \{0,1\}$. A node is *inner* if it is a proper prefix of another; otherwise it is a *leaf*; the *root* is the empty string ϵ. T is *full* if whenever $t \in T$ then either both $t0, t1$ or none is in T. The *height* of T is $\max_{t \in T} |t|$.

A decision tree repeats querying variables, each query depending on the answers received so far, and finally outputs a bit.

Definition 1.3.15. Let $\bar{X} = X_1 \cdots X_n$ be a tuple of variables. A *decision tree (with variables $\bar{X}$)* is a pair (T, ℓ) such that T is a full binary tree and the *labeling* ℓ maps every leaf into $\{0,1\}$ and every inner node t to some variable in $\bar{X}$; we say t *queries* $\ell(t)$. We require that no proper prefix of a non-leaf t queries the same variable as t.

A partial assignment β defined on the variables $\bar{X}$ determines a leaf $t(\beta) \in T$, namely the unique leaf $t = t_1 \cdots t_{|t|} \in T$ such that for all $i \in [|t|]$, $t_i = \beta(X_j)$ where $j \in [n]$ is such that $\ell(t_1 \cdots t_{i-1}) = X_j$. The *output of* (T, ℓ) *on* β is $\ell(t(\beta))$.

(T, ℓ) is *equivalent* to a formula φ in the variables $\bar{X}$ if for all assignments β to $\bar{X}$ the output of (T, ℓ) on β is the truth value of φ under β.

Remark 1.3.16. Every formula φ in the variables $\bar{X} := X_1 \cdots X_n$ is equivalent to a decision tree with variables $\bar{X}$ of height n.

Proof. The tree queries all variables and outputs the truth value of φ. Formally, $T := \{0,1\}^{\leqslant n}$ with $\ell(t) := X_{|t|+1}$ for $|t| < n$, and for leaves $t \in \{0,1\}^n$ set $\ell(t)$ to be the truth value of φ under the assignment $X_i \mapsto t_i$. □

Example 1.3.17. Every decision tree equivalent to a formula defining $\textsc{Par}_n \subseteq \{0,1\}^n$ has height n.

Proof. Assume (T, ℓ) has height $< n$. For an assignment β choose $i \in [n]$ such that $\ell(t) \neq X_i$ for all prefixes t of $t(\beta)$; this exists because $|t(\beta)| < n$. Let γ agree with β except $\gamma(X_i) = 1 - \beta(X_i)$. Then $t(\beta) = t(\gamma)$, so the tree has the same output, but exactly one of $\beta(X_1) \cdots \beta(X_n), \gamma(X_1) \cdots \gamma(X_n)$ is in $\textsc{Par}_n$. □

Theorem 1.3.18. *Let $n > 0$ and $k \in [n]$ and φ be a formula in the variables $\bar{X} = X_1 \cdots X_n$.*

1. *If φ is equivalent to a decision tree of height k, then φ is equivalent both to a k-DNF and to a k-CNF.*
2. *If φ is equivalent to both a k-DNF and a k-CNF, then φ is equivalent to a decision tree of height $\leqslant k^2$.*

Proof. 1: Let (T,ℓ) be a decision tree of height k that is equivalent to φ. Let $t = t_1 \cdots t_{|t|}$ range over the leafs that output 1 and let $s = s_1 \cdots s_{|s|}$ range over the leafs that output 0. Then φ is equivalent to both

$$\bigvee_t \bigwedge_{i<|t|} \neg^{1-t_{i+1}} \ell(t_1 \cdots t_i) \quad \text{and} \quad \bigwedge_s \bigvee_{i<|s|} \neg^{s_{i+1}} \ell(s_1 \cdots s_i).$$

2: Let $\psi = T_1 \vee T_2 \vee \cdots$ and $\chi = C_1 \wedge C_2 \wedge \cdots$ be equivalent to φ, where the T_i conjunctions of $\leqslant k$ literals and the C_j are disjunctions of $\leqslant k$ literals. We can assume that all T_i are satisfiable and that all C_j are falsifiable (otherwise omit them).

We specify a decision tree by informally describing what variables to query or stop with an output, given answers $b_1, \ldots, b_{r-1}$ to previous queries $X_{i_1}, \ldots, X_{i_{r-1}}$, in other words, given the partial assignment that maps X_{i_j} to b_j. Formally, this is a node $t = t_1 \cdots t_{r-1}$ with $b_j = t_{j+1}, \ell(t_1 \cdots t_j) = X_{i_j}$ for all $j < r$. Naturally, we say a partial assignment *satisfies (falsifies)* T_i if it satisfies (falsifies) all (some) of its conjuncts.

The tree proceeds in rounds. In each round a partial assignment β is given; in the first round β is the empty assignment. The tree outputs 1 if β satisfies some T_i or all C_j. It outputs 0 if β falsifies all T_i or some C_j. Otherwise it queries the variables of T_{i_0} outside the domain of β; here, i_0 is minimal such that β does not falsify T_{i_0}.

Key observation: if C_j is not satisfied by β, then some of its variables is queried.

Otherwise the variables in C_j outside the domain of β are disjoint from the variables in T_{i_0} outside the domain of β; but then there exists an assignment extending β that satisfies T_i and falsifies C_j – contradicting $\psi \equiv \chi$.

Thus, after $\leqslant k$ rounds, β is defined on all variables of all C_j or satisfies C_j. Then the tree halts with an output. Since every round queries $\leqslant k$ variables, the height is $\leqslant k^2$. □

1.4 Formal reasoning I: Gentzen's Logischer Kalkül

Definition 1.4.1. A *sequent* is a pair (Γ, Δ) of finite sets of formulas Γ, Δ, written $\Gamma \Rightarrow \Delta$. $\Gamma \Rightarrow \Delta$ is *valid* if $((\bigwedge \Gamma) \to (\bigvee \Delta))$ is valid, equivalently, $\Gamma \vDash \bigvee \Delta$.

Definition 1.4.2. An *LK-proof* is a finite sequence of sequents such that every sequent in it is a conclusion of an LK-rule with premisses appearing earlier in the sequence. There are the following *LK-rules*, written $\frac{\text{Premisses}}{\text{Conclusion}}$. We write e.g. Γ, φ instead of $\Gamma \cup \{\varphi\}$.

$$\text{Axiom} \quad \frac{}{\Gamma, \varphi \Rightarrow \Delta, \varphi} \qquad \text{Weakening} \quad \frac{\Gamma \Rightarrow \Delta}{\Gamma' \Rightarrow \Delta'} \quad \text{if } \Gamma \subseteq \Gamma', \Delta \subseteq \Delta'$$

$$\neg\text{-left} \quad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma, \neg\varphi \Rightarrow \Delta} \qquad \neg\text{-right} \quad \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi}$$

$$\wedge\text{-left} \quad \frac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, (\varphi \wedge \psi) \Rightarrow \Delta} \qquad \wedge\text{-right} \quad \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, (\varphi \wedge \psi)}$$

An LK-proof is a proof *of* its last sequent. A sequent is *LK-provable* if there is an LK-proof of it. A formula φ is *LK-provable* if so is $\Rightarrow \varphi$ (i.e. $\varnothing \Rightarrow \{\varphi\}$).

Remark 1.4.3. Define LK⁺-proof like LK-proofs but additionally allowing the rules:

$$\lor\text{-left}\quad \frac{\Gamma,\varphi\Rightarrow\Delta \qquad \Gamma,\psi\Rightarrow\Delta}{\Gamma,(\varphi\lor\psi)\Rightarrow\Delta} \qquad\qquad \lor\text{-right}\quad \frac{\Gamma\Rightarrow\Delta,\varphi,\psi}{\Gamma\Rightarrow\Delta,(\varphi\lor\psi)}$$

$$\to\text{-left}\quad \frac{\Gamma\Rightarrow\Delta,\varphi \qquad \Gamma,\psi\Rightarrow\Delta}{\Gamma,(\varphi\to\psi)\Rightarrow\Delta} \qquad\qquad \to\text{-right}\quad \frac{\Gamma,\varphi\Rightarrow\Delta,\psi}{\Gamma\Rightarrow\Delta,(\varphi\to\psi)}$$

$$\leftrightarrow\text{-left}\quad \frac{\Gamma,\varphi,\psi\Rightarrow\Delta \qquad \Gamma\Rightarrow\Delta,\varphi,\psi}{\Gamma,(\varphi\leftrightarrow\psi)\Rightarrow\Delta} \qquad \leftrightarrow\text{-right}\quad \frac{\Gamma,\varphi\Rightarrow\Delta,\psi \qquad \Gamma,\psi\Rightarrow\Delta,\varphi}{\Gamma\Rightarrow\Delta,(\varphi\leftrightarrow\psi)}$$

Then LK⁺-provable sequents are LK-provable.

Proof. In an LK⁺-proof replace applications of $\to$-left and $\to$-right by the "proof trees":

$$\dfrac{\dfrac{\Gamma\Rightarrow\Delta,\varphi \qquad \dfrac{\Gamma,\psi\Rightarrow\Delta}{\Gamma\Rightarrow\Delta,\neg\psi}}{\Gamma\Rightarrow\Delta,(\varphi\land\neg\psi)}}{\Gamma,\neg(\varphi\land\neg\psi)\Rightarrow\Delta} \qquad\qquad \dfrac{\dfrac{\dfrac{\Gamma,\varphi\Rightarrow\Delta,\psi}{\Gamma,\varphi,\neg\psi\Rightarrow\Delta}}{\Gamma,(\varphi\land\neg\psi)\Rightarrow\Delta}}{\Gamma\Rightarrow\Delta,\neg(\varphi\land\neg\psi)}$$

More precisely, an application is replaced by a sequence of sequents suitably listing the sequents in these trees. Proceed similarly for the other rules. □

Example 1.4.4. Below is an LK⁺-proof of $\neg(X\land Y)\Rightarrow(\neg X\lor\neg Y)$ numbered and annotated with the rules applied, and the same proof displayed as a "proof tree".

1	$X\Rightarrow X,\neg Y$	Axiom
2	$\Rightarrow X,\neg X,\neg Y$	$\neg$-right on 1
3	$Y\Rightarrow \neg X, Y$	Axiom
4	$\Rightarrow Y,\neg X,\neg Y$	$\neg$-right on 3
5	$\Rightarrow (X\land Y),\neg X,\neg Y$	$\land$-right on 2,4
6	$\Rightarrow (X\land Y),(\neg X\lor\neg Y)$	$\lor$-right on 5
7	$\neg(X\land Y)\Rightarrow(\neg X\lor\neg Y)$	$\neg$-left on 6

$$\dfrac{\dfrac{\dfrac{\dfrac{X\Rightarrow X,\neg Y}{\Rightarrow X,\neg X,\neg Y} \qquad \dfrac{Y\Rightarrow Y,\neg X}{\Rightarrow Y,\neg X,\neg Y}}{\Rightarrow (X\land Y),\neg X,\neg Y}}{\Rightarrow (X\land Y),(\neg X\lor\neg Y)}}{\neg(X\land Y)\Rightarrow(\neg X\lor\neg Y)}$$

Intuition: proof trees are constructed bottom-up. Start with the sequent $\Gamma\Rightarrow\Delta$ to be proved. Read it as an assumption "all formulas in Γ are true and all formulas in Δ are false" that you wish to refute. All rules are self-explanatory when read bottom-up in this way. E.g. at the branching point above ($\land$-right) we have "all $(X\land Y),\neg X,\neg Y$ are false" and make a case distinction: "all $X,\neg X,\neg Y$ are false" or "all $Y,\neg X,\neg Y$ are false". Each leaf of the tree is the desired contradiction: a formula is assumed both true and false.

Exercise 1.4.5. Define a new connective $(\varphi\oplus\psi)$ with intended meaning "either φ or ψ". Which $\oplus$-left and $\oplus$-right rules would you add?

Remark 1.4.6.

1. *(Soundness)* For every LK-rule: if all premisses are valid, then so is the conclusion.

2. *(Inversion)* For every LK-rule except Weakening: if the conclusion is valid, then all premisses are valid.

Theorem 1.4.7 (LK completeness)**.** *A sequent or formula is valid if and only if it is LK-provable. In fact, every valid formula φ has an LK-proof of length $\leqslant 2^{|\varphi|}$.*

Proof. It suffices to consider sequents. $\Leftarrow$ we show, given an LK-proof $S_1, \dots, S_k$, that all sequents S_i are valid. This follows from Remark 1.4.6 (1) by a simple induction.

$\Rightarrow$: let $\Gamma \Rightarrow \Delta$ be valid. We proceed by induction on the number of $\wedge, \neg$ occurrences in it. If this number is 0, all formulas in $\Gamma \cup \Delta$ are variables. Then there exists a variable appearing in both $\Gamma \cup \Delta$ – otherwise an assignment mapping all variables in Γ to 1 and all in Δ to 0, shows $\Gamma \Rightarrow \Delta$ is not valid. Hence, $\Gamma \Rightarrow \Delta$ is the conclusion of Axiom.

If Γ (resp. Δ) contains a negation $\neg\varphi$ or a conjunction $(\varphi \wedge \psi)$, then $\Gamma \Rightarrow \Delta$ is the conclusion of $\neg$-left or $\wedge$-left (resp. $\neg$-right or $\wedge$-right). The premisses are valid by Remark 1.4.6 (2), so LK-provable by induction. Hence $\Gamma \Rightarrow \Delta$ is LK-provable.

For the second statement, choose s_n minimal such that every valid sequent with n occurrences of $\wedge, \neg$ has an LK-proof of length s_n. By the above we have the recurrence $s_{n+1} \leqslant 2s_n + 1$ with $s_0 = 1$. As $n < |\varphi|$, our claim follows:

$$s_n \leqslant 2s_{n-1} + 1 \leqslant 2(2s_{n-2} + 1) + 1 = 2^2 s_{n-2} + 2 + 1 \leqslant \cdots \leqslant 2^n s_0 + 2^{n-1} + \ldots + 1 = 2^{n+1}. \qquad \square$$

Remark 1.4.8. Inspecting the proof we see that the above holds for LK without Weakening and requiring in Axiom that there appears a variable on both sides.

The theorem implies that the set of provable sequents is not increased when adding any sound rule (i.e., preserving validity). An important example is *Cut*:

$$\frac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}.$$

1.4.1 Compactness theorem and applications

Theorem 1.4.9 (Compactness)**.** *A set of formulas T is satisfiable if and only if every finite subset of T is satisfiable.*

Proof. $\Rightarrow$ is trivial. $\Leftarrow$: let T_n be the set of formulas in T in the variables $X_1, \dots, X_n$. By Corollary 1.3.7, T_n contains only finitely many formulas up to logical equivalence. Let B_n be the set of partial assignments to the variables $X_1, \dots, X_n$ that satisfy T_n. Then $B_n \neq \varnothing$ by assumption. Hence $B := \bigcup_n B_n$ is infinite.

Choose $b_1 \in \{0, 1\}$ such that $\{\beta \in B \mid \beta(X_1) = b_1\}$ is infinite. Choose $b_2 \in \{0, 1\}$ such that $\{\beta \in B \mid \beta(X_1) = b_1, \beta(X_2) = b_2\}$ is infinite. And so on.

Let γ be the total assignment $X_i \mapsto b_i$. We claim $\gamma \vDash T$. It suffices to show $\gamma_n := \gamma \upharpoonright \{X_1, \dots, X_n\} \in B_n$ for all $n > 0$. But there exists $\beta \in B$ extending γ_n (even infinitely many). Say $\beta \in B_m$ for $m \geqslant n$. As $\beta \vDash T_m \supseteq T_n$ we have $\beta \upharpoonright \{X_1, \dots, X_n\} = \gamma_n \vDash T_n$. $\square$

Exercise 1.4.10. Let A be a (possibly infinite) alphabet. A *tree T over A* is a prefix closed subset of A^*. It is *finitely branching* if for every $t \in T$ the set $\{a \in A \mid ta \in T\}$ is finite. Show: if T is infinite, then there is an *infinite branch*: a sequence $a_1, a_2, \ldots$ such that $a_1 \cdots a_n \in T$ for all $n \in \mathbb{N}_{>0}$.

Definition 1.4.11. Let T be a set of formulas and φ a formula. T *proves* φ, symbolically $T \vdash \varphi$, if there is a finite $T_0 \subseteq T$ such that $T_0 \Rightarrow \varphi$ is LK-provable.

Theorem 1.4.12 (Deductive LK completeness)**.** *Let T be a set of formulas and φ a formula. Then $T \vDash \varphi$ if and only if $T \vdash \varphi$.*

Proof. The following are equivalent: $T \vDash \varphi$, $T \cup \{\neg\varphi\}$ is unsatisfiable, $T \cup \{\neg\varphi\}$ is unsatisfiable for some finite $T_0 \subseteq T$ (compactness), $T_0 \Rightarrow \varphi$ is valid for some finite $T_0 \subseteq T$, $T_0 \Rightarrow \varphi$ is LK-provable for some finite $T_0 \subseteq T$ (Theorem 1.4.7), $T \vdash \varphi$. □

We recall some graph terminology. A *directed graph* G is a pair (V, E) where V is a non-empty set of *vertices*, and $E \subseteq V^2$ a set of *edges* that is irreflexive (i.e, $(v, v) \notin E$ for all $v \in V$). If additionally E is symmetric (i.e., if $(v, v') \in E$, then $(v, v') \in E$ for all $v, v' \in V$), then G is a *graph*. A graph $H = (W, F)$ is a *subgraph* of G if $W \subseteq V$ and $F \subseteq E$; it is *induced* if $F = E \cap W^2$. Roughly, induced subgraphs are obtained by deleting vertices, and subgraphs by additionally deleting edges.

Example 1.4.13. A graph G is 3-*colorable* if there exists a function $c : V \to [3]$ such that $c(v) \neq c(v')$ for all $(v, v') \in E$. An infinite graph $G = (V, E)$ is 3-colorable if and only if every finite subgraph is 3-colorable.

Proof. We prove this only for the case that V is countable. Then we can choose for every $v \in V$ variables X_v^1, X_v^2, X_v^3. Write $G = (V, E)$. Let T_G be the set of the following formulas for all $v \in V$ and all $(u, w) \in E$

$$\begin{aligned}
&(X_v^1 \vee X_v^2 \vee X_v^3),\\
&(\neg X_v^1 \vee \neg X_v^2),\ (\neg X_v^2 \vee \neg X_v^3),\ (\neg X_v^1 \vee \neg X_v^3),\\
&(\neg X_u^1 \vee \neg X_w^1),\ (\neg X_u^2 \vee \neg X_w^2),\ (\neg X_u^3 \vee \neg X_w^3).
\end{aligned}$$

Intuitively, X_v^i is true if and only if v has color i. The first line states that v gets at least one color, the second that v gets at most one colour, the third that (u, w) is not monochromatic. We now verify these intuitions by showing that T_G is satisfiable if and only if G is 3-colorable.

Assume G is 3-colourable, say via $c : V \to [3]$. Define $\beta(X_v^i) := 1$ if $c(v) = i$, and $\beta(X_v^i) := 0$ otherwise. Then $\beta \vDash T_G$.

Assume $\gamma \vDash \varphi_G$. Let $v \in V$. By the first two lines, there is exactly one $i \in [3]$ such that $\gamma(X_v^i) = 1$. Define $c(v)$ to be this i. Let $(u, w) \in E$. By the 3rd line, $c(u) \neq c(w)$.

By compactness, T_G is satisfiable if every finite subset is 3-colorable. Such a finite subset is included in T_H for a finite subgraph H of G, so is satisfiable if H is 3-colorable. Hence G is 3-colorable if every finite subgraph is. The converse is trivial. □

1.5 Sat modelling

This subsection shows by means of examples how basic computational problems can be modeled by the problem Sat. This is practically relevant for software engineering because there are powerful Sat solvers that show a remarkable and theoretically not well-understood efficiency on real world inputs.

Proposition 1.5.1. *There exists an algorithm that decides the problem*

Sat	
Input:	a propositional formula φ.
Problem:	is φ satisfiable?

Proof. Compute the set V of variables occurring in φ. Compute a list of all partial assignments β to V and check for each such β, using the algorithm from Proposition 1.2.4, whether $\beta \models \varphi$. If at least one check clears, output 1; else output 0. □

Remark 1.5.2 (P versus NP)**.** The above algorithm is not efficient: if φ has n variables, then there are 2^n many assignments to consider. The hypothesis $\mathrm{P} \neq \mathrm{NP}$ is equivalent to the statement that efficient algorithms do not exist, i.e., runtime $\leqslant n^{O(1)}$ is impossible. The *Exponential Time Hypothesis* states that even runtime $\leqslant 2^{o(n)}$ is impossible.

P versus NP is the core problem of theoretical computer science and, more generally, one of "the three greatest problems of mathematics".[1] The pivotal role of Sat stems from the Cook-Levin theorem (1973) stating, roughly speaking, that every NP problem can in a certain sense be modeled by Sat. We showcase this by examples. It is known that if any of the problems mentioned in this subsection admits an effective algorithm, then $\mathrm{P} = \mathrm{NP}$.

Proposition 1.5.3. *There is an efficient algorithm deciding* Sat *if and only if there is an efficient algorithm deciding*

3Sat	
Input:	a 3-CNF φ.
Problem:	is φ satisfiable?

Proof. "Only if" is trivial. "If": the algorithm of Proposition 1.3.13 outputs 3-CNFs. □

Example 1.5.4. Consider the computational problem

3Col	
Input:	a (finite) graph G.
Problem:	is G 3-colourable?

There is an efficient algorithm that, given a (finite) graph G, computes a CNF φ_G that is satisfiable if and only if G is 3-colourable.

[1] S. Smale. Mathematical problems for the next century. In Mathematics: Frontiers and Perspectives, American Mathematical Society, Providence, RI, 271–294, 2000.

Proof. Set $\varphi_G := \bigwedge T_G$ from Example 1.4.13. □

Example 1.5.5. A *vertex cover* of a graph $G = (V,E)$ is a set $C \subseteq V$ such that for all $(u,v) \in E$ we have $u \in C$ or $v \in C$. Consider the problem

VERTEX COVER	
Input:	a graph G and $k \in \mathbb{N}$.
Problem:	does G have a vertex cover of size at most k?

There is an efficient algorithm that, given a graph G and $k \in \mathbb{N}$, outputs a CNF $\varphi_{G,k}$ that is satisfiable if and only if G has a vertex cover of size at most k.

Proof. Write $G = (V,E)$ and let $k \in \mathbb{N}_{>0}$. Use variables $X_{i,v}$ for $v \in V$ and $i \in [k]$. For every $v, v' \in V, v \neq v'$, and $i \in [k]$ consider the clauses:

$$\bigvee_{v \in V} X_{i,v}, \ (\neg X_{i,v} \vee \neg X_{i,v'}).$$

Let χ be the conjunction of all these clauses. Then for all assignments β:

$$\beta \vDash \chi \iff \{(i,v) \mid \beta(X_{i,v}) = 1\} \text{ is the graph of a function from } [k] \text{ into } V.$$

Let $\varphi_{G,k}$ be the conjunction of χ and for every $(u,v) \in E$ the clause

$$\bigvee_{i \in [k]} X_{i,u} \vee \bigvee_{i \in [k]} X_{i,v}.$$

For $k = 0$ set $\varphi_{G,k} := X$ if $E = \emptyset$, and $\varphi_{G,k} := (X \wedge \neg X)$ otherwise. Clearly, $\varphi_{G,k}$ is as desired and efficiently computed – provided $k \leqslant |G|$. For $k > |V|$, re-define $\varphi_{G,k} := \varphi_{G,|V|}$. □

Example 1.5.6. A *partition* of as set S is a a family $\mathcal{P}$ of pairwise disjoint sets with $S = \bigcup \mathcal{P} = \bigcup_{P \in \mathcal{P}} P$. Consider the problem

SET COVER	
Input:	a family $\mathcal{F}$ of sets.
Problem:	does $\mathcal{F}$ contain a partition of $\bigcup \mathcal{F}$?

There is an efficient algorithm that, given a family of sets $\mathcal{F}$, outputs a CNF $\varphi_{\mathcal{F}}$ that is satisfiable if and only if $\mathcal{F}$ contains a partition of $\bigcup \mathcal{F}$.

Proof. If $\mathcal{F} = \emptyset$, let $\varphi_{\mathcal{F}}$ be some satisfiable formula. Otherwise choose variables X_S for $S \in \mathcal{F}$ and let $\varphi_{\mathcal{F}}$ be the conjunction of $(\neg X_S \vee \neg X_{S'})$ for all $S, S' \in \mathcal{F}$ with $S \cap S' \neq \emptyset$, and $\bigvee_{a \in S \in \mathcal{F}} X_S$ for all $a \in \bigcup \mathcal{F}$. □

Exercise 1.5.7. A *clique* in a graph $G = (V,E)$ is a set $C \subseteq V$ such that $(u,v) \in E$ for all distinct $u, v \in C$. Consider the problem

CLIQUE	
Input:	a graph G and $k \in \mathbb{N}$.
Problem:	does G contain a clique of size k?

Show that there exists an efficient algorithm that, given a graph G and $k \in \mathbb{N}$, outputs a formula $\varphi_{G,k}$ that is satisfiable if and only if G contains a clique of size k.

Exercise 1.5.8. A *dominating set* in a graph $G = (V, E)$ is a set $D \subseteq V$ such that for all $v \in V \setminus D$ there is $u \in D$ with $(u, v) \in E$. Consider the problem

Dominating Set	
Input:	a graph G and $k \in \mathbb{N}$.
Problem:	does G contain a dominating set of size k?

Show that there exists an efficient algorithm that, given a graph G and $k \in \mathbb{N}$, outputs a formula $\varphi_{G,k}$ that is satisfiable if and only if G contains a dominating set of size k.

1.5.1 Horn formulas

Example 1.5.9. A *path in* a directed graph $G = (V, E)$ is a finite sequence $v_0, \ldots, v_k$ of pairwise distinct vertices such that $(v_i, v_{i+1}) \in E$ for all $i < k$; it is *from* v_0 and *to* v_k.

Reachability	
Input:	a directed graph G and vertices s, t.
Problem:	is there a path from s to t in G?

There is an efficient algorithm that, given a directed graph G and vertices s, t, outputs a CNF $\varphi_{G,s,t}$ that is satisfiable if and only if there *does not* exist a path from s to t in G.

Proof. Let $G = (V, E)$ be a directed graph and $s, t \in V$. For every $v \in V$ let X_v be a variable. Define $\varphi_{G,s,t}$ as the conjunction of $X_s, \neg X_t$ and $(\neg X_u \vee X_v)$ for every $(u, v) \in E$.

Assume $\beta \models \varphi_{G,s,t}$. Then $\beta(X_s) = 1, \beta(X_t) = 0$. Assume there is a path $s = v_0, \ldots, v_k = t$ in G. It suffices to show $\beta(X_{v_i}) = 1$ for all $i \leqslant k$. This is clear, for $i = 0$. For $i + 1$, we have $\beta(X_{v_i}) = 1$ by induction, so $\beta(X_{v_{i+1}}) = 1$ because $\beta \models (\neg X_{v_i} \vee X_{v_{i+1}})$.

Assume there is *no* path from s to t in G. Set $\beta(X_v) := 1$ if there is a path from s to v in G; otherwise $\beta(X_v) := 0$. Then $\beta(X_s) = 1$ and $\beta(X_t) = 0$. Assume β falsifies a clause $(\neg X_u \vee X_v)$ where $(u, v) \in E$. Then $\beta(X_u) = 1$ and $\beta(X_v) = 0$. Hence there is a path from s to u in G but not to v – nonsense. □

Note $\varphi_{G,s,t}$ is a Horn formula in the following sense.

Definition 1.5.10. A clause $(\neg^{b_1} X_1 \vee \cdots \vee \neg^{b_n} X_n)$ for variables $X_1, \ldots, X_n$ and bits $b_1, \ldots, b_n$ is *Horn* if at most one b_i is 0. A *Horn formula* is a conjunction of Horn clauses.

Theorem 1.5.11. *There is an efficient algorithm deciding*

Horn Sat	
Input:	a Horn formula φ.
Problem:	is φ satisfiable?

Proof. For a clause $C = (\neg^{b_1}X_1 \vee \cdots \vee \neg^{b_n}X_n)$ consider the set $\{\neg^{b_1}X_1, \dots, \neg^{b_n}X_n\}$ of literals. We denote this set again by C. The algorithm takes as input a conjunction φ of clauses C; let $\mathcal{C}$ be the set of these clauses C, each viewed as a set of literals as above. Say an assignment β *satisfies* $\mathcal{C}$, symbolically $\beta \models \mathcal{C}$, if it satisfies some literal in each $C \in \mathcal{C}$. Observe that this happens if and only if $\beta \models \varphi$. The algorithm is as follows:

```
1. while there exists $X \in Var$ with $\{X\} \in \mathcal{C}$.
2.     choose such an $X$
3.     $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C \in \mathcal{C} \mid X \in C\}$
4.     $\mathcal{C} \leftarrow \{C \setminus \{\neg X\} \mid C \in \mathcal{C}\}$
5. if $\emptyset \in \mathcal{C}$, then output 0
6. else output 1
```

This algorithm is efficient because each while loop decreases $|\mathcal{C}|$, and can be efficiently executed. To prove the output in line 5 is correct, it suffices to show that the while loop preserves truth, i.e., for all β: if $\beta \models \mathcal{C}$ and the while loop updates $\mathcal{C}$ to $\mathcal{C}'$, then $\beta \models \mathcal{C}'$. But since $\{X\} \in \mathcal{C}$ and $\beta \models \mathcal{C}$, we have $\beta(X) = 1$. Let $C' \in \mathcal{C}'$, say $C' = C \setminus \{\neg X\}$ for $C \in \mathcal{C}$. Then β satisfies some $\lambda \in C$. Then $\lambda \neq \neg X$, so $\lambda \in C'$. Hence β satisfies C'.

To prove the output in line 6 is correct, note it is only reached if every $C \in \mathcal{C}$ contains a negation. This is satisfiable: take the all 0 assignment. It thus suffices to prove that the while loop preserves unsatisfiability: if $\mathcal{C}'$ is satisfiable, then so is $\mathcal{C}$. But if $\beta \models \mathcal{C}'$, then $\gamma \models \mathcal{C}$ where γ maps X to 1 and otherwise agrees with β. □

Corollary 1.5.12. *There is an efficient algorithm deciding* Reachability.

1.6 Formal reasoning II: Resolution

In the previous section we replaced a clause by the set of its disjuncts (literals), and a CNF by a set of clauses. It is common to call also sets of literals *clauses* and we do so for the rest of this chapter. Crucially, we allow the empty clause $\emptyset$. Look at the algorithm for Horn Sat. Its while loop replaces clauses $\{X\}$ and C with $\neg X \in C$ by a new clause $C \setminus \{X\}$ which is *implied* in the obvious sense. Resolution is a formal proof system that operates with clauses and is based on this simple derivation rule. We adapt our terminology:

Definition 1.6.1. A partial assignment β *satisfies* a clause C, symbolically $\beta \models C$, if $\beta \models \lambda$ for some $\lambda \in C$ (a literal); it *falsifies* C if is falsifies every $\lambda \in C$. A clause C is *tautological* if $X, \neg X \in C$ for some variable X. β *satisfies* a set of clauses $\mathcal{C}$, symbolically $\beta \models \mathcal{C}$, if $\beta \models C$ for all $C \in \mathcal{C}$. If such β exists, $\mathcal{C}$ is *satisfiable.* $\mathcal{C}$ *implies* a clause C, symbolically $\mathcal{C} \models C$, if $\beta \models C$ for every total assignment β with $\beta \models \mathcal{C}$.

Remark 1.6.2. Every partial assignment falsifies the empty clause $C = \emptyset$, and every partial assignment satisfies the empty set of clauses $\mathcal{C} = \emptyset$.

It is important that the definition of $\mathcal{C} \vDash C$ refers to *total* assignments. E.g. for $\mathcal{C} := \varnothing$ and $C := \{X, \neg X\}$ we have $\mathcal{C} \vDash C$ but there is a partial assignment β with $\beta \vDash \mathcal{C}$ and $\beta \nvDash C$, namely the empty assignment.

Definition 1.6.3. Let $\mathcal{C}$ be a set of clauses. A *Resolution proof of C from $\mathcal{C}$* is a finite sequence $C_1, \ldots, C_\ell$ of clauses such that $C_\ell = C$ and for all $i \in [\ell]$ one of the following holds:

- $C_i \in \mathcal{C}$, i.e., C is an *input clause*;
- $C_i = \{X, \neg X\}$ for some variable X, i.e., C is an *axiom*;
- there is $j < i$ such that $C_j \subseteq C_i$, i.e., C_i is a *weakening of C_j*;
- there are $j, k < i$ and a variable $X \in C_j$ such that $\neg X \in C_k$ and $C_i = (C_j \setminus \{X\}) \cup (C_k \setminus \{\neg X\})$, i.e., C_i is a *cut of C_j, C_k (on X)*.

A *Resolution refutation of $\mathcal{C}$* is a Resolution proof of $\varnothing$ from $\mathcal{C}$.

Exercise 1.6.4. Make precise and prove: Resolution is "the same" as the sequent calculus operating with sequents made up of variables only and rules Axiom, Weakening and Cut.

Lemma 1.6.5 (Strong soundness)**.** *Let $\ell \in \mathbb{N}$, $C_1, \ldots, C_\ell$ a Resolution proof and β be a partial assignment. If β satisfies every axiom and input clause in the proof, then $\beta \vDash C_\ell$.*

Proof. We show $\beta \vDash C_i$ for all $i \in [\ell]$ by induction on i. If C_i is an input clause or an axiom this follows by assumption. If C_i is a weakening of C_j for $j < i$, then $\beta \vDash C_j$ by induction, so $\beta \vDash C_i$. If C_i is a cut of C_j, C_k for $j, k < i$, say on $X \in C_j$ we distinguish cases. If β does not satisfy X, then it satisfies some literal $\lambda \in C \setminus \{X\}$ and $\lambda \in C_i$; otherwise, $\beta(X) = 1$ and $\beta \nvDash \neg X$, so β satisfies some literal $\lambda \in C_k \setminus \{\neg X\}$; then $\lambda \in C_i$. □

The qualifier "strong" comes from the use of *partial* assignments (recall Remark 1.6.2). We now show that refutations can dispense with weakenings and tautological clauses.

Lemma 1.6.6. *Let $\mathcal{C}$ be a set of clauses and $\ell \in \mathbb{N}$. If there is a Resolution refutation of $\mathcal{C}$ of length ℓ, then there is one of length $\leqslant \ell$ in which every clause is non-tautological, and every clause is in $\mathcal{C}$ or a cut of earlier clauses.*

Proof. Let $C_1, \ldots, C_\ell = \varnothing$ be a refutation of $\mathcal{C}$. First delete all tautological clauses. The result is still a Resolution refutation: if C_i is non-tautological but a cut of C_j, C_k with, say, C_j tautological, then C is a weakening of C_k. We denote the result again by $C_1, \ldots, C_\ell$.

For $i = 1, \ldots, n$ replace C_i by $C'_i \subseteq C_i$ as follows. If $C_i \in \mathcal{C}$, set $C'_i := C_i$. If C_i is a weakening of C_j, then also of C'_j. Set $C'_i := C_j$. If C_i is a cut of C_j, C_k distinguish cases. Say, the cut is on $X \in C_j$. If both $X \in C'_j$ and $\neg X \in C'_k$, then let C'_i be the cut of C'_j, C'_k on X. Otherwise C_i is a weakening of C'_j or C'_k. Set $C'_i := C'_j$ or $C'_i := C'_k$.

Then $C'_1, \ldots, C'_\ell$ is a Resolution refutation where weakenings just repeat clauses. Delete these repetitions. □

Definition 1.6.7. Let β be a partial assignment and C a clause that is not satisfied by β. Let $C\restriction\beta$ be the clause obtained by removing from C all literals that are falsified by β.

For a set of clauses $\mathcal{C}$ let

$$\mathcal{C}\restriction\beta := \{C\restriction\beta \mid C \in \mathcal{C} \text{ and } \beta \nvDash C\}.$$

Notation: if $X_1, \ldots, X_n$ lists the domain of β and $b_i := \beta(X_i) \in \{0,1\}$, we write

$$C[X_1/b_1, \ldots, X_n/b_n] := C\restriction\beta \quad \text{and} \quad \mathcal{C}[X_1/b_1, \ldots, X_n/b_n] := \mathcal{C}\restriction\beta.$$

Exercise 1.6.8. Let $\mathcal{C}$ be a set of clauses, C a clause and β a partial assignment.

1. $C\restriction\beta = \emptyset$ if and only if β falsifies C.
2. $\mathcal{C}\restriction\beta = \emptyset$ if and only if $\beta \vDash \mathcal{C}$.
3. $\mathcal{C}\restriction\beta$ is satisfiable if and only if some extension γ of β satisfies $\mathcal{C}$.

Lemma 1.6.9. *Let $\ell \in \mathbb{N}$, $\mathcal{C}$ be a set of clauses and β a partial assignment. If $\mathcal{C}$ has a Resolution refutation of length ℓ, then $\mathcal{C}\restriction\beta$ has a Resolution refutation of length $\leqslant \ell$.*

Proof. Let $C_1, \ldots, C_\ell = \emptyset$ be a Resolution refutation of $\mathcal{C}$. By Lemma 1.6.6 we can assume every C_i is in $\mathcal{C}$ or a cut. Let $C_{\nu_1}, \ldots, C_{\nu_{\ell'}}$ be the subsequence of clauses not satisfied by β. Note $\nu_{\ell'} = \ell$. We claim $C_{\nu_1}\restriction\beta, \ldots, C_{\nu_{\ell'}}\restriction\beta = \emptyset$ is a Resolution refutation of $\mathcal{C}\restriction\beta$.

Assume C_{ν_i} is a cut of C_j, C_k for $j, k < \nu_i$, say on X. First case: β is defined on X. Then β satisfies one of C_j, C_k, say C_j. Then β does not satisfy C_k: otherwise it satisfies a literal not involving X in C_k and this literal is in C_{ν_i} - but β does not satisfy C_{ν_i}. Hence $C_k = C_{\nu_{k'}}$ for some $\nu_{k'} < \nu_i$. Then $C_{\nu_i}\restriction\beta$ is a weakening of $C_{\nu_{k'}}\restriction\beta$.

Second case: β is not defined on X. As β does not satisfy C_{ν_i} is does not satisfy C_j, C_k, so they equal C_{ν_j}, C_{ν_k} for some $\nu_j, \nu_k < \nu_i$. Then $C_{\nu_i}\restriction\beta$ is a cut of $C_{\nu_j}\restriction\beta, C_{\nu_k}\restriction\beta$ on X. □

Theorem 1.6.10 (Refutation completeness)**.** *A set of clauses $\mathcal{C}$ is unsatisfiable if and only if $\mathcal{C}$ has a Resolution refutation.*

Proof. $\Leftarrow$ follows from Lemma 1.6.5. $\Rightarrow$: assume $\mathcal{C}$ is unsatisfiable; by compactness we can assume $\mathcal{C}$ is finite. We proceed by induction on the number n of variables occurring in (some clause in) $\mathcal{C}$. If $n = 0$, then $\emptyset \in \mathcal{C}$. Assume $n > 0$ and let X be a variable in $\mathcal{C}$. By Exercise 1.6.8 (3), both $\mathcal{C}[X/0]$ and $\mathcal{C}[X/1]$ are unsatisfiable. By induction they have refutations. Say, $C_1, \ldots, C_\ell = \emptyset$ is a refutation of $\mathcal{C}[X/0]$. Then $C_1 \cup \{X\}, \ldots, C_\ell \cup \{X\} = \{X\}$ becomes a Resolution proof if we add some weakenings. Namely, if C_i is an input clause or axiom that does not contain X, then insert C_i before $C_i \cup \{X\}$. Similarly we get a proof of $\{\neg X\}$ from a refutation of $\mathcal{C}[X/1]$. Compose the proofs and add a final cut on X to get a Resolution refutation of $\mathcal{C}$. □

Exercise 1.6.11. This proof, like many proofs by induction, can be read computationally: it implicitly specifies a recursive algorithm that computes a refutation given as input an unsatisfiable $\mathcal{C}$. Describe this algorithm. How long, at most, is the constructed refutation?

Exercise 1.6.12. Show that there is an efficient algorithm deciding

2-SAT	
Input:	a 2-CNF φ.
Problem:	is φ satisfiable?

Infer Corollary 1.5.12. Further infer that there is an efficient algorithm deciding

BIPARTITE	
Input:	a graph $G = (V, E)$.
Problem:	is G *bipartite*, i.e., are there disjoint $V_0, V_1 \subseteq V$ such that $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$?

Hint: a cut of clauses each of size $\leqslant 2$ has again size $\leqslant 2$.

Theorem 1.6.13 (Deductive Resolution completeness)**.** *Let $\mathcal{C}$ be a set of clauses and C a clause. There is a Resolution proof of C from $\mathcal{C}$ if and only if $\mathcal{C} \vDash C$.*

Proof. $\Rightarrow$ is Lemma 1.6.5. $\Leftarrow$: let $\neg C$ be the set of clauses $\{\neg^{1-b}X\}$ for every literal $\neg^b X \in C$. Then $\mathcal{C} \cup \neg C$ is unsatisfiable. By refutation completeness, it has a refutation $C_1, \ldots, C_\ell = \varnothing$. Then $C_1 \cup C, \ldots, C_\ell \cup C = C$ is a Resolution proof of C from tautological clauses and clauses $D \cup C$ with $D \in \mathcal{C}$. Adding some weakenings gives a proof from $\mathcal{C}$. □

Remark 1.6.14. The use of weakenings above cannot be avoided. E.g., every Resolution proof of $C := \{X\}$ from $\mathcal{C} := \{\varnothing\}$ requires it. The use of axioms $\{X, \neg X\}$ can also not be avoided since every proof of $C := \{X, \neg X\}$ from $\mathcal{C} := \varnothing$ requires it.

1.6.1 Linear and SLD Resolution

Definition 1.6.15. Let $\mathcal{C}$ be a set of clauses. A *linear* Resolution proof from $\mathcal{C}$ is a sequence of clauses $C_1, \ldots, C_\ell$ such that $C_1 \in \mathcal{C}$ and for every $1 < i \leqslant \ell$, C_i is a cut of C_{i-1} and a *side clause* D_i which is an input clause (i.e., in $\mathcal{C}$) or equals C_j for some $j < i$.

A linear Resolution proof is *SLD* is all C_i are *negative*, i.e., contain only negative literals – a literal is *negative* if it starts with $\neg$ and otherwise *positive*.

Remark 1.6.16. In an SLD Resolution proof all side clauses contain exactly one positive literal. Hence only *Horn* clauses are involved, i.e., clauses with a most one positive literal.

We now strengthen Theorem 1.6.10.

Theorem 1.6.17 (Linear refutation completeness)**.** *If $\mathcal{C}$ is an unsatisfiable set of clauses, then there is a linear Resolution refutation of $\mathcal{C}$.*

Proof. We can assume that $\mathcal{C}$ is *minimally unsatisfiable*, i.e., every proper subset of $\mathcal{C}$ is satisfiable. We show that then $\mathcal{C}$ has a linear refutation starting with any $C \in \mathcal{C}$. We proceed by induction on the number n of variables in $\mathcal{C}$. If $n = 0$, $\mathcal{C} = \{\varnothing\}$ and there is nothing to show. Assume $n > 0$. Given $C \in \mathcal{C}$ we distinguish two cases.

First case: $|C| = 1$, say, $C = \{\neg^{1-b}X\}$ for a variable X and $b \in \{0,1\}$. Let $\mathcal{C}'$ be a minimally unsatisfiable subset of $\mathcal{C}[X/b]$. Then there exists $C' \in \mathcal{C}'$ with $C' \cup \{\neg^b X\} \in \mathcal{C}$: otherwise $\mathcal{C}' \subseteq \mathcal{C} \setminus \{C\}$ is satisfiable.

By induction, there is a linear refutation $C' = C'_1, \dots, C'_\ell = \varnothing$ of $\mathcal{C}'$, hence of $\mathcal{C}[X/b]$. The input side clauses D'_i are in $\mathcal{C}[X/b]$. If they are in $\mathcal{C}$, then $C, C'_1, \dots, C'_\ell$ is a linear refutation of $\mathcal{C}$: the initial cut has side clause $C' \cup \{\neg^b X\}$. Otherwise, there are i with $D'_i \cup \{\neg^b X\} \in \mathcal{C}$, say i_0 is minimal such. Using these side clauses instead and adding $\neg^b X$ to all $C'_i, i \geqslant i_0$, gives a linear proof of $\neg^b X$ from $\mathcal{C}$. A final cut with C gives a linear refutation.

Second case: $|C| > 1$. Choose a literal in C, say $\neg^b X$ for a variable X and $b \in \{0,1\}$. Let $\mathcal{C}' \subseteq \mathcal{C}[X/b]$ be minimally unsatisfiable. We claim $C[X/b] = C \setminus \{\neg^b X\} \in \mathcal{C}'$. Indeed, $\mathcal{C}[X/b] \setminus \{C[X/b]\}$ is satisfiable: since $\mathcal{C}$ is minimally unsatisfiable, there is $\beta \vDash \mathcal{C} \setminus \{C\}$; then $\beta \nvDash C$, so $\beta(X) = b$, so $\beta \vDash (\mathcal{C} \setminus \{C\})[X/b]$.

By induction, there is a linear refutation $C'_1, \dots, C'_\ell = \varnothing$ of $\mathcal{C}'$ with $C'_1 = C[X/b] = C \setminus \{\neg^b X\}$. For an input side clause D'_i there is $D_i \in \mathcal{C}$ such that $D_i = D'_i$ or $D_i = D'_i \cup \{\neg^b X\} \in \mathcal{C}$. Adding $\neg^b X$ to all C'_i, gives a linear proof of $\{\neg^b X\}$ from $\mathcal{C}$ with side clauses D_i. It starts with $C'_1 \cup \{\neg^b X\} = C$.

Observe $\mathcal{D} := (\mathcal{C} \setminus \{C\}) \cup \{\neg^b X\}$ is unsatisfiable, $\mathcal{C} \setminus \{C\}$ is satisfiable, so a minimally unsatisfiable subset of $\mathcal{D}$ contains $\{\neg^b X\}$. The first case gives a linear refutation of $\mathcal{D}$ starting with $\{\neg^b X\}$. Putting the above proof of $\{\neg^b X\}$ in front, gives a linear refutation of $\mathcal{C}$ that starts with C. □

Corollary 1.6.18 (SLD refutation completeness)**.** *Every unsatisfiable set of Horn clauses has a SLD Resolution refutation.*

Proof. A minimally unsatisfiable $\mathcal{C}' \subseteq \mathcal{C}$ contains a negative clause C (otherwise the all 1 assignment satisfies it). By the previous proof there is a linear refutation $C_1, \dots, C_\ell = \varnothing$ of $\mathcal{C}'$, hence of $\mathcal{C}$, that starts with $C_1 = C$. As side clauses are Horn, all C_i are negative. □

Exercise 1.6.19. Show that there exists an efficient algorithm that, given an unsatisfiable set of Horn clauses, outputs a SLD Resolution refutation of it.

Hint: this is implicitly done by the algorithm in Theorem 1.5.11.

Remark 1.6.20. It is not known whether linear Resolution efficiently simulates general Resolution. Is there $c \in \mathbb{N}$ such that for all $\ell \in \mathbb{N}$: if $\mathcal{C}$ has a refutation of length ℓ, then it has a linear one of length $\leqslant \ell^c$?

The name SLD stands for "Linear Resolution with Selection function for Definite clauses" and comes from logic programming. We explain what this is in Section 2.9.

1.7 Treelike Resolution

In general a Resolution proof can derive a clause and use it in many later derivation steps as a premiss of cut or weakening. A proof is *treelike* if it is used at most once. The formal definition needs care because, given a clause in a proof, it is not uniquely determined how it is derived: it can be from different earlier clauses and different occurrences of these. Not even the rule used is visible, e.g., an input clause might as well as be a cut, or a cut can be a weakening (cut a tautological clause with itself).

We recall some standard terminology: given a directed graph (V, E), the *in-degree* (*out-degree*) of a vertex $v \in V$ is the number of $u \in V$ such that $(u, v) \in E$ ($(v, u) \in E$). (V, E) is *acyclic* if it contains no directed cycle (i.e., a path from a vertex to itself of length > 1). *Dag* stands for *directed acyclic graph.*

We naturally view binary trees T as dags. More precisely, we say (V, E) is *isomorphic* to T if there is a bijection $B : V \to T$ such that for all $u, v \in V$ we have:

$$(u, v) \in E \iff B(u) = B(v)b \text{ for some } b \in \{0, 1\}.$$

Definition 1.7.1. Let $\mathcal{C}$ be a set of clauses and C a clause. Let $C_1, \dots, C_\ell = C$ be a Resolution proof of C from $\mathcal{C}$. A *proof-dag* of the proof is a directed graph $([\ell], E)$ such that every $i \in [\ell]$ has in-degree 0,1 or 2. In the first case, C_i is an axiom or input clause, in the second a weakening from C_j where $j < i$ is such that $(j, i) \in E$, and in the third C_i is a cut of C_j, C_k where $j, k < i$ are such that $(j, i), (k, i) \in E$.

The proof is *treelike* if it has a proof-dag such that every $i \in [\ell]$ has out-degree $\leqslant 1$

Note a proof-dag is a dag because $(i, j) \in E$ implies $j < i$. For emphasis, usual Resolution proofs are sometimes called *daglike.*

Proposition 1.7.2. *Let $\mathcal{C}$ be a set of clauses, C a clause and $\ell \in \mathbb{N}$. The following are equivalent.*

1. *There is a treelike Resolution proof of C from $\mathcal{C}$ of length $\leqslant \ell$.*
2. *There is a Resolution proof of C from $\mathcal{C}$ with a proof-dag isomorphic to a binary tree of size $\leqslant \ell$.*
3. *There is a binary tree T of size $\leqslant \ell$ and a map c from T to clauses such that*
   (a) *if t is a leaf, then $c(t)$ is an axiom or in $\mathcal{C}$;*
   (b) *if $t0 \in T, t1 \notin T$, then $c(t)$ is a weakening of $c(t0)$;*
   (c) *if $t0, t1 \in T$, then $c(t)$ is a cut of $c(t0), c(t1)$;*
   (d) *$c(\epsilon) = C$.*

Proof. $2 \Rightarrow 1$ is trivial. For $3 \Rightarrow 2$, let $t_1, \dots, t_{|T|}$ list T such that $i < j$ if $|t_i| > |t_j|$. Then $c(t_1), \dots, c(t_{|T|})$ is a refutation of $\mathcal{C}$ with a proof-dag isomorphic to T.

For $1 \Rightarrow 3$ the idea is to start walking from C taking steps to one of the premisses. Since in each step we have at most 2 possibilities, the walks are determined by binary strings and these form T. We now formalize this.

Let $C_1, \dots, C_\ell = C$ be a proof with proof-dag $([\ell], E)$ such that every $\nu \in [\ell]$ has out-degree $\leqslant 1$. For a binary string $t = t_1 \cdots t_s$ with $s \leqslant \ell$ define $\nu_t \leqslant \ell$ as follows – intuitively $\nu_t = 0$ means "fail". If $s = 0$, i.e., $t = \epsilon$, let $\nu_t := \ell$. If $s > 0$ write $t' := t_1 \cdots t_{s-1}$. If $\nu_{t'} := 0$ or $\nu_{t'}$ has in-degree 0 in $([\ell], E)$, set $\nu_t := 0$. If $\nu_{t'}$ has in-degree 1 in $([\ell], E)$, set $\nu_t := 0$ if

$t_s = 1$ and otherwise let ν_t satisfy $(\nu_t, \nu) \in E$. If $\nu_{t'}$ has in-degree 2 in $([\ell], E)$, let $\nu_0, \nu_1 \in [\ell]$ satisfy $(\nu_0, \nu), (\nu_1, \nu) \in E$ and set $\nu_t := \nu_{t_s}$.

Let T be the set of binary strings t with $\nu_t \neq 0$. This is a binary tree and setting $c(t) := C_{\nu_t}$ satisfies (a)-(d).

To see $|T| \leqslant \ell$, it suffices to show that $t \mapsto \nu_t$ is an injection on T. Assume there is $t \in T$ such that $\nu_t = \nu_{t'}$ for some $t' \in T \smallsetminus \{t\}$. Choose such t of minimal length, and a witnessing t' for t. It is easy to see that $\nu_{t''} < \ell$ for all $t'' \neq \epsilon$. Hence both t, t' are non-empty. Write $t = t_- b$ and $t' = t'_- b'$. If $t_- = t'_-$, then $b \neq b'$ and clearly $\nu_t \neq \nu_{t'}$. So $t_- \neq t'_-$. By minimality of t, $\nu_{t_-} \neq \nu_{t'_-}$. But $\nu_t = \nu_{t'}$ has an edge to both $\nu_{t_-}, \nu_{t'_-}$, so out-degree $\geqslant 2$. Contradiction. □

Remark 1.7.3. In the proof of $1 \Rightarrow 3$, the assumption of treelikeness is used only to show that $|T| \leqslant \ell$. Without this assumption we get $|T| \leqslant 2^{\ell+1}$ in (3). Interestingly, we shall prove later that the exponential blow-up moving from a daglike to a treelike refutation can, in general, not be avoided (Section 1.8.1).

We give a simple direct proof of completeness of treelike Resolution:

Proposition 1.7.4 (Treelike refutation completeness)**.** *Let $n \in \mathbb{N}$. If $\mathcal{C}$ is an unsatisfiable set of clauses in n variables, then it has a treelike Resolution refutation of length $3 \cdot 2^n$.*

Proof. We verify Proposition 1.7.2 (c) for the binary tree

$$T := \big\{ t \in \{0,1\}^* \mid |t| \leqslant n \big\} \cup \big\{ t0 \mid t \in \{0,1\}^n \big\}.$$

Then $|T| = 2^{n+1} + 2^n$ as claimed. For $t = t_1 \dots t_m$ with $m < n$ define

$$c(t) := \big\{ \neg^{t_1} X_1, \dots, \neg^{t_m} X_m \big\}.$$

Note $c(\epsilon) = \varnothing$ and $c(t)$ is a cut on $X_{|t|+1}$ of $c(t0), c(t1)$. For t of length n let β be an assignment with $t = \beta(X_1) \cdots \beta(X_n)$. Then β falsifies $c(t)$. Since $\mathcal{C}$ is unsatisfiable, there is $C \in \mathcal{C}$ such that β falsifies C. Then $C \subseteq c(t)$. Define $c(t0) := C$. □

Exercise 1.7.5 (Treelike deductive completeness)**.** Let $\mathcal{C}$ be a set of clauses and C a clause such that $\mathcal{C} \vDash C$. Then there is a treelike resolution proof of C from $\mathcal{C}$ of length $O(2^n)$.

Exercise 1.7.6 (Search decision trees)**.** Suppose you have an unsatisfiable set of clauses $\mathcal{C}$, the world hides an assignment, and you look for a falsified clause in $\mathcal{C}$. How many values do you need to uncover? A *$\mathcal{C}$-labeled decision tree* (T, ℓ) is defined like a decision tree (Definition 1.3.15) but ℓ maps every leaf of T to a clause from $\mathcal{C}$ (instead of 0 or 1). It *solves the search problem of* $\mathcal{C}$ if $\beta \nvDash \ell(t(\beta))$ for every assignment β.

1. For every treelike Resolution refutation of $\mathcal{C}$ there is one at most as long and with proof-dag isomorphic to some binary tree T such that for some ℓ, (T, ℓ) is a $\mathcal{C}$-labeled decision tree that solves the search problem of $\mathcal{C}$.
2. For every $\mathcal{C}$-labeled decision tree that solves the search problem of $\mathcal{C}$, there is a treelike Resolution refutation of $\mathcal{C}$ with proof-dag isomorphic to T.

1.7.1 DPLL algorithms

DPLL stands for Davis, Putnam, Logemann, and Loveland.

Definition 1.7.7. A *DPLL algorithm*, given as input a set of clauses $\mathcal{C}$, proceeds as follows.

```
1. if $\mathcal{C} = \varnothing$ then output 1
2. else if $\varnothing \in \mathcal{C}$, output 0
3. else choose a variable $X$ occurring in $\mathcal{C}$
4. recurse setting $\mathcal{C} \leftarrow \mathcal{C}[X/0]$
5. if this recursive call returns 1, output 1
6. else recurse setting $\mathcal{C} \leftarrow \mathcal{C}[X/1]$
7. if this recursive call returns 1, output 1
8. else output 0
```

Here ***choose*** in line 3 refers to some to-be-specified algorithm that, given a non-empty set of clauses $\mathcal{C}$, outputs a variable occurring in $\mathcal{C}$.

Exercise 1.7.8. Every DPLL algorithm decides

CNF-SAT	
Input:	a set of clauses $\mathcal{C}$.
Problem:	is $\mathcal{C}$ satisfiable?

Remark 1.7.9. In engineering practice, the choice of the subroutine ***choose*** can drastically affect the observed runtimes on real-life inputs. DPLL algorithms are a basic version of so-called SAT-*solvers*, some of which are surprisingly fast on real-life instances. However, if P ≠ NP, then there are no efficient DPLL algorithms. We shall prove later, without relying on any unproven hypotheses like P ≠ NP, that all DPLL algorithms must have even exponential runtime (in the worst case).

We show that rejecting runs of DPLL algorithms "are" treelike Resolution refutations.

Theorem 1.7.10. *If $\mathcal{C}$ is a finite unsatisfiable set of clauses, then the number of steps of every DPLL algorithm on $\mathcal{C}$, is at least as large as the minimal length of a treelike Resolution refutation of $\mathcal{C}$.*

Proof. Every recursive call is determined by a binary string $b_1 \cdots b_w$ (for some $w \in \mathbb{N}$) and has an associated sequence of variables $X_1 \cdots X_w$. It then either rejects in line 2 or chooses a variable X_{w+1} in line 3 and has two recursive calls in lines 4 and 6 determined by $b_1 \cdots b_w 0$ and $b_1 \cdots b_w 1$. The call determined by $b_1 \cdots b_w$ recurses on $\mathcal{C}[X_1/b_1, \ldots, X_w/b_w]$. Set

$$C_{b_1 \cdots b_w} := \left\{ \neg^{b_1} X_1, \ldots, \neg^{b_w} X_w \right\}.$$

If the algorithm does not reject in line 2, then it recurses. The crucial observation is that then $C_{b_1 \cdots b_w}$ is a cut of $C_{b_1 \cdots b_w 0}$ and $C_{b_1 \cdots b_w 1}$ on X_{w+1}.

If the algorithm rejects in line 2, then $\varnothing \in \mathcal{C}[X_1/b_1, \ldots, X_w/b_w]$. This means that $[X_1/b_1, \ldots, X_w/b_w]$ falsifies some clause $C \in \mathcal{C}$. This means $C \subseteq C_{b_1 \cdots b_w}$. In this case additionally define $C_{b_1 \cdots b_w 0} := C$. Setting $c(b) := C_b$ satisfies Proposition 1.7.2 (c). □

1.8 Resolution lower bounds

In this section we give a natural example of a small set of clauses OP_n encoding the "Ordering principle" that has short daglike Resolution refutations but every treelike refutation must be exponentially long. Second we give a natural example of a small set of clauses PHP_n encoding the "Pigeonhole principle" such that every daglike Resolution refutation must be exponentially long. By Theorem 1.7.10 these results imply:

Corollary 1.8.1. *For every sufficiently large $n \in \mathbb{N}$, every DPLL algorithm makes at least $2^{\Omega(n)}$ many steps on input OP_n or PHP_n.*

Our use of "exponential" is not precise: as PHP_n and OP_n have size $m \geqslant n^3$, the lower bound is actually only $2^{\Omega(m^{1/3})}$.

1.8.1 Separation of treelike and daglike Resolution

Recall, an *order* on a set S is a relation $R \subseteq S^2$ that is irreflexive (i.e., $(a,a) \notin R$ for all $a \in S$) and transitive (i.e.,$(a,b),(b,c) \in R$ implies $(a,c) \in R$ for all $a,b,c \in S$). If S is finite, it has a minimal element, so there is no function mapping any element to an R-smaller one. This is expressed by the following set of clauses being unsatisfiable:

Definition 1.8.2 (Ordering principle)**.** Let $n \in \mathbb{N}_{>0}$ and R_{ij}, F_{ij} be variables for $i,j \in [n]$. Let OP_n contain for every $i,j,j',k \in [n], j \neq j'$, the clauses

$$\{\neg R_{ii}\},\ \{\neg R_{ij}, \neg R_{jk}, R_{ik}\},$$
$$\{F_{i1},\dots,F_{in}\},\ \{\neg F_{ij} \vee \neg F_{ij'}\},\ \{\neg F_{ij}, R_{ji}\}.$$

Note OP_n has $O(n^2)$ variables and $O(n^3)$ clauses.

Proposition 1.8.3. *OP_n is unsatisfiable for every $n \in \mathbb{N}_{>0}$.*

Proof. Assume $\beta \models OP_n$. By the first group of clauses, $R := \{(i,j) \in [n]^2 \mid \beta(R_{ij}) = 1\}$ is an order on $[n]$. By the second group, we get a function F mapping i to j if $\beta(F_{ij}) = 1$. F maps every $i \in [n]$ to an R-smaller element. This is impossible. □

Proposition 1.8.4 (Stålmark)**.** *For every $n \in \mathbb{N}$ there are Resolution refutations of OP_n of length $O(n^3)$.*

Proof. For $i,k \in [n]$ let $C^i_k = \{R_{1i},\dots,R_{ki}\}$. We consecutively for $k = n,\dots,1$ derive all $C^1_k,\dots,C^k_k$. Then $C^1_1 = \{R_{11}\}$ and a cut with $\{\neg R_{11}\}$ gives the empty clause.

To start derive C^i_n for every $i \in [n-1]$: cut $\{F_{i1},\dots,F_{in}\}$ consecutively with with $\{\neg F_{ij}, R_{ji}\}$ for $j = 1,\dots,n$ to get $\{R_{1i},\dots,R_{ni}\} = C^i_n$.

Assume $C^1_{k+1},\dots,C^k_{k+1}$ have been derived. For every $i \in [k]$ derive C^i_k as follows. For every $j \in [k]$ cut C^i_{k+1} on $R_{(k+1)i}$ with $\{\neg R_{j(k+1)}, \neg R_{(k+1)i}, R_{ji}\}$ to get $D_j := C^i_k \cup \{\neg R_{j(k+1)}\}$. Cut C^{k+1}_{k+1} consecutively for $j = 1,\dots,k$ with D_j to get $C^i_k \cup \{R_{(k+1),(k+1)}\}$; a cut with the axiom $\{\neg R_{(k+1)(k+1)}\}$ gives C^i_k.

The refutation consists of the $O(n^3)$ input clauses, the $O(n^2)$ clauses C^i_k, and, for every i, k, additionally n clauses D_j, and n intermediate clauses from the consecutive cuts. In total, the proof has length $O(n^3)$. □

Note this length is almost optimal because $|OP_n| \geqslant n^3$. We aim to show:

Theorem 1.8.5. *For every sufficiently large $n \in \mathbb{N}$, every treelike Resolution refutation of OP_n has length $\geqslant 2^{n/6}$.*

For the proof we need the following combinatorial lemma.

Lemma 1.8.6 (Spira). *For a finite binary tree T there exists $t \in T$ such that $T_t := \{t' \in T \mid t$ is a prefix of $t'\}$ has size*

$$\lfloor |T|/3 \rfloor \leqslant |T_t| \leqslant 2|T|/3.$$

Proof. The desired t is computed by the following algorithm.

```
1. $t \leftarrow \epsilon$
2. **while** $|T_t| > 2|T|/3$
3.    **if** $t1 \notin T$ or $|T_{t0}| \geqslant |T_{t1}|$, **then** $t \leftarrow t0$
4.    **else** $t \leftarrow t1$
5. output $t$
```

Obviously, the output t satisfies $|T_t| \leqslant 2|T|/3$. Assume $|T_t| < \lfloor |T|/3 \rfloor$. Note $t \neq \epsilon$, say $t = t'0$ for some $t' \in T$. Then $t_1 \notin T$ or $|T_{t1}| \leqslant |T_{t0}| < \lfloor |T|/3 \rfloor$. Hence $|T_{t'}| \leqslant 2\lfloor |T|/3 \rfloor - 2 + 1 < 2|T|/3$ (with the $+1$ counting $t' \in T_{t'}$). □

Proof of Theorem 1.8.5. Informally, the proof can be outlined as follows: given a refutation of length ℓ, represent it by a binary tree T and clause labeling c. Choose t according to Spira's lemma. If $c(t)$ can be satisfied by a "good" (partial) assignment, do so and delete T_t. This gives a refutation with a new input clause $c(t)$. Otherwise delete everything outside T_t. This gives a proof of $c(t)$, a clause that is unsatisfiable by "good" assignments. Repeat, always considering "good" assignments extending the current one. After $O(\log |T|)$ steps arrive at T' of constant size and a "good" assignment β. Being "good" allows to extend β' to a "good" assignment that satisfies all old input clauses. This is a contradiction – provided the various extensions of "good" assignments exist. They do if ℓ is small.

The "good" assignments are $\beta_{\bar{\imath}}$ where $\bar{\imath} = i_1 \cdots i_s \in [n]^s$ with $s \leqslant n$ and pairwise distinct i_ν. It is defined on R_{ij} if $i = i_\nu, j = i_\mu$ for certain $\nu, \mu \in [s]$; then it maps R_{ij} to 1 if $\mu < \nu$, and to 0 otherwise. It is defined on F_{ij} if $i = i_\nu$ for some $\nu \in [s-1]$; then it maps F_{ij} to 1 if $j = i_{\nu+1}$, and to 0 otherwise.

It should be clear that $\beta_{\bar{\imath}}$ does not falsify any clause in OP_n. A *good extension* of $\beta_{\bar{\imath}}$ is a partial assignment $\beta_{\bar{\imath}'}$ where $\bar{\imath}$ is a prefix of $\bar{\imath}'$. If $\beta_{\bar{\imath}}$ is undefined on R_{ij} or F_{ij}, then there is a good extension $\beta_{\bar{\imath}'}$ of $\beta_{\bar{\imath}}$ with $|\bar{\imath}'| \leqslant s + 2$ that evaluates the variable: prolongue $\bar{\imath}$ by those of i, j that do not appear in $\bar{\imath}$.

Assume OP_n has a treelike refutation of length ℓ. Choose T, c according to Proposition 1.7.2 (c). In particular, $|T| \leqslant \ell$.

By a *subtree of* T we mean a set of the form $\{t_0 t \mid t \in T'\} \subseteq T$ for $t_0 \in T$ and a binary tree T'; t_0 is the *root* of the subtree. The subtree is *full* if for all t it contains either both or none of $t0, t1$, if both $t0, t1 \in T$. Obviously, Spira's lemma applies to subtrees.

We construct full subtrees $T^0, T^1, \ldots$ of T and tuples $\bar{\imath}_0, \bar{\imath}_1, \ldots$ such that for all r:

(a) T^r and $c{\restriction}T^r$ "is" (satisfies (a)-(d) of Proposition 1.7.2 (3)) a treelike Resolution proof of $c(t^r)$ from a set of clauses $OP_n \cup \mathcal{C}^r$.

(b) $\mathcal{C}^r$ contains only clauses satisfied by $\beta_{\bar{\imath}_r}$.

(c) no good extension of $\beta_{\bar{\imath}}$ satisfies $c(t^r)$ where t^r is the root of T^r.

We start with $T^0 := T$ and let $\bar{\imath}_0$ be the empty tuple; $t^0 = \epsilon$, $\mathcal{C}^0 = \emptyset$. Assume $T^r, \bar{\imath}_r$ is constructed. Choose $t \in T^r$ according to Spira's lemma and do the following:

– if there exists a good extension of $\beta_{\bar{\imath}_r}$ that satisfies $c(t)$, then choose such $\beta_{\bar{\imath}_{r+1}}$ with $|\bar{\imath}_{r+1}| \leqslant |\bar{\imath}_r| + 2$; set $T^{r+1} := (T^r \setminus T^r_t) \cup \{t\}$; then $\mathcal{C}^{r+1} = \mathcal{C}^r \cup \{t\}, t^{r+1} = t^r$.

– else set $\bar{\imath}_{r+1} := \bar{\imath}_r$ and $T^{r+1} := T^r_t$; then $\mathcal{C}^{r+1} = \mathcal{C}^r, t^{r+t} = t$.

By construction $|T^{r+1}| \leqslant |T^r| - \lfloor |T^r|/3 \rfloor + 1$ or $|T^{r+1}| \leqslant 2|T^r|/3$. In both cases $|T^{r+1}| \leqslant 3|T^r|/4$ as long as $|T^r| \geqslant 100$, so $|T^{r+1}| \leqslant (3/4)^{r+1}\ell$. Fix the first r such that $|T^r| < 100$. Then

$$r \leqslant \lceil 2.5 \log \ell \rceil.$$

We strive for a contradiction as follows. Consider those < 100 leaves of T^r labeled by a clause C in OP_n or an axiom $\{X, \neg X\}$. Consecutively choose good expansions to satisfy all of them: if the clause has $\leqslant 3$ variables, choose a good extension that evaluates all of them by prolonging the current tuple by $\leqslant 6$ points; since good assignments cannot falsify clauses from OP_n or axioms, it is satisfying. For a yet unsatisfied clause $C_i := \{F_{i1}, \ldots F_{in}\} \in OP_n$, the current tuple ends in i or does not contain i; choose a new j and prolongue the tuple by j or i j, to satisfiy F_{ij} and hence the clause.

By (a) and (b), the result $\beta_{\bar{\imath}'}$ satisfies all clauses labeling leaves of T^r and $|\bar{\imath}'| < |\bar{\imath}_r| + 600$. By strong soundness (Lemma 1.6.5), $\beta_{\bar{\imath}'}$ satisfies $c(t^r)$. This contradicts (c).

This contradiction needs $\beta_{\bar{\imath}'}$ to be well-defined. It is if $|\bar{\imath}_r| + 600 \leqslant n$. As $|\bar{\imath}_r| \leqslant 2r$ we conclude $2r + 600 > n$, so $5 \log \ell + 2 + 600 > n$. This implies the theorem. □

1.8.2 Haken's theorem

Definition 1.8.7 (Pigeonhole principle). Let $n \in \mathbb{N}_{>0}$. Let X_{ij} be a variable for $i \in [n+1]$ and $j \in [n]$. PHP_n contains for every $i, i' \in [n+1], i \neq i'$, and $j \in [n]$ the clauses

$$\{X_{i1}, \ldots, X_{in}\}, \quad \{\neg X_{ij}, \neg X_{i'j}\}$$

Note PHP_n has $(n+1)n$ many variables and $O(n^3)$ many clauses. Intuitively, X_{ij} being true means "pigeon i flies to hole j". The clauses state that every pigeon flies to some hole, and no two pigeons occupy the same hole.

Proposition 1.8.8. *PHP_n is unsatisfiable for every $n \in \mathbb{N}_{>0}$.*

Proof. Assume $\beta \vDash PHP_n$. For every $i \in [n+1]$ choose $j \in [n]$ such that that $\beta(X_{ij}) = 1$ – it exists because $\beta \vDash \{X_{i1}, \dots, X_{in}\}$. Let $f : [n+1] \to [n]$ be the function mapping i to the chosen j. Then f is injective: otherwise there are $i, i' \in [n+1], i \neq i'$, and $j \in [n]$ such that $j = f(i) = f(i')$; then $\beta(X_{ij}) = \beta(X_{i'j}) = 1$, contradicting $\beta \vDash \{\neg X_{ij}, \neg X_{i'j}\}$. □

Theorem 1.8.9 (Haken)**.** *For all sufficiently large $n \in \mathbb{N}$, every Resolution refutation of PHP_n has length at least $2^{n/140}$.*

Proof. Informally, the proof can be outlined as follows: show that every refutation of length ℓ of PHP_n contains "n-wide" clauses. Find a "good" partial assignment β that satisfies all "n-wide" clauses and apply it to the given refutation. Being "good" ensures that the result is a refutation of PHP_m for some $m \leqslant n$. This refutations contains no clauses that are "n-wide". If m is sufficiently large, it contains no "m-wide" clauses, a contradiction. A "good" β yielding sufficiently large m exists if ℓ is small.

Let $n \in \mathbb{N}_{>0}$. A *matching* M is a bijection from a subset $dom(M) \subseteq [n+1]$ onto a subset $im(M) \subseteq [n]$. The partial assignment β_M is defined on X_{ij} if $i \in dom(M)$ or $j \in im(M)$; then it maps X_{ij} to 1 if $M(i) = j$, and to 0 otherwise.

Claim 1: Let M be a matching and $r := |dom(M)|$. Then $PHP_n \restriction \beta_M$ equals PHP_{n-r} up to renaming variables.

Proof. We show that the clauses in $PHP_n \restriction \beta_M$ are as in PHP_n but with i ranging over $[n+1] \setminus dom(M)$ and j ranging over $[n] \setminus im(M)$.

Suppose $\{X_{i1}, \dots, X_{in}\} \restriction \beta_M$ is in $PHP_n \restriction \beta_M$. Then β_M does not satisfy $\{X_{i1}, \dots, X_{in}\}$, so $i \notin dom(M)$. Then β_M falsifies X_{ij} if and only if $j \in im(M)$. Hence $\{X_{i1}, \dots, X_{in}\} \restriction \beta_M = \{X_{ij} \mid j \in [n] \setminus im(M)\}$.

Suppose $\{\neg X_{ij}, \neg X_{i'j}\} \restriction \beta_M$ is in $PHP_n \restriction \beta_M$. Then β_M does not satisfy $\{\neg X_{ij}, \neg X_{i'j}\}$. Then $j \notin im(M)$ and $i, i' \notin dom(M)$. ⊣

We are interested in assignments β_M for maximal matchings M. Note $im(M) = [n]$ and $dom(M) = [n+1] \setminus i_M$ for some $i_M \in [n+1]$. In particular, β_M is total. Observe $\beta_M \vDash \neg X_{ij}$ if and only if $\beta_M \vDash X_{ij'}$ for some $j' \neq j$. For a clause C let the clause $\tilde{C}$ contain X_{ij} if $X_{ij} \in C$ or $\neg X_{ij'} \in C$ for some $j \neq j'$. Then $\beta_M \vDash C$ if and only if $\beta_M \vDash \tilde{C}$. In particular, $C, \tilde{C}$ have the same supports: a set $S \subseteq [n+1]$ is a *support* of a clause C if $\beta_M \vDash C$ for every maximal matching M with $i_M \notin S$. Let $s(C)$ be the minimal size of a support of C.

Note $[n+1]$ is a support of any clause – trivially, because there are no matchings defined on $[n+1]$. It is the only support of the empty clause, so $s(\varnothing) = n+1$. An axiom $C := \{\neg X_{ij}, \neg X_{i'j}\}$ is satisfied by all β_M for maximal M, so $s(C) = 0$. An axiom $C := \{X_{ij}, \dots, X_{in}\}$ has support $\{i\}$ and $s(C) = 1$.

Claim 2: For every clause C we have $|\tilde{C}| \geqslant s(C)(n+1-s(C))$.

Proof. If $s(C) = n+1$ or $s(C) = 0$ there is nothing to show, so assume $0 < s(C) \leqslant n$ and let S be a support of C of size $s(C)$. Let $i \in S$ and $i' \in [n+1] \smallsetminus S$. Then there exists a maximal matching M with $i_M = i$ such that $\beta_M \nvDash C$, so $\beta_M \nvDash \tilde{C}$. Let M' be the maximal matching with $i_{M'} = i'$ and $M'(i) := M(i')$ and $M'(i'') = M(i'')$ for all $i'' \notin \{i, i''\}$. Then $S \subseteq \mathit{dom}(M')$, so $\beta_{M'} \vDash C$, so $\beta_{M'} \vDash \tilde{C}$. Hence there is a variable in $\tilde{C}$ mapped to 1 by M' but not by M. The only such variable is $X_{iM(i')}$, so $X_{iM(i')} \in \tilde{C}$.

Thus, $\tilde{C}$ contains the pairwise distinct variables $X_{iM(i')}$ for $(i, i') \in S \times ([n+1] \smallsetminus S)$. ⊣

Key Claim: Every refutation of PHP_n contains a clause C with $|\tilde{C}| > n^2/9$.

Proof. If C is a weakening of C', then $s(C) \leqslant s(C')$. If C is a cut of C', C'', or, more generally, $\{C', C''\} \vDash C$, then $s(C) \leqslant s(C') + s(C'')$: if S', S'' are supports of C', C'', then $S' \cup S''$ is a support of C. We already noted that clauses $C \in PHP_n$ have $s(C) \leqslant 1$ and $s(\varnothing) = [n+1]$. Let C be the first clause in the refutation with $s(C) \geqslant n/3$. Then C is a cut of clauses with smaller support, so $n/3 \leqslant s(C) < 2n/3$. By Claim 2, $s(C) > n^2/9$. ⊣

Let $C_1, \ldots, C_\ell = \varnothing$ be a refutation of PHP_n.

Let V_1 be the set of ($(n+1)n$ many) variables and $W_1 := \{\nu \in [\ell] \mid |\tilde{C}_\nu| \geqslant (n+1)n/100\}$. We claim that there exists $X_{i_1 j_1} \in V_1$ such that $X_{i_1 j_1} \in \tilde{C}_\nu$ for $\geqslant |W_1|/100$ many $\nu \in W_1$. Otherwise there are $< |V_1| \cdot |W_1|/100$ pairs (X, ν) such that $X \in V_1 \cap \tilde{C}_\nu$ and $\nu \in W_1$. On the other hand, there are $\geqslant |W_1| \cdot (n+1)n/100$ such pairs – a contradiction.

Let V_2 be the set of $X_{ij} \in V_1$ with $i_1 \neq i, j_1 \neq j$, and let W_2 be the set of $\nu \in W_1$ such that $\tilde{C}_\nu$ does not contain $X_{i_1 j_1}$ and contains $\geqslant (n+1)n/100$ variables in V_2. Note $|W_2| \leqslant 0.99|W_1|$. We claim that there exists $X_{i_2 j_2} \in V_2$ that $X_{i_2 j_2} \in \tilde{C}_\nu$ for $\geqslant |W_2|/100$ many $\nu \in W_2$. Otherwise there are $< |V_2| \cdot |W_2|/100 < (n+1)n|W_2|/100$ pairs (X, ν) such that $X \in V_2 \cap \tilde{C}_\nu$ and $\nu \in W_2$. On the other hand, there are $\geqslant |W_2| \cdot (n+1)n/100$ such pairs – a contradiction.

We continue like this producing $W_1, W_2, \ldots$ until $W_r = \varnothing$. Then $r \leqslant \log(\ell)/|\log(0.99)| < 70\log(\ell)$ because $|W_r| \leqslant 0.99^r \cdot |W_1| \leqslant 0.99^r \ell < 1$. Let M be the matching that maps i_1 to $j_1, \ldots$ and i_r to j_r, and is undefined elsewhere. Then for all $\nu \in W$ either $\beta_M \vDash C_\nu$ or $|\tilde{C}_\nu \upharpoonright \beta_M| < (n+1)n/100$. By the proof of Lemma 1.6.9, applying β_M to $C_1, \ldots, C_\ell$, we get a refutation of $PHP_n \upharpoonright \beta_M$ consisting of clauses $C_\nu \upharpoonright \beta_M$ for certain $\nu \in [\ell]$ with $|\tilde{C}_\nu \upharpoonright \beta_M| < (n+1)n/100$.

By Claim 1, this is a refutation of PHP_{n-r} up to a copy of variables. Up to copying, we can assume $\mathit{dom}(M) = \{n+1-r, \ldots, n+1\}$ and $\mathit{im}(M) = \{n-r, \ldots, n\}$ and we indeed have a refutation of PHP_{n-r}. Let $\hat{C}$ be defined as $\tilde{C}$ but in the variables of PHP_{n-r}. By the Key Claim (for $n-r$ instead of n and $\hat{}$ instead of $\tilde{}$), our refutation contains a clause D with $|\hat{D}| \geqslant (n-r)^2/9$. Say, $D = C_\nu \upharpoonright \beta_M$ where $\nu \in [\ell]$. It is easy to check that $\hat{D} \subseteq \tilde{C}_\nu \upharpoonright \beta_M$ (even $=$ holds). Thus $(n+1)n/100 > (n-r)^2/9$. If $\ell < 2^{n/140}$, we get a contradiction:

$$(n+1)n/100 > (n - 70\log\ell)^2/9 > (1 - 70/140)^2 n^2/9 \geqslant n^2/36. \qquad \square$$

Remark 1.8.10. Originally, Haken had 577 instead 140 but this constant is of no interest whatsoever and we make no effort to optimize it.

Chapter 2

First-order logic

This chapter introduces first-order logic and is structured similarly as the chapter on propositional logic. Following the outline in the Preface it defines syntax and semantics and then semantic concepts in the canonical way. It proceeds with Gentzen's sequent calculus and then Resolution. As a slight deviation, in order to establish some initial intuition, it starts defining *structures*, the "worlds" first-order logic talks about. Most, if not all, objects of study in mathematics are naturally viewed as structures. An important example from computer science is a *database*. In fact, it is Codd's seminal suggestion dating 1970 to identify databases with (typically finite) relational structures.

2.1 Structures

What is meant by the real numbers $\mathbb{R}$? Is it the set of reals, the ordered set of reals, the field of reals, the ordered field of reals or what? When defining an artificial language talking about $\mathbb{R}$ it is necessary to make a choice. This is the choice of a *language*:

Definition 2.1.1. A *language* L is a set of *relation symbols* and *function symbols*; each symbol has an associated *arity* $r \in \mathbb{N}$. Function symbols of arity 0 are *constants*.

An *L-structure* $\mathfrak{A}$ is a pair $(A, (s^{\mathfrak{A}})_{s \in L})$ where $A \neq \varnothing$ is the *universe* and $s^{\mathfrak{A}}$ is the *interpretation of* $s \in L$ *in* $\mathfrak{A}$:

– if $s = R \in L$ is an r-ary relation symbol, then $R^{\mathfrak{A}} \subseteq A^r$;

– if $s = f \in L$ is an r-ary function symbol, then $f^{\mathfrak{A}} : A^r \to A$.

If $c \in L$ is a constant, then $c^{\mathfrak{A}}$ maps the empty tuple to some $a \in A$: we identify $c^{\mathfrak{A}}$ with a.

Notation: If $L = \{s_1, \dots, s_n\}$ we write an L-structure $\mathfrak{A} = (A, (s^{\mathfrak{A}})_{s \in L})$ as $(A, s_1^{\mathfrak{A}}, \dots, s_n^{\mathfrak{A}})$.

Examples 2.1.2.

1. Let $L = \varnothing$. Then $\mathfrak{R} = (\mathbb{R})$ is the set of reals.

2. Let $L = \{<\}$ for a binary relation symbol $<$. Then $\mathfrak{R} = (\mathbb{R}, <^{\mathfrak{R}})$ is the ordered set of reals on $\mathbb{R}$ where $<^{\mathfrak{R}}$ is the usual order on $\mathbb{R}$.

3. Let $L = \{+, \cdot, -, 0, 1\}$ with $+, \cdot$ binary function symbols, $-$ a unary function symbol, and $0, 1$ constants. Then $\mathfrak{R} = (\mathbb{R}, +^{\mathfrak{R}}, \cdot^{\mathfrak{R}}, -^{\mathfrak{R}}, 0^{\mathfrak{R}}, 1^{\mathfrak{R}})$ is the *field of reals* where $+^{\mathfrak{R}}, \cdot^{\mathfrak{R}}$ are addition and multiplication on $\mathbb{R}$, $-^{\mathfrak{R}}$ maps $r \in \mathbb{R}$ to $-r$ (additive inverse), and $\underline{0}^{\mathfrak{R}} := 0, \underline{1}^{\mathfrak{R}} := 1$.

4. Let $L = \{+, \cdot, -, 0, 1. <\}$. Then $\mathfrak{R} = (\mathbb{R}, +^{\mathfrak{R}}, \cdot^{\mathfrak{R}}, -^{\mathfrak{R}}, \underline{0}^{\mathfrak{R}}, \underline{1}^{\mathfrak{R}} <^{\mathfrak{R}})$, with the interpretations defined as before, is the *ordered field of reals.*

Definition 2.1.3. Let L be a language, $L' \subseteq L$ and $\mathfrak{A}$ be an L-structure. The *L'-reduct* of $\mathfrak{A} = (A, (s^{\mathfrak{A}})_{s \in L})$ is the L'-structure

$$\mathfrak{A} \restriction L' := (A, (s^{\mathfrak{A}})_{s \in L'}).$$

Conversely, $\mathfrak{A}$ is an *L-expansion* of $\mathfrak{A} \restriction L'$.

Example 2.1.4. The *language of Peano arithmetic* is $L_{PA} := \{\underline{0}, S, +, \cdot, <\}$ where $\underline{0}, +, \cdot, <$ are as above and S is a unary function symbols. The *standard model of arithmetic* $\mathfrak{N}$ has universe $\mathbb{N}$ and interprets $\underline{0}, + \cdot, <$ by 0, addition, multiplication and order on $\mathbb{N}$, and S by the successor, i.e., $S^{\mathfrak{N}} : \mathbb{N} \to \mathbb{N}$ is given by $S^{\mathfrak{N}}(n) = n + 1$.

From now on we adapt our notation for (directed) graphs to the new formalism:

Example 2.1.5. Let E be a binary relation symbol. A *directed graph* is an $\{E\}$-structure $\mathfrak{G} = (G, E^{\mathfrak{G}})$ where $E^{\mathfrak{G}}$ is irreflexive (i.e., $(g, g) \notin E^{\mathfrak{G}}$ for all $g \in G$). $\mathfrak{G}$ is a *graph* if additionally $E^{\mathfrak{G}}$ is symmetric (i.e., $(g, g') \in E^{\mathfrak{G}}$ implies $(g', g) \in E^{\mathfrak{G}}$ for all $g, g' \in G$).

Here is a way how to see words as structures:

Example 2.1.6. Let A be an alphabet. Let $L_A := \{<\} \cup \{P_a \mid a \in A\}$ where $<$ is a binary relation symbol and the P_a are unary relation symbols. Let $w = a_1 \cdots a_n \in A^n$ be a word (over A) of length $n > 0$. The *word structure* $\mathfrak{S}(w)$ has universe $[n]$ and interprets $<$ by the usual order on $[n]$ and P_a by the occurrences of a in w, that is,

$$P_a^{\mathfrak{S}(w)} := \{i \in [n] \mid a_i = a\}.$$

Codd's relational model of databases works as follows.

Example 2.1.7 (Relational database)**.** L is *relational* if it contains only relation symbols. A database is given by tables like

ID	*Author*	*Title*	*Publisher*	*Year*
1	Codd	A Relational Model...	ACM Press	1970
2	Gutenberg	Printing for dummies	Mainzer	1452
3	Caesar	My life with Asterix	SPQR	-44
4	Darwin	Adam and Eve	Vatican Press	1862
5	Codd	Relational completeness...	Prentice-Hall	1972

Item	€
1	1.90
4	99.00
3	8.99

This is viewed as a $\{Works, Price\}$-structure $\mathfrak{D}$ where *Works* is a 5-ary relation symbol and *Price* a binary one. The universe D of $\mathfrak{D}$ consists of all cell entries, say as words over the usual alphabet and rational numbers. Rows display tuples in $Works^{\mathfrak{D}}$ and $Price^{\mathfrak{D}}$, e.g. (4, Darwin, Adam and Eve, Vatican Press, 1862) $\in Works^{\mathfrak{D}}$ and (1, 1.90) $\in Price^{\mathfrak{D}}$.

The column names *ID*, *Author*,... are called "attributes" and are not (but could) be incorporated in $\mathfrak{D}$; *ID* exemplifies a so-called "key" used to link entries across tables.

Example 2.1.8 (The world of efficient algorithms)**.** We define a structure $\mathfrak{E}$ with universe $\{0,1\}^*$ that interprets the language PV. PV contains an r-ary relation symbol $Q_{\mathbb{A}}$ for every efficient algorithm $\mathbb{A}$ that decides an r-ary relation $Q \subseteq (\{0,1\}^*)^r$. We let $Q_{\mathbb{A}}^{\mathfrak{E}} := Q$.

Further, PV contains a r-ary function symbol $f_{\mathbb{A}}$ for every efficient algorithm $\mathbb{A}$ that computes a function from $(\{0,1\}^*)^r$ to $\{0,1\}^*$. We let $f_{\mathbb{A}}^{\mathfrak{E}}$ be this function.

For example, there is a binary relation symbol $< \in PV$ such that $<^{\mathfrak{E}}$ contains precisely those pairs $(a,b) \in (\{0,1\}^*)^2$ such that a preceeds b in the *lexicographic order*: $\epsilon, 0, 1, 00, 01, 11, 000, \ldots$.

By Corollary 1.5.12 there is a unary relation symbol $Q \in PV$ such that $Q^{\mathfrak{E}}$ is Reachability, that is, $Q^{\mathfrak{E}}$ contains precisely those $a \in \{0,1\}^*$ that encode (in some reasonable sense) triples (G,s,t) such that G is a finite directed graph with a path from s to t.

Conceptually, two structures are "the same" if they are *isomorphic*:

Definition 2.1.9. Let L be a language and $\mathfrak{A}, \mathfrak{B}$ be L-structures with universes A, B. A function $\pi : A \to B$ is an *homomorphism from* $\mathfrak{A}$ *to* $\mathfrak{B}$, symbolically $\pi : \mathfrak{A} \to_h \mathfrak{B}$, if for all $r \in \mathbb{N}$, all r-ary relations symbols $R \in L$, all r-ary function symbols $f \in L$ and all $\bar{a} \in A^r$:

- $\bar{a} \in R^{\mathfrak{A}} \implies \pi(\bar{a}) \in R^{\mathfrak{B}}$;
- $\pi(f^{\mathfrak{A}}(\bar{a})) = f^{\mathfrak{B}}(\pi(\bar{a}))$.

Here, we write $\pi(\bar{a}) = (\pi(a_1), \ldots, \pi(a_r))$ for $\bar{a} = (a_1, \ldots, a_r) \in A^r$. If above we require $\iff$ instead of $\implies$, and π is injective, then it is an *(algebraic) embedding of* $\mathfrak{A}$ *into* $\mathfrak{B}$, symbolically $\pi : \mathfrak{A} \to_a \mathfrak{B}$.

If π is a bijective embedding, then π is an *isomorphism from* $\mathfrak{A}$ *onto* $\mathfrak{B}$, symbolically $\pi : \mathfrak{A} \cong \mathfrak{B}$. If such π exists, $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic*, symbolically $\mathfrak{A} \cong \mathfrak{B}$.

The reader has seen ad hoc definitions of homomorphisms of groups, rings, fields, graphs and what not. We trust she notices that the above unifies all these notions.

Exercise 2.1.10. Let K be a field and $L := \{+, -, 0\} \cup K$ with binary $+$, unary $-$, constant 0 and every $\lambda \in K$ a unary function symbol. K-vectorspaces are L-structures in the straightforward way. Show that homomorphism means linear map.

Exercise 2.1.11. Show that a graph $\mathfrak{G}'$ is isomorphic to an (induced) subgraph of a graph $\mathfrak{G}$ if and only if there is an injective homomorphism (an embedding) of $\mathfrak{G}'$ into $\mathfrak{G}$.

Exercise 2.1.12. Let $\mathit{bin}: \mathbb{N} \to \{0,1\}^*$ map $n \in \mathbb{N}$ to its binary expansion $b_1 \cdots b_\ell$ where $\ell := \lceil \log(n+1) \rceil$ and $n = \sum_{i=1}^{\ell} b_i \cdot 2^{\ell - i}$. Conversely, let $\mathit{num}: \{0,1\}^* \to \mathbb{N}$ map $a = a_1 \cdots a_{|a|} \in \{0,1\}^*$ to $n \in \mathbb{N}$ such that $\mathit{bin}(n) = 1a_1 \cdots a_{|a|}$.

Show that $\mathit{bin}: \mathfrak{N} \to_a \mathfrak{E} \upharpoonright L$ and $\mathit{num}: \mathfrak{E} \upharpoonright L \to_a \mathfrak{N}$ for a suitable $L \subseteq PV$.

We now unify the notions of induced subgraph, subgroup, subring, subfield,...

Definition 2.1.13. Let L be a language and $\mathfrak{A}, \mathfrak{B}$ be L-structures with universes A, B. $\mathfrak{A}$ is a *substructure of* $\mathfrak{B}$ and $\mathfrak{B}$ is an *extension of* $\mathfrak{A}$, symbolically $\mathfrak{A} \subseteq \mathfrak{B}$, if $A \subseteq B$ and for all $r \in \mathbb{N}$, all r-ary relations symbols $R \in L$, all r-ary function symbols $f \in L$:

- $R^{\mathfrak{A}} = R^{\mathfrak{B}} \cap A^r$;
- $f^{\mathfrak{A}} = f^{\mathfrak{B}} \upharpoonright A^r$.

Exercise 2.1.14. The following are equivalent.

1. There is an embedding of $\mathfrak{A}$ into $\mathfrak{B}$.
2. $\mathfrak{A}$ is isomorphic to a substructure of $\mathfrak{B}$.
3. $\mathfrak{B}$ is isomorphic to an extension of $\mathfrak{A}$.

2.2 Syntax

Let L be a language. We define first-order formulas as words over the alphabet

$$), \ (, \ \wedge, \ \neg, \exists, \dot{=}, x_0, x_1, \ldots$$

plus the symbols in L; $\mathit{Var} := \{x_0, x_1 \ldots\}$ is the set of *(individual) variables.*

Definition 2.2.1 (Syntax: terms)**.** The set of L-*terms* is the smallest set T of words such that $\mathit{Var} \subseteq T$ and

if $r \in \mathbb{N}$ and $f \in L$ is an r-ary function symbol and $t_1 \ldots t_r \in T$, then $ft_1 \cdots t_r \in T$.

In particular, if $c \in L$ is a constant, then c is an L-term.

Notation: for better readability we sometimes use *infix notation* for binary function symbols $f \in L$, that is, we write $(t \ f \ t')$ instead of ftt'.

Example 2.2.2. In the language L_{PA} of Peano arithmetic, e.g.

$$\cdot + x_0 x_1 x_2, \quad \cdot + \underline{0} x_1 + x_5 S x_1$$

are L_{PA}-terms. Using infix notation they read

$$((x_0 + x_1) \cdot x_2), \quad ((\underline{0} + x_1) \cdot (x_5 + Sx_1)).$$

Exercise 2.2.3. Find 4 different ways to assign arities to functions symbols $+, \cdot$ such that $\cdot + x_0 x_1 x_2$ is an $\{+, \cdot\}$-term. For a certain choice of arities, can $++$ or $\cdot ++$ be $\{+, \cdot\}$-terms?

Lemma 2.2.4 (Unique readability of terms)**.** *For every L-term t, either $t = x$ for some $x \in \mathit{Var}$ or $t = ft_1 \cdots t_r$ for some $r \in \mathbb{N}$, some r-ary functions symbol $f \in L$ and L-terms $t_1, \dots, t_r$. In the second case, r, f and $t_1, \dots, t_r$ are uniquely determined.*

Proof. It is easy to see that exactly one case holds. For uniqueness assume

$$ft_1 \cdots t_r = gu_1 \cdots u_s$$

for $f, g \in L$ function symbols and L-terms t_i, u_j. Then $f = g$, so $r = s$, and it suffices to show $t_1 = u_1, \dots, t_r = u_r$. This follows from the

Claim: no proper prefix of an L-term is an L-term.

Assume there is an L-term t with a proper prefix t' that is an L-term. Choose such t of minimal length, and a witnessing t' for it. Then t is not a variable nor a constant since t' would then be the empty word and this is not an L-term. Hence $t = ft_1 \cdots t_r$ with $r > 0$. Then $t' = ft'_1 \cdots t'_r$. Since $t \neq t'$ there is a minimal $i \in [r]$ such that $t_i \neq t'_i$. Then t_i is a proper prefix of t'_i or vice-versa. But both t_i, t'_i are shorter than t, a contradiction. □

Definition 2.2.5 (Syntax: formulas)**.** The set of *L-formulas* is the smallest set F of words such that

(F1) if t_0, t_1 are L-terms, then $t_0 \doteq t_1 \in F$;

(F2) if $r \in \mathbb{N}$ and $R \in L$ is an r-ary relation symbol and $t_1, \dots, t_r$ are L-terms, then $Rt_1 \cdots t_r \in F$;

(F3) if $\varphi \in F$, then $\neg\varphi \in F$;

(F4) if $\varphi, \psi \in F$, then $(\varphi \wedge \psi) \in F$;

(F5) if $\varphi \in F$ and $x \in \mathit{Var}$, then $\exists x \varphi \in F$.

Lemma 2.2.6 (Unique readability of formulas)**.** *For every L-formula φ exactly one of the following holds.*

1. *$\varphi = t_0 \doteq t_1$ for some L-terms t_0, t_1; then φ is* atomic.
2. *$\varphi = Rt_1 \cdots t_r$ for some $r \in \mathbb{N}$, some r-ary relation symbol $R \in L$ and L-terms $t_1, \dots, t_r$; then φ is* atomic.
3. *$\varphi = \neg\psi$ for some L-formula ψ; then φ is a* negation (of ψ)*;*
4. *$\varphi = (\psi \wedge \chi)$ for some L-formulas ψ, χ; then φ is a* conjunction (of ψ and χ)*;*
5. *$\varphi = \exists x \psi$ for some $x \in \mathit{Var}$ and some L-formula ψ.*

Moreover, in case 1, t_0, t_1 are uniquely determined; in case 2, $r, R, t_1, \dots, t_r$ are uniquely determined; in case 3, ψ is uniquely determined; in case 4, ψ, χ are uniquely determined; in case 5, ψ is uniquely determined.

Exercise 2.2.7. Prove this.

2.3 Semantics

Let L be a language and $\mathfrak{A}$ an L-structure with universe A.

2.3.1 Semantics of terms

Definition 2.3.1. An *assignment (in $\mathfrak{A}$)* is a function $\beta : \mathit{Var} \to A$. For an L-term t we define $t^{\mathfrak{A}}[\beta]$, the *value of t in $\mathfrak{A}$ under β*, by recursion on t:

– if $t = x$ for some$x \in \mathit{Var}$, then $t^{\mathfrak{A}} := \beta(t)$;

– if $t = ft_1\cdots t_r$ with $r \in \mathbb{N}$, $f \in L$ an r-ary function symbol, and L-terms $t_1, \dots, t_r$, then

$$t^{\mathfrak{A}}[\beta] := f^{\mathfrak{A}}\big(t_1^{\mathfrak{A}}[\beta], \dots, t_r^{\mathfrak{A}}[\beta]\big).$$

As usual this definition should be read as a specification of a recursive algorithm.

Example 2.3.2. Let $\mathfrak{N}$ be the standard model of arithmetic and β given by $\beta(x_i) := i+1$.

$$\begin{array}{rcl}(+x_0 \cdot x_1x_2)^{\mathfrak{N}}[\beta] & = & +^{\mathfrak{N}}(x_0^{\mathfrak{N}}[\beta], (\cdot x_1x_2)^{\mathfrak{N}}[\beta]) = +^{\mathfrak{N}}(\beta(x_0), \cdot^{\mathfrak{N}}(x_1^{\mathfrak{N}}[\beta], x_2^{\mathfrak{N}}[\beta])) \\ & = & +^{\mathfrak{N}}(1, \cdot^{\mathfrak{N}}(\beta(x_1), \beta(x_2))) = +^{\mathfrak{N}}(1, \cdot^{\mathfrak{N}}(2,3)) = 7\end{array}$$

Lemma 2.3.3 (Coincidence lemma for terms)**.** *Let t be an L-term and let $V \subseteq \mathit{Var}$ contain all variables occurring in t. Let β, γ be assignments in $\mathfrak{A}$ such that $\beta{\restriction}V = \gamma{\restriction}V$. Then*

$$t^{\mathfrak{A}}[\beta] = t^{\mathfrak{A}}[\gamma].$$

Proof. We proceed by induction on t, that is, we show:

1. the claim holds for variables;
2. if $r \in \mathbb{N}$ and $f \in L$ is an r-ary function symbol and the claim holds for $t_1, \dots, t_r$, then the claim holds for $ft_1\cdots t_r$.

1 is obvious. For 2 argue

$$(ft_1\cdots t_r)^{\mathfrak{A}}[\beta] = f^{\mathfrak{A}}\big(t_1^{\mathfrak{A}}[\beta], \dots, t_r^{\mathfrak{A}}[\beta]\big) = f^{\mathfrak{A}}\big(t_1^{\mathfrak{A}}[\gamma], \dots, t_r^{\mathfrak{A}}[\gamma]\big) = (ft_1\cdots t_r)^{\mathfrak{A}}[\gamma],$$

where the middle equality follows because γ, β agree on the variables in the t_i. □

Definition 2.3.4. An L-term is *closed* if no variable occurs in it.

Notation: if t is closed, the value $t^{\mathfrak{A}}[\beta]$ does not depend on β and we write $t^{\mathfrak{A}}$ for it. Writing $t = t(x_1 \dots, x_k)$ means the variables in t are among $x_1, \dots, x_k$. We set

$$t^{\mathfrak{A}}[a_1, \dots, a_k] := t^{\mathfrak{A}}[\beta]$$

where β is an assignment in $\mathfrak{A}$ with $\beta(x_1) = a_1, \dots, \beta(x_k) = a_k$. Note this is well-defined, i.e., independent from the choice of β, by the coincidence lemma.

For $\bar{a} = a_1\cdots a_k \in A^k$ write $\beta[\bar{x}/\bar{a}]$ for the assignment that maps x_i to a_i and every other variable y to $\beta(y)$. Note $t^{\mathfrak{A}}[\bar{a}] = t^{\mathfrak{A}}[\beta[\bar{x}/\bar{a}]]$ for every assignment β.

Lemma 2.3.5. *Assume $\mathfrak{B} \subseteq \mathfrak{A}$. Then $t^{\mathfrak{B}}[\beta] = t^{\mathfrak{A}}[\beta]$ for every L-term t and every assignment β in $\mathfrak{B}$.*

Proof. If $t = x$ is a variable, then $t^{\mathfrak{B}}[\beta] = \beta(x) = t^{\mathfrak{A}}[\beta]$. If $t = ft_1 \cdots t_r$ and the clai holds for the t_i, then $t^{\mathfrak{B}}[\beta] = f^{\mathfrak{B}}(t_1^{\mathfrak{B}}[\beta], \dots, t_r^{\mathfrak{B}}[\beta]) = f^{\mathfrak{A}}(t_1^{\mathfrak{A}}[\beta], \dots, t_r^{\mathfrak{A}}[\beta]) = t^{\mathfrak{A}}[\beta]$ where the replacement of $f^{\mathfrak{B}}$ by $f^{\mathfrak{A}}$ uses $\mathfrak{B} \subseteq \mathfrak{A}$. □

Exercise 2.3.6. Assume $\mathfrak{B}$ is an L-structure and $\pi : \mathfrak{B} \to_h \mathfrak{A}$ a homomorphism. Then for every L-term t and every assignment β in $\mathfrak{B}$: $\pi(t^{\mathfrak{B}}[\beta]) = t^{\mathfrak{A}}[\pi \circ \beta]$.

Lemma 2.3.7. *Let $\mathfrak{A}_0 \subseteq A$ and assume $A_0 \neq \emptyset$ or L contains a constant. Then there is a unique substructure $\mathfrak{B} \subseteq \mathfrak{A}$ with universe*

$$B = \left\{ t^{\mathfrak{A}}[\beta] \mid t \text{ } L\text{-term }, \beta : \mathit{Var} \to A_0 \right\}.$$

Further $\mathfrak{B} \subseteq \mathfrak{C}$ for every $\mathfrak{C} \subseteq \mathfrak{A}$ with universe $C \supseteq A_0$. We write $\langle A_0 \rangle^{\mathfrak{A}}$ for $\mathfrak{B}$ and say $\mathfrak{B}$ is generated by A_0 in $\mathfrak{A}$.

Proof. Uniqueness is clear: every $B \subseteq A$ is the universe of at most one substructure of $\mathfrak{A}$, namely the one with interpretations given by the restrictions of the interpretations in $\mathfrak{A}$. For existence we only need that these restrictions "live" on B: $f^{\mathfrak{A}} : B^r \to B$ for every $r \in N$ and every r-ary function symbol $f \in L$. Let $b_1, \dots, b_r \in B$, say $b_i = t_i^{\mathfrak{A}}[\beta_i]$. We can assume the t_i have disjoint variables (otherwise make copies). Then there is β that agrees with all β_i on the variables in t_i. By the coincidence lemma for terms, $t_i^{\mathfrak{A}}[\beta_i] = t_i^{\mathfrak{A}}[\beta]$. Then $f^{\mathfrak{A}}(b_1, \dots, b_r) = t^{\mathfrak{A}}[\beta]$ for $t := ft_1 \cdots t_r$. Hence, $f^{\mathfrak{A}}(b_1, \dots, b_r) \in B$.

Given $\mathfrak{C} \subseteq \mathfrak{A}$ with $C \supseteq A_0$, first note $B \subseteq C$: given $b \in B$ choose suitable t, β such that $b = t^{\mathfrak{A}}[\beta]$; then $b = t^{\mathfrak{C}}[\beta]$ by the coincidence lemma for terms; hence, $b \in C$.

Then $\mathfrak{B} \subseteq \mathfrak{C}$ follows easily. E.g., for an r-ary relation symbol $R \in L$ we have $R^{\mathfrak{B}} = R^{\mathfrak{A}} \cap B^r = (R^{\mathfrak{A}} \cap C^r) \cap B^r = R^{\mathfrak{C}} \cap B^r$. □

2.3.2 Semantics of formulas

We define truth by *Tarski's T-conditions* for first-order logic.

Definition 2.3.8. For an L-formula φ and an assignment β in $\mathfrak{A}$ we define β *satisfies* φ *in* $\mathfrak{A}$, or φ *is true in* $\mathfrak{A}$ *under* β, symbolically $\mathfrak{A} \models \varphi[\beta]$ by recursion on φ:

(T1) if $\varphi = t_0 \doteq t_1$ for L-terms t_0, t_1, then:

$$\mathfrak{A} \models \varphi[\beta] \iff t_0^{\mathfrak{A}}[\beta] = t_1^{\mathfrak{A}}[\beta];$$

(T2) if $\varphi = Rt_r \cdots t_r$ for $r \in \mathbb{N}$, $R \in L$ an r-ary relation symbol and L-terms $t_1, \cdots, t_r$, then:

$$\mathfrak{A} \models \varphi[\beta] \iff (t_1^{\mathfrak{A}}[\beta], \dots, t_r^{\mathfrak{A}}[\beta]) \in R^{\mathfrak{A}};$$

(T3) if $\varphi = \neg\psi$ for some L-formula ψ, then:

$$\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{A} \not\models \psi[\beta];$$

(T3) if $\varphi = (\psi \wedge \chi)$ for L-formulas ψ, χ, then:

$$\mathfrak{A} \models \varphi[\beta] \iff \text{both } \mathfrak{A} \models \psi[\beta] \text{ and } \mathfrak{A} \models \chi[\beta];$$

(T4) if $\varphi = \exists x\psi$ for some $x \in \mathit{Var}$ and L-formula ψ, then:

$$\mathfrak{A} \models \varphi[\beta] \iff \text{there is } a \in A\text{: } \mathfrak{A} \models \psi\big[\beta[x/a]\big].$$

Notation: for L-formulas φ, ψ the abbreviations $(\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi)$ are defined as in propositional logic. Further,

$$\forall x\varphi := \neg\exists x\neg\varphi.$$

Then $\mathfrak{A} \models \forall x\varphi[\beta]$ if and only if for all $a \in A$: $\mathfrak{A} \models \varphi\big[\beta[x/a]\big]$.

Example 2.3.9. Let $\mathfrak{G} = (G, E^{\mathfrak{G}})$ be a graph, β an assignment in $\mathfrak{G}$ and

$$\varphi = \exists x_0\exists x_1(Ex_0x_1 \wedge \exists x_2(Ex_0x_2 \wedge Ex_1x_2)).$$

The following are equivalent to $\mathfrak{G} \models \varphi[\beta]$.

there are $g_0, g_1 \in G$: $\mathfrak{G} \models (Ex_0x_1 \wedge \exists x_2(Ex_0x_2 \wedge Ex_1x_2))[\beta[x_0x_1/g_0g_1]]$
there are $g_0, g_1 \in G$: $\mathfrak{G} \models Ex_0x_1[\beta[x_0x_1/g_0g_1]]$
  and $\mathfrak{G} \models \exists x_2(Ex_0x_2 \wedge Ex_1x_2)[\beta[x_0x_1/g_0g_1]]$
there are $g_0, g_1 \in G$: $(\beta[x_0x_1/g_0g_1](x_0), \beta[x_0x_1/g_0g_1](x_1)) \in E^{\mathfrak{G}}$
  and there is $g_2 \in G$: $\mathfrak{G} \models (Ex_0x_2 \wedge Ex_1x_2)[\gamma]$ for $\gamma := \beta[x_0x_1x_2/g_0g_1g_2]$
there are $g_0, g_1, g_2 \in G$: $(g_0, g_1) \in E^{\mathfrak{G}}$
  and $(\gamma(x_0), \gamma(x_2)), (\gamma(x_1), \gamma(x_2)) \in E^{\mathfrak{G}}$
there are $g_0, g_1, g_2 \in G$: $(g_0, g_1), (g_0, g_2), (g_1, g_2) \in E^{\mathfrak{G}}$
$\mathfrak{G}$ contains a triangle.

Note $\mathfrak{G} \models \varphi[\beta]$ is independent of β. The reason is that all variables are quantified.

Definition 2.3.10. For every L-formula define the set *free*(φ) of *free variables of* φ by recursion on φ:

- if φ is atomic, then *free*(φ) is the set of variables occurring in φ;
- if $\varphi = \neg\psi$ for some ψ, then *free*$(\varphi) :=$ *free*(ψ);
- if $\varphi = (\psi \wedge \chi)$ for some ψ, χ, then *free*$(\varphi) :=$ *free*$(\psi) \cup$ *free*(χ);
- if $\varphi = \exists x\psi$ for some ψ and $x \in \mathit{Var}$, then *free*$(\varphi) :=$ *free*$(\psi) \setminus \{x\}$.

Example 2.3.11. Let E be a binary relations symbol and $\varphi = (\exists x_0 Ex_0x_1 \wedge \exists x_2 Ex_1x_0)$.

free(φ) = *free*$(\exists x_0 Ex_0x_1) \cup$ *free*$(\exists x_2 Ex_1x_0) = (\{x_0,x_1\} \setminus \{x_0\}) \cup (\{x_1,x_0\} \setminus \{x_2\}) = \{x_0,x_1\}$.

Lemma 2.3.12 (First coincidence lemma). *Let φ be an L-formula and β,γ assignments in $\mathfrak{A}$ such that $\beta\restriction$free$(\varphi) = \gamma\restriction$free$(\varphi)$. Then*

$$\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{A} \models \varphi[\gamma].$$

Proof. We proceed by induction on φ, that is, we show:

1. the claim holds for atomic L-formulas;
2. if the claim holds for ψ, then also for $\neg\psi$;
3. if the claim holds for ψ,χ, then also for $(\psi \wedge \chi)$;
4. if the claim holds for ψ, then also for $\exists x\psi$.

For 1, consider an atomic L-formula of the form $Rt_1\cdots t_r$. Then $t_i^{\mathfrak{A}}[\beta] = t_i^{\mathfrak{A}}[\gamma]$ by the coincidence lemma for terms, so $\mathfrak{A} \models Rt_1\cdots t_r[\beta]$ and $\mathfrak{A} \models Rt_1\cdots t_r[\beta]$ are equivalent to $(t_1^{\mathfrak{A}}[\beta],\dots,t_r^{\mathfrak{A}}[\beta]) = (t_1^{\mathfrak{A}}[\gamma],\dots,t_r^{\mathfrak{A}}[\gamma]) \in R^{\mathfrak{A}}$. The case $t_0 \doteq t_1$ is similar.

2 and 3 are trivial. For 4, let $\varphi = \exists\psi$ and argue

$$\begin{aligned}\mathfrak{A} \models \varphi[\gamma] &\iff \text{there is } a \in A\text{: } \mathfrak{A} \models \psi[\beta[x/a]]\\ &\iff \text{there is } a \in A\text{: } \mathfrak{A} \models \psi[\gamma[x/a]] \iff \mathfrak{A} \models \varphi[\gamma];\end{aligned}$$

for the middle equivalence note that $\gamma[x/a],\beta[x/a]$ agree on *free*$(\psi) \subseteq$ *free*$(\varphi) \cup \{x\}$ if β,γ agree on *free*(φ) and we assume the claim holds for ψ. □

Lemma 2.3.13 (Second coincidence lemma). *Let $L' \subseteq L$ and φ be an L'-formula and β and assignment in $\mathfrak{A}$. Then*

$$\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{A}\restriction L' \models \varphi[\beta]$$

Proof. A straightforward induction on terms shows $t^{\mathfrak{A}}[\beta] = t^{\mathfrak{A}\restriction L'}[\beta]$. The claim then follows for atomic φ. The cases $\varphi = \neg\psi, (\psi \wedge \chi)$ are easy. The case $\varphi = \exists x\psi$ is also easy:

$$\begin{aligned}\mathfrak{A} \models \varphi[\beta] &\iff \text{there is } a \in A : \mathfrak{A} \models \psi[\beta[x/a]]\\ &\iff \text{there is } a \in A : \mathfrak{A}\restriction L' \models \psi[\beta[x/a]] \iff \mathfrak{A}\restriction L' \models \varphi[\beta]\end{aligned}$$

where the middle equivalence follows from the inductive assumption that our claim holds for ψ (and all assignments). □

Notation: Writing $\varphi = \varphi(x_1,\dots,x_k)$ means *free*$(\varphi) \subseteq \{x_1,\dots,x_k\}$. We say $\bar{a} = (a_1,\cdots,a_k) \in A^k$ *satisfies* $\varphi(\bar{x})$ in $\mathfrak{A}$ and write

$$\mathfrak{A} \models \varphi[a_1,\dots,a_k]$$

to mean $\mathfrak{A} \models \varphi[\beta[\bar{x}/\bar{a}]]$ for some assignment β in $\mathfrak{A}$.

Remark 2.3.14. In database theory, first-order logic is referred to as the *relational calculus*. A notational variant is the *relational algebra*. Many declarative languages to specify database queries have been suggested, and those translatable to first-order logic are called *Codd complete*. The most important one is SQL and has higher expressive power. SQL can express connectness of graphs and we shall see later that first-order logic can't (Example 2.6.18). Another important query language is DATALOG. It can express connectedness but not all first-order queries.

Exercise 2.3.15. Recall $\mathfrak{D}$ from Example 2.1.7. Write formulas $\varphi(x), \psi(x)$ such that $\varphi(x)$ is satisfied in $\mathfrak{D}$ by precisely the authors of books without a price, and $\psi(x)$ by those who wrote more than one book. Expand $\mathfrak{D}$ by interpreting a binary relation symbol $<$ by the usual order $<^{\mathfrak{D}}$ on the rational numbers in D. Write a formula $\psi(x)$ satisfied by exactly the title of the most expensive book.

Exercise 2.3.16. Write an L_{PA}-formula $\varphi(x,y)$ such that $(p,q) \in \mathbb{N}^2$ satisfies $\varphi(x,y)$ in $\mathfrak{N}$ if and only if (p,q) is a prime twin. Write an L_{PA}-sentence that is true in $\mathfrak{N}$ if and only if Goldbach's conjecture is true.

Definition 2.3.17. An *L-sentence* is an L-formula φ with *free*$(\varphi) = \varnothing$. Then we say $\mathfrak{A}$ *satisfies* φ or φ *is true in* $\mathfrak{A}$ or $\mathfrak{A}$ *is a model of* φ, symbolically

$$\mathfrak{A} \models \varphi,$$

if $\mathfrak{A} \models \varphi[\beta]$ for some (equivalently, every) assignment β in $\mathfrak{A}$. A *theory* T is a set of L-sentences. $\mathfrak{A}$ is a *model of* T if $\mathfrak{A} \models T$, i.e., $\mathfrak{A} \models \varphi$ for all $\varphi \in T$.

Exercise 2.3.18. Recall $L = \{+,-,0\} \cup K$ from Example 2.1.10. Write an L-theory T whose models are precisely the K-vectorspaces.

Exercise 2.3.19 (Isomorphism lemma)**.** Let $\mathfrak{A}, \mathfrak{B}$ be L-structures and $\pi : \mathfrak{A} \cong \mathfrak{B}$. For all L-formulas and assignments β in $\mathfrak{A}$: $\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{B} \models \varphi[\pi \circ \beta]$.

Definition 2.3.20. An L-formula is *quantifier free* if $\exists$ does not occur in it.

A *universal* L-formula has the form $\forall \bar{x}\varphi$ where $\bar{x} = x_1 \cdots x_k$ is a k-tuple of variables for some $k \in \mathbb{N}$ and φ is quantifier free; here we write $\forall \bar{x}$ instead $\forall x_1 \cdots \forall x_k$.

Similarly, an *existential* L-formula is one of the form $\exists \bar{x}\varphi$ for quantifier free φ.

Exercise 2.3.21. Let $\mathfrak{A} \subseteq \mathfrak{B}$ and β be an assignment in $\mathfrak{A}$. Show:

1. If φ is quantifier free, then: $\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{B} \models \varphi[\beta]$.
2. If φ is universal, then: $\mathfrak{B} \models \varphi[\beta] \implies \mathfrak{A} \models \varphi[\beta]$.
3. If φ is existential, then: $\mathfrak{A} \models \varphi[\beta] \implies \mathfrak{B} \models \varphi[\beta]$.

Exercise 2.3.22. Let φ be a universal L-sentence. Then $\mathfrak{A} \models \varphi$ if and only if $\langle A_0 \rangle^{\mathfrak{A}} \models \varphi$ for all finite $\varnothing \neq A_0 \subseteq A$.

2.4 Semantic concepts

Let L be a language.

Definition 2.4.1. Let φ, ψ be formulas and Φ a set of formulas.

1. φ is *satisfiable* if $\mathfrak{A} \models \varphi[\beta]$ for some L-structure $\mathfrak{A}$ and some assignment β in $\mathfrak{A}$.
2. φ is *valid* if $\mathfrak{A} \models \varphi[\beta]$ for every L-structure $\mathfrak{A}$ and every assignment β in $\mathfrak{A}$
3. φ, ψ are *(logically) equivalent*, symbolically $\varphi \equiv \psi$, if $(\varphi \leftrightarrow \psi)$ is valid.
4. Φ is *satisfiable* if there exists an L-structure $\mathfrak{A}$ and an assignment β in $\mathfrak{A}$ such that $\mathfrak{A} \models \Phi[\beta]$, i.e., $\mathfrak{A} \models \chi[\beta]$ for all $\chi \in \Phi$.
5. Φ *(logically) implies* φ, symbolically $\Phi \models \varphi$, if $\mathfrak{A} \models \varphi[\beta]$ for every L-structure $\mathfrak{A}$ and every assignments β in $\mathfrak{A}$ with $\mathfrak{A} \models \Phi[\beta]$.

Remark 2.4.2.

1. φ is valid if and only if $\neg\varphi$ is unsatisfiable.
2. $\Phi \cup \{\varphi\} \models \psi$ if and only if $\Phi \models (\varphi \to \psi)$.
3. $\Phi \models \varphi$ if and only if $\Phi \cup \{\neg\varphi\}$ is unsatisfiable.

Example 2.4.3. Let φ, ψ be L-formulas and $z \notin \mathit{free}(\varphi)$. Then

$$\begin{aligned}
&\neg\exists x\varphi \equiv \forall x\neg\varphi,\ \neg\forall x\varphi \equiv \exists x\neg\varphi \\
&\exists x(\varphi \vee \psi) \equiv (\exists x\varphi \vee \exists x\psi),\ \forall x(\varphi \wedge \psi) \equiv (\forall x\varphi \wedge \forall x\psi), \\
&(\varphi \wedge \exists z\psi) \equiv \exists z(\varphi \wedge \psi),\ (\varphi \vee \exists z\psi) \equiv \exists z(\varphi \vee \psi), \\
&(\varphi \wedge \forall z\psi) \equiv \forall z(\varphi \wedge \psi),\ (\varphi \vee \forall z\psi) \equiv \forall z(\varphi \vee \psi).
\end{aligned}$$

2.4.1 Prenex forms

Definition 2.4.4. An L-formula is *prenex* if it has the form

$$Q_1x_1 \cdots Q_kx_k\varphi$$

where $k \in \mathbb{N}$, the x_i are pairwise distinct, the Q_i are $\exists$ or $\forall$, and φ is quantifier free.

Proposition 2.4.5. *Every L-formula is equivalent to a prenex one.*

Proof. We proceed by induction on φ. Abbreviate $Q_1y_1 \cdots Q_ky_k$ by $\bar{Q}\bar{y}$. If φ is atomic, then it is prenex (with $k = 0$). If $\varphi(\bar{x}) = \neg\psi(\bar{x})$, by induction $\psi(\bar{x}) \equiv \bar{Q}\bar{y}\chi(\bar{x}, \bar{y})$ for some quantifier free $\chi(\bar{x}, \bar{y})$. Then $\varphi(\bar{x}) \equiv \neg\bar{Q}\bar{y}\chi(\bar{x}, \bar{y})$. Let $\bar{Q}'\bar{y}$ be obtained from $\bar{Q}\bar{y}$ by swapping $\exists/\forall$. Then $\varphi(\bar{x}) \equiv \bar{Q}'\bar{y}\neg\chi(\bar{x}, \bar{y})$ follows using row 1 in Example 2.4.3.

If $\varphi(\bar{x}) = (\psi_0(\bar{x}) \wedge \psi_1(\bar{x}))$, suppose $\psi_0(\bar{x}) \equiv \bar{Q}_0\bar{y}_0\chi_0(\bar{x}, \bar{y}_0)$ and $\psi_1(\bar{x}) \equiv \bar{Q}_1\bar{y}_1\chi_1(\bar{x}, \bar{y}_1)$. Let $\bar{y}_0', \bar{y}_1'$ be copies of $\bar{y}_0, \bar{y}_1$ consisting of variables that do not occur in the formulas. Let $\chi_1(\bar{x}, \bar{y}_1')$ be obtained form $\chi_1(\bar{x}, \bar{y}_1)$ by replacing the variables $\bar{y}_1$ by $\bar{y}_1'$. It is easy to check that $\bar{Q}_1\bar{y}_1\chi_1(\bar{x}, \bar{y}_1) \equiv \bar{Q}_1\bar{y}_1'\chi_1(\bar{x}, \bar{y}_1')$. Similarly, $\bar{Q}_0\bar{y}_0\chi_0(\bar{x}, \bar{y}_0) \equiv \bar{Q}_0\bar{y}_0'\chi_0(\bar{x}, \bar{y}_0')$. Then $\varphi(\bar{x}) \equiv \bar{Q}_0\bar{y}_0'\bar{Q}_1\bar{y}_1'(\chi_0(\bar{x}, \bar{y}_0') \wedge \chi_1(\bar{x}, \bar{y}_1'))$ follows using rows 3 and 4 in Example 2.4.3. □

Example 2.4.6. This proof implicitly specifies an efficient recursive algorithm computing equivalent prenex formulas. Given $((\exists y Exy \wedge \neg\exists x Exy) \wedge \neg\forall y Eyx)$ it works as follows:

$$\begin{aligned}
&(\exists y Exy \wedge \forall x \neg Exy) \wedge \exists y \neg Eyx)\\
&(\exists y' Exy' \wedge \forall x' \neg Ex'y) \wedge \exists y \neg Eyx)\\
&(\exists y' \forall x'(Exy' \wedge \neg Ex'y) \wedge \exists y \neg Eyx)\\
&(\exists y' \forall x'(Exy' \wedge \neg Ex'y) \wedge \exists y'' \neg Ey''x)\\
&\exists y' \forall x' \exists y''((Exy' \wedge \neg Ex'y) \wedge \neg Ey''x)
\end{aligned}$$

2.4.2 Tautologies

We now aim to show that equivalences observed for propositional logic continue to hold. More precisely, e.g. the equivalences in Example 1.3.4 hold for first-order formulas φ, ψ, χ.

Definition 2.4.7. An L-formula φ is a *tautology* if there is a propositional tautology α, say in propositional variables $\bar{X} := X_1 \cdots X_n$ such that φ is obtained from replacing the propositional variables by L-formulas $\bar{\psi} = \psi_1 \cdots \psi_n$. More precisely, this means $\varphi = \alpha[\bar{X}/\bar{\psi}]$ defined by recursion (writing some extra parentheses):

$$X_i[\bar{X}/\bar{\psi}] := \psi_i, \;(\neg\alpha_0)[\bar{X}/\bar{\psi}] := \neg(\alpha_0[\bar{X}/\bar{\psi}]), \;(\alpha_0 \wedge \alpha_1)[\bar{X}/\bar{\psi}] := (\alpha_0[\bar{X}/\bar{\psi}] \wedge \alpha_1[\bar{X}/\bar{\psi}]).$$

Example 2.4.8. E.g. $(\exists y \forall x Exy \to \forall x \exists y Exy)$ or $(x_0 \dot{=} x_0 \vee \neg x_1 \dot{=} x_1)$ are valid but not tautologies. Tautologies are e.g. $(x_0 \dot{=} x_0 \vee \neg x_0 \dot{=} x_0)$ or $(\forall x_0 Ex_0x_1 \to ((\exists x_7 Ex_7x_7 \wedge \neg Ex_1x_2) \to \forall x_0 Ex_0x_1))$ – take $\alpha := (X_1 \to (X_2 \to X_1))$.

Example 2.4.9. The *equality axioms* Eq_L *(for language L)* are valid and not tautological; these are for $r \in \mathbb{N}$ and r-ary function symbols $f \in L$ and r-ary relation symbols $R \in L$:

$$\begin{aligned}
&\forall x\; x \dot{=} x, \;\forall xy(x \dot{=} y \to y \dot{=} x), \;\forall xyz((x \dot{=} y \wedge y \dot{=} z) \to x \dot{=} z),\\
&\forall x_1 \cdots x_r y_1 \cdots y_r \big((\textstyle\bigwedge_{i=1}^r x_i \dot{=} y_i) \to f x_1 \cdots x_r \dot{=} f y_1 \cdots y_r\big),\\
&\forall x_1 \cdots x_r y_1 \cdots y_r \big((\textstyle\bigwedge_{i=1}^r x_i \dot{=} y_i) \to (R x_1 \cdots x_r \to R y_1 \cdots y_r)\big).
\end{aligned}$$

Proposition 2.4.10. *Tautologies are valid.*

Proof. In the notation above, show for every L-structure $\mathfrak{A}$ and every assignment β in $\mathfrak{A}$:

$$\mathfrak{A} \models (\alpha[\bar{X}/\bar{\psi}])[\beta] \iff \beta_{prop} \models \alpha$$

where β_{prop} is a propositional assignment that maps X_i to 1 if $\mathfrak{A} \models \psi_i[\beta]$, and to 0 otherwise. This is a trivial induction on α. □

Definition 2.4.11. An L-*literal* is an atomic L-formula or a negation thereof. A *DNF* is an L-formula of the form $\bigvee_{i=1}^n \bigwedge_{j=1}^{n_i} \lambda_{ij}$, a *CNF* is an L-formula of the form $\bigwedge_{i=1}^n \bigvee_{j=1}^{n_i} \lambda_{ij}$ where $n, n_i \in \mathbb{N}$ and the λ_i, λ_{ij} are L-literals.

Corollary 2.4.12. *Every quantifier free L-formula is equivalent to both a DNF and a CNF.*

Proof. Let φ be quantifier free. It can be seen as a propositional formula where the propositional variables are atomic L-formulas. More precisely, $\varphi = \alpha[\bar{X}/\bar{\psi}]$ where α is a propositional formula and $\bar{\psi}$ is a tuple of atomic L-formulas. If $(\alpha \leftrightarrow \alpha')$ is a valid propositional formula, then $(\alpha \leftrightarrow \alpha')[\bar{X}/\bar{\psi}] = (\alpha[\bar{X}/\bar{\psi}] \leftrightarrow \alpha'[\bar{X}/\bar{\psi}])$ is a valid L-formula by Proposition 2.4.10, so $\alpha[\bar{X}/\bar{\psi}], \alpha'[\bar{X}/\bar{\psi}]$ are equivalent L-formulas. By Corollary 2.4.12 α' can be chosen as a propositional DNF or CNF. $\square$

Exercise 2.4.13. Replacing within a subformula occurrence ψ by an equivalent formula χ leads to an equivalent formula. Prove this as follows. Let $P \notin L$ be a 0-ary relation symbol, φ an $L \cup \{P\}$-formula and ψ, χ L-formulas. Let $\varphi[P/\psi]$ be obtained from φ by replacing every occurrence of P by ψ. Define this by recursion on φ. Then prove

$$\psi \equiv \chi \implies \varphi[P/\psi] \equiv \varphi[P/\chi].$$

2.5 Skolemization

Let L be a language and $\mathfrak{A}$ an L-structure.

2.5.1 Substitutions

Informally: suppose you have a formula $\varphi(x) = \ldots x \ldots x \ldots$, and a term s. Substituting s for x gives $\varphi[x/t] := \ldots t \ldots t \ldots$. The truth of $\varphi[x/t]$ should be equivalent to the element denoted by s satisfying $\varphi(x)$.

Some care is needed here: let β be an assignment in $\mathfrak{A}$, $\varphi(x) := \forall y\ x \dot{=} y$ and $t := y$. Then $\varphi[x/t] = \forall y\ y \dot{=} y$ is a valid sentence, so true in $\mathfrak{A}$. The "element denoted by t" is $t^{\mathfrak{A}}[\beta] = \beta(y) =: a$. That it "satisfies $\varphi(x)$" means $\mathfrak{A} \models \varphi[a]$ – if and only if $A = \{a\}$.

Intuitively, the problem is that a variable in t gets quantified within φ after substitution. We now proceed formally and carefully verify a corrected version of the initial intuition.

Definition 2.5.1. A *substitution* is a function σ with finite domain $dom(\sigma) \subseteq Var$ and values in the set of L-terms. For an L-term t define t^σ by recursion:

– if $t = y$ is a variable, then $y^\sigma := \sigma(y)$ if $y \in dom(\sigma)$, and $y^\sigma = y$ otherwise.

– if $t = ft_1 \cdots t_r$ for some $r \in \mathbb{N}$, $f \in L$ and $t_1, \ldots, t_r$, then $t^\sigma := ft_1^\sigma \cdots t^\sigma$.

For an L-formula φ define φ^σ by recursion:

– if $\varphi = t_0 \dot{=} t_1$ for some t_0, t_1, then $\varphi^\sigma :=\ t_0^\sigma \dot{=} t_1^\sigma$;

– if $\varphi = Rt_1 \cdots t_r$ for r-ary $R \in L$ and L-terms t_i, then $\varphi^\sigma = Rt_1^\sigma \cdots t_r^\sigma$;

– if $\varphi = \neg\psi$ for some ψ, then $\varphi^\sigma := \neg\psi^\sigma$;

– if $\varphi = (\psi \wedge \chi)$ for some ψ, χ, then $\varphi^\sigma := (\psi^\sigma \wedge \chi^\sigma)$;

– if $\varphi = \exists y \psi$ for some ψ and $y \in \mathit{Var}$, then $\varphi^\sigma := \exists y \psi^{\sigma'}$ for $\sigma' := \sigma \restriction (\mathit{dom}(\sigma) \setminus \{y\})$.

If $x_1, \ldots, x_k$ lists $\mathit{dom}(\sigma)$ and $t_i := \sigma(x_i)$ we denote σ by $[x_1/t_1, \ldots, x_k/t_k]$ and φ^σ, t^σ by $\varphi[x_1/t_1, \ldots, x_k/t_k]$, $t[x_1/t_1, \ldots, x_k/t_k]$. Further, if $\varphi = \varphi(x_1, \ldots, x_k, \bar{x})$ we write

$$\varphi(t_1, \ldots, t_k, \bar{x}) := \varphi^\sigma.$$

Note substitution is "simultaneous": $Exy[x/y, y/z] = Eyz$ and $(Exy[x/y])[y/z] = Ezz$.

Definition 2.5.2. Let x be a variable, t an L-term and φ an L-formula. Recursively define *x is free for t in φ*:

– x is free for t in every atomic L-formula;

– x is free for t in $\neg\psi$ if and only if x is free for t in ψ;

– x is free for t in $(\psi \wedge \chi)$ if and only if x is free for t in both ψ and χ;

– x is free for t in $\exists y \psi$ if and only if either $x = y$, or, $y \neq x$ and x is free for t in ψ and y does not occur in t.

Observe that if t is closed, then x is free for t in φ for all x, φ.

Lemma 2.5.3 (Substitution)**.** *Let x be a variable, t an L-term, β an assignment in $\mathfrak{A}$ and $a := t^{\mathfrak{A}}[\beta] \in A$.*

1. *For every L-term s: $(s[x/t])^{\mathfrak{A}}[\beta] = s^{\mathfrak{A}}\big[\beta[x/a]\big]$.*
2. *For every L-formula φ with x free for t in φ:*

$$\mathfrak{A} \models (\varphi[x/t])[\beta] \iff \mathfrak{A} \models \varphi\big[\beta[x/a]\big].$$

Proof. 1 is proved by a straightforward induction on t. We prove 2 by induction on φ. If φ is atomic, the claim follows from 1: e.g., if $\varphi = t_0 \dot{=} t_1$ then the l.h.s. means $(t_0[x/t])^{\mathfrak{A}}[\beta] = (t_t[x/t])^{\mathfrak{A}}[\beta]$,and the r.h.s. means $t_0^{\mathfrak{A}}[\beta[x/a]] = t_1^{\mathfrak{A}}[\beta[x/a]]$; these are equivalent by (1).

The cases that φ is a negation or a conjunction are easy. Suppose $\varphi = \exists y \psi$. In case $y = x$, $\varphi[x/t] = \varphi$ and we have to show $\mathfrak{A} \models \varphi[\beta] \iff \mathfrak{A} \models \varphi[\beta[x/a]]$. This follows from the first coincidence lemma because $x \notin \mathit{free}(\varphi)$.

So assume $y \neq x$. Then x is free for t in ψ and y does not occur in t. Then $\varphi[x/t] = \exists y(\psi[x/t])$, so $\mathfrak{A} \models (\varphi[x/t])[\beta]$ if and only if $\mathfrak{A} \models (\psi[x/t])[\gamma_b]$ for some $b \in A$ where $\gamma_b := \beta[y/b]$. By induction this is equivalent to

$$\mathfrak{A} \models \psi\big[\gamma_b[x/t^{\mathfrak{A}}[\gamma_b]]\big].$$

But by the coincidence lemma for terms, $t^{\mathfrak{A}}[\gamma_b] = t^{\mathfrak{A}}[\beta[y/b]] = t^{\mathfrak{A}}[\beta] = a$, so $\gamma_b[x/t^{\mathfrak{A}}[\gamma_b]] = (\beta[y/b])[x/a] = (\beta[x/a])[y/b]$ as $y \neq x$. Thus, the above is equivalent to

$$\mathfrak{A} \models \psi\big[(\beta[x/a])[y/b]\big].$$

That this holds for some $b \in A$ means $\mathfrak{A} \models \exists y \psi\big[\beta[x/a]\big]$. □

Exercise 2.5.4. Let φ be an L-formula, $x \in \mathit{Var}$ and t an L-term.

1. Rename bounded variables to get φ' equivalent to φ such that x is free for t in φ'.
2. If x is free for t in $\varphi(x, \bar{x})$, then $(\varphi(t, \bar{x}) \to \exists x \varphi(x, \bar{x}))$ is valid.

Exercise 2.5.5. Let T be an L-theory, $\varphi(\bar{x})$ an L-formula, and $\bar{c}$ a tuple of constants outside L. Show that $T \vDash \forall \bar{x} \varphi(\bar{x}), T \vDash \varphi(\bar{x})$ and $T \vDash \varphi(\bar{c})$ are equivalent.

2.5.2 Skolemization

Definition 2.5.6. Let $\varphi(\bar{x})$ be a prenex L-formula, i.e., it is of the form

$$\forall \bar{x}_1 \exists y_1 \forall \bar{x}_2 \exists y_2 \cdots \forall \bar{x}_k \exists y_k \forall \bar{x}_{k+1} \psi(\bar{x}, \bar{x}_1, y_1, \bar{x}_2, y_2, \ldots, \bar{x}_k, y_k, \bar{x}_{k+1})$$

where $k \in \mathbb{N}$, ψ is quantifier free, the variables displayed are pairwise distinct, and some tuples $\bar{x}_i$ may be empty. Let r_i be the length of $\bar{x}_i$ ($r_i = 0$ is allowed).

Let $f_1, \ldots, f_k \notin L$ be function symbols; the arity of f_i is $r_1 + \cdots + r_i$. Let σ be the substitution $y_i \mapsto f_i \bar{x}_1 \cdots \bar{x}_i$. A *Skolemization of* φ is the $L \cup \{f_1, \ldots, f_k\}$-formula:

$$\varphi^{Sk} \;:=\; \forall \bar{x}_1 \cdots \bar{x}_{k+1} \psi^\sigma = \forall \bar{x}_1 \cdots \bar{x}_{k+1}\ \psi\big(\bar{x}, \bar{x}_1,\ f_1\bar{x}_1,\ \bar{x}_2,\ f_2\bar{x}_1\bar{x}_2,\ \ldots, \bar{x}_k,\ f_k\bar{x}_1\cdots\bar{x}_k,\ \bar{x}_{k+1}\big).$$

The notation φ^{Sk} suppresses the choice of new function symbols.

Example 2.5.7. Continuing Example 2.4.6 for $((\exists y Exy \wedge \neg \exists x Exy) \wedge \neg \forall y Eyx)$ gives a Skolemization $\forall x'((Exc \wedge \neg Ex'y) \wedge \neg Efx'x)$ for a constant c and unary f.

Another equivalent prenex formula is computed

$$\begin{aligned}
&((\exists y Exy \wedge \neg \exists x Exy) \wedge \neg \forall y Eyx),\\
&\forall x'((\exists y Exy \wedge \neg Ex'y) \wedge \neg \forall y Eyx),\\
&\forall x' \exists y'((Exy' \wedge \neg Ex'y) \wedge \neg \forall y Eyx),\\
&\forall x' \exists y' \exists y''((Exy' \wedge \neg Ex'y) \wedge Ey''x),
\end{aligned}$$

and gives a Skolemization $\forall x'((Exfx' \wedge \neg Ex'y) \wedge Egx'x)$ for unary f, g.

Example 2.5.8. We write the group axioms with a ternary relation P for the graph of the group operation and $\rightsquigarrow$ skolemize with binary g, unary i and constant e:

Existence of values (uniqueness omitted): $\forall xy \exists z Pxyz \rightsquigarrow Pxygxy$.

Associativity: $\begin{array}{l} \forall xyzuvw((Pxyu \wedge Pyzv \wedge Pxvw) \to Puzw) \\ \forall xyzuvw((Pxyu \wedge Pyzv \wedge Puzw) \to Pxvw) \end{array}$.

Left neutral and left inverse: $\begin{array}{l} \exists x(\forall u Pxuu \wedge \forall y \exists z Pzyx) \\ \equiv \exists x \forall y \exists z \forall u (Pxuu \wedge Pzyx) \quad \rightsquigarrow \quad \forall yu(Peuu \wedge Piyye) \end{array}$.

We see that Skolemization introduces symbols for the group operation, inverse and neutral element – for suitably chosen prenex forms.

Lemma 2.5.9. *Let φ be a prenex L-sentence.*

1. *$(\varphi^{Sk} \to \varphi)$ is valid.*
2. *Every model of φ has an expansion to a model of φ^{Sk}.*

In particular, φ is satisfiable if and only if so is φ^{Sk}.

Proof. We claim that for every L-formula $\varphi(\bar{z}, \bar{x}, y)$ such that y is free for $f\bar{x}$ in φ:

(a) $(\forall\bar{x}(\varphi[y/f\bar{x}]) \to \forall\bar{x}\exists y\varphi)$ is valid;

(b) if $\mathfrak{A} \vDash \forall\bar{x}\exists y\varphi[\beta]$ for some L-structure $\mathfrak{A}$ and assignment β in $\mathfrak{A}$, then there is an $L \cup \{f\}$-expansion $\mathfrak{A}'$ of $\mathfrak{A}$ such that $\mathfrak{A}' \vDash \forall\bar{x}(\varphi[y/f\bar{x}])[\beta]$.

(a): $((\varphi[y/f\bar{x}]) \to \exists y\varphi)$ is valid by Exercise 2.5.4. (b): suppose $\mathfrak{A} \vDash \forall\bar{x}\exists y\varphi[\beta]$ and let $\bar{b} := \beta(\bar{z})$. Let $\mathfrak{A}'$ be the $L \cup \{f\}$-expansion of $\mathfrak{A}$ that interprets f by a function that maps $\bar{a}$ to an a with $\mathfrak{A} \vDash \varphi(\bar{z}, \bar{x}, y)[\bar{b}, \bar{a}, a]$. Note $\mathfrak{A} \vDash \varphi[\beta[\bar{x}/\bar{a}, y/a]]$ for $a := (f\bar{x})^{\mathfrak{A}'}[\beta[\bar{x}/\bar{a}]]$. By the substitution lemma, $\mathfrak{A}' \vDash (\varphi[y/f\bar{x}])[\beta[\bar{x}/\bar{a}]]$.

We prove 1 for formulas φ of the form displayed above by induction on k. If $k = 0$, then $\varphi^{Sk} = \varphi$. For $k > 0$ let φ_- be φ with $\forall\bar{x}_1\exists y_1$ deleted. Note

$$\varphi^{Sk} = \big(\forall\bar{x}_1(\varphi_-[y_1/f\bar{x}_1])\big)^{Sk}.$$

Hence, by induction, $(\varphi^{Sk} \to \forall\bar{x}_1(\varphi_-[y_1/f\bar{x}_1]))$ is valid. Since also $(\forall\bar{x}_1(\varphi_-[y_1/f\bar{x}_1]) \to \varphi)$ is valid by (a), the claim follows.

For 2, we show for every $\mathfrak{A}, \beta$ there is an expansion $\mathfrak{A}'$ of $\mathfrak{A}$ such that: if $\mathfrak{A} \vDash \varphi[\beta]$, then $\mathfrak{A}' \vDash \varphi^{Sk}[\beta]$. For $k = 0$ there is nothing to show. For $k > 0$ assume $\mathfrak{A} \vDash \forall\bar{x}_1\exists y_1\varphi_-[\beta]$. By (b) choose an $L \cup \{f_1\}$-expansion $\mathfrak{A}'$ of $\mathfrak{A}$ such that $\mathfrak{A}' \vDash \forall\bar{x}_1(\varphi_-[y_1/f_1\bar{x}_1])[\beta]$. By induction there is an expansion $\mathfrak{A}''$ of $\mathfrak{A}'$ such that

$$\mathfrak{A}'' \vDash \big(\forall\bar{x}_1(\varphi_-[y_1/f_1\bar{x}_1])\big)^{Sk}[\beta].$$

This formula equals φ^{Sk}. □

2.5.3 Löwenheim-Skolem theorem

A set S is *countable* if there is a bijection from $\mathbb{N}$ onto S, and *at most countable* if it is finite or countable. An L-structure $\mathfrak{A}$ is *(at most) countable* or *(in)finite* if so is its universe A.

Theorem 2.5.10 (Löwenheim-Skolem - downwards)**.** *Assume L is at most countable and T is an L-theory. If T is satisfiable, then T has an at most countable model.*

Proof. There are countably many L-formulas, so we can write $T = \{\varphi_1, \varphi_2 \ldots\}$ (possibly with repetitions). Set $T^{Sk} := \{\varphi_1^{Sk}, \varphi_2^{Sk}, \ldots\}$ where each Skolemization φ_i^{Sk} uses its own finite set of new functions symbols. Let $\mathfrak{A} \vDash T$ and $\mathfrak{A}'$ be the expansion to the new functions symbols simultaneously for all φ_i^{Sk} – so that $\mathfrak{A}' \vDash T^{Sk}$ (Lemma 2.5.9 (2)). Note the language L' of $\mathfrak{A}'$ is at most countable, in particular, there are at most countably many L'-terms $t(x)$. Let $a \in A$ be arbitrary and consider $\langle\{a\}\rangle^{\mathfrak{A}'}$. As the universe consists of the values $t^{\mathfrak{A}'}[a]$, it is at most countable. As T^{Sk} is universal, $\langle\{a\}\rangle^{\mathfrak{A}'} \vDash T^{Sk}$ (Exercise 2.3.21). Then $\langle\{a\}\rangle^{\mathfrak{A}'} \vDash T$ by Lemma 2.5.9 (1), so $\langle\{a\}\rangle^{\mathfrak{A}'}\restriction L \vDash T$ by the second coincidence lemma. □

Example 2.5.11. Consider the language $L = \{+, \cdot, -, 0, 1\}$ of fields and let $\mathfrak{C}$ be the field of complex numbers. Apply the above to the theory T of all sentences true in $\mathfrak{C}$. Then there is a countable field $\mathfrak{C}'$ that satisfies the same L-sentences as $\mathfrak{C}$. In fact, methods of model theory show that the algebraic closure of the field of rational numbers is such a field.

2.6 Formal reasoning I: Gentzen's Logischer Kalkül

Let L be a language and C be a countable set of constants with $L \cap C = \varnothing$.

Definition 2.6.1. A *sequent* is a pair (Γ, Δ) of finite sets of L-sentences, written $\Gamma \Rightarrow \Delta$. It is *valid* if $\Gamma \vDash \bigvee \Delta$. It is *LK-provable* if there is an LK-proof ending with it.

An *LK-proof* is a finite sequence of sequents such that every sequent in it is a conclusion of an LK-rule with premisses appearing earlier in the sequence.

There are the following *LK-rules*, written $\frac{\text{Premisses}}{\text{Conclusion}}$:

- Rules of propositional LK (for sequents as above);
- $\exists$-left $\dfrac{\Gamma, \varphi(c) \Rightarrow \Delta}{\Gamma, \exists x \varphi(x) \Rightarrow \Delta}$ where $c \in C$ does not appear in the conclusion;
- $\exists$-right $\dfrac{\Gamma \Rightarrow \Delta, \varphi(t)}{\Gamma \Rightarrow \Delta, \exists x \varphi(x)}$ where t is a closed $L \cup C$-term;
- Cut: $\dfrac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$;
- Equality rules:

$$\frac{}{\Gamma \Rightarrow \Delta, t \doteq t}, \quad \frac{}{\Gamma, t \doteq t' \Rightarrow \Delta, t' \doteq t}, \quad \frac{}{\Gamma, t \doteq t', t \doteq t'' \Rightarrow \Delta, t \doteq t''},$$
$$\frac{}{\Gamma, t_1 \doteq t_1', \ldots, t_r \doteq t_r' \Rightarrow \Delta, f t_0 \cdots t_r \doteq f t_1' \cdots t_r'}, \quad \frac{}{\Gamma, t_1 \doteq t_1', \ldots, t_r \doteq t_r', R t_1 \cdots t_r \Rightarrow \Delta, R t_1' \cdots t_r'}$$

  where $r \in \mathbb{N}$, $f, R \in L$ are r-ary function and relation symbols, and t, t', t'', t_i, t_i' are closed $L \cup C$-terms.

Intuition: as in the propositional case LK-proofs are best read and constructed bottom-up, reading a sequent $\Gamma \Rightarrow \Delta$ as "all formulas in Γ are true and all formulas in Δ are false". Then the new rules $\exists$-left/right become clear: $\exists$-right assumes $\exists x \varphi(x)$ is false and infers $\varphi(t)$ is false for arbitrary t; $\exists$-left assumes $\exists x \varphi(x)$ is true and gives a new name to an example, that is, infers $\varphi(c)$ is true for a new constant c.

Remark 2.6.2. Define LK$^+$-proofs like LK-proofs but adding the left/right-rules for $\vee, \to$ and $\leftrightarrow$ from Remark 1.4.3 and additionally:

$\forall$-left $\dfrac{\Gamma, \varphi(t) \Rightarrow \Delta}{\Gamma, \forall x \varphi(x) \Rightarrow \Delta}$ where t is a closed $L \cup C$-term;

$$\forall\text{-right}\quad \frac{\Gamma \Rightarrow \Delta, \varphi(c)}{\Gamma \Rightarrow \Delta, \forall x \varphi(x)} \quad \text{where } c \in C \text{ does not appear in the conclusion.}$$

Then LK⁺-provable sequents are LK-provable.

Proof. Replace ∀-left/right applications by

$$\dfrac{\dfrac{\dfrac{\Gamma, \varphi(t) \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi(t)}}{\Gamma \Rightarrow \Delta, \exists x \neg\varphi(x)}}{\Gamma, \neg\exists x \neg\varphi(x) \Rightarrow \Delta} \qquad \dfrac{\dfrac{\dfrac{\Gamma \Rightarrow \Delta, \varphi(c)}{\Gamma, \neg\varphi(c) \Rightarrow \Delta}}{\Gamma, \exists x \neg\varphi(x) \Rightarrow \Delta}}{\Gamma \Rightarrow \Delta, \neg\exists x \neg\varphi(x)}$$

For the ∃-left application on the right, note c does not appear in $\Gamma, \exists x \neg\varphi(x) \Rightarrow \Delta$. □

Example 2.6.3. The following is an LK⁺-proof for distinct $c, d \in C$ and a formula $\varphi(x, y)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi(d,c) \Rightarrow \varphi(d,c)}{\varphi(d,c) \Rightarrow \exists y \varphi(d,y)}}{\forall x \varphi(x,c) \Rightarrow \exists y \varphi(d,y)}}{\forall x \varphi(x,c) \Rightarrow \forall x \exists y \varphi(x,y)}}{\exists y \forall x \varphi(x,y) \Rightarrow \forall x \exists y \varphi(x,y)}$$

Exercise 2.6.4. Show that the equality axioms from Example 2.4.9 are LK⁺-provable.

2.6.1 Equality free case

Definition 2.6.5. Let *LK⁻-proofs* be defined as LK-proofs but omitting Cut and Equality rules. A sequent $\Gamma \Rightarrow \Delta$ is $\doteq$-free if all formulas in $\Gamma \cup \Delta$ are, and a formula is *$\doteq$-free* if $\doteq$ does not occur in it.

Theorem 2.6.6 (LK⁻ completeness). *A $\doteq$-free sequent is valid if and only if it is LK⁻-provable.*

Proof. ⇐ (Soundness): it suffices to show for all rules that if the premisses are valid, then so is the conclusion. We verify this for the new rules ∃-left/right. Write $\neg\Delta := \{\neg\psi \mid \psi \in \Delta\}$

Assume $\Gamma \Rightarrow \Delta, \exists x \varphi(x)$ is not valid. Then there is a $L \cup C$-structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Gamma \cup \neg\Delta$ and $\mathfrak{A} \models \neg\exists x \varphi(x)$, so $\mathfrak{A} \models \neg\varphi[a]$ for all $a \in A$. In particular, given a closed term t, we have $\mathfrak{A} \models \neg\varphi[t^{\mathfrak{A}}]$. Since t is closed, x is free for t in φ. By the substitution lemma, $\mathfrak{A} \models \neg\varphi(t)$. Hence $\mathfrak{A}$ witnesses that $\Gamma \Rightarrow \Delta, \varphi(t)$ is not valid.

Assume $\Gamma, \exists x \varphi(x) \Rightarrow \Delta$ is not valid. Choose a finite $C_0 \subseteq C$ such that all formulas in this sequent are $L \cup C_0$-sentences. Then there is a $L \cup C_0$-structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Gamma \cup \neg\Delta$ and $\mathfrak{A} \models \exists x \varphi(x)$. Choose $a \in A$ such that $\mathfrak{A} \models \varphi[a]$ and $c \in C \setminus \{c\}$and let $\mathfrak{A}'$ be the $L \cup C \cup \{c\}$ expansion of $\mathfrak{A}$ that interprets c by a. Then $\mathfrak{A}' \models \varphi[c^{\mathfrak{A}}]$, so $\mathfrak{A}' \models \varphi(c)$ by the substitution lemma. Since c does not occur in $\Gamma \cup \neg\Delta$, the second coincidence lemma implies $\mathfrak{A}' \models \Gamma \cup \neg\Delta$. Hence $\mathfrak{A}'$ witnesses that $\Gamma, \varphi(c) \Rightarrow \Delta$ is not valid.

$\Rightarrow$ (Completeness): suppose $\Gamma \Rightarrow \Delta$ is $\doteq$-free and not LK$^-$-provable. Choose a finite $L_0 \subseteq L$ such that only $L_0 \cup C$-sentences appear. By the second coincidence lemma, it suffices to construct an $L_0 \cup C$-structure $\mathfrak{A}$ satisfying $\Gamma \cup \neg\Delta$.

Consider triples (b, φ, t) for $b \in \{0,1\}, \varphi$ an $L_0 \cup C$-sentence, and t a closed $L_0 \cup C$-term. Let $(b_0, \varphi_0), (b_1, \varphi_1, t_1), \ldots$ be an enumeration such that every triple appears infinitely often.

We inductively construct LK$^-$-unprovable sequents $\Gamma_0 \Rightarrow \Delta_0, \Gamma_1 \Rightarrow \Delta_1, \ldots$ starting with $\Gamma_0 := \Gamma, \Delta_0 := \Delta$. Assume $\Gamma_i \Rightarrow \Delta_i$ is constructed and LK$^-$-unprovable.

To define $\Gamma_{i+1} \Rightarrow \Delta_{i+1}$ consider (b_i, φ_i, t_i) and do the following:

1. if $\varphi_i = \neg\psi \in \Gamma_i$ and $b_i = 0$, then $\Gamma_i \Rightarrow \Delta_i, \psi$ is LK$^-$-unprovable; set $\Gamma_{i+1} := \Gamma_i$ and $\Delta_{i+1} := \Delta_i \cup \{\psi\}$;
2. if $\varphi_i = \neg\psi \in \Delta_i$ and $b_i = 1$, then set $\Gamma_{i+1} := \Gamma_i \cup \{\psi\}$ and $\Delta_{i+1} := \Delta_i$;
3. if $\varphi_i = (\psi \wedge \chi) \in \Gamma_i$ and $b_i = 0$, then $\Gamma_i, \psi, \chi \Rightarrow \Delta_i$ is not LK$^-$-provable; set $\Gamma_{i+1} := \Gamma_i \cup \{\psi, \chi\}$ and $\Delta_{i+1} := \Delta_i$.
4. if $\varphi_i = (\psi \wedge \chi) \in \Delta_i$ and $b_i = 1$, then $\Gamma_i \Rightarrow \Delta_i, \psi$ or $\Gamma_i \Rightarrow \Delta_i, \chi$ is not LK$^-$-provable; choose $\Gamma_{i+1} \Rightarrow \Delta_{i+1}$ accordingly.
5. if $\varphi_i = \exists x \varphi(x) \in \Gamma_i$ and $b_i = 0$, then choose $c \in C$ not occurring in $\Gamma_i \Rightarrow \Delta_i$; then $\Gamma_i, \varphi(c) \Rightarrow \Delta_i$ is not LK$^-$-provable; set $\Gamma_{i+1} := \Gamma_i \cup \{\varphi(c)\}$ and $\Delta_{i+1} := \Delta_i$.
6. if $\varphi_i = \exists x \varphi(x) \in \Delta_i$ and $b_i = 1$, then $\Gamma_i \Rightarrow \Delta_i, \varphi(t_i)$ is not LK$^-$-provable; set $\Gamma_{i+1} := \Gamma_i$ and $\Delta_{i+1} := \Delta_i \cup \{\varphi(t_i)\}$.
7. if none of the above, set $\Gamma_{i+1} := \Gamma_i$ and $\Delta_{i+1} := \Delta_i$.

By construction we we have $\Gamma_0 \subseteq \Gamma_1 \subseteq \cdots$ and $\Delta_0 \subseteq \Delta_1 \subseteq \cdots$. Set

$$\Gamma^* := \textstyle\bigcup_{i \in \mathbb{N}} \Gamma_i \quad \text{and} \quad \Delta^* := \bigcup_{i \in \mathbb{N}} \Delta_i.$$

Claim: (Γ^*, Δ^*) has the *Henkin properties*, namely for all $L_0 \cup C$-sentences ψ, χ and $L_0 \cup C$-formulas $\varphi(x)$:

(H1) $\Gamma^* \cap \Delta^* = \varnothing$;

(H2) if $\neg\psi \in \Gamma^*$, then $\psi \in \Delta^*$;

(H3) if $\neg\psi \in \Delta^*$, then $\psi \in \Gamma^*$;

(H4) if $(\psi \wedge \chi) \in \Gamma^*$, then $\psi, \chi \in \Gamma^*$;

(H5) if $(\psi \wedge \chi) \in \Delta^*$, then $\psi \in \Delta^*$ or $\chi \in \Delta^*$;

(H6) if $\exists x \varphi(x) \in \Gamma^*$, then $\varphi(c) \in \Gamma^*$ for some $c \in C$;

(H7) if $\exists x \varphi(x) \in \Delta^*$, then $\varphi(t) \in \Delta^*$ for every closed $L_0 \cup C$-term t.

Proof of the claim: (H1) follows from $\Gamma_i \cap \Delta_i = \varnothing$ for all $i \in \mathbb{N}$ (by LK$^-$-unprovability). To see e.g. (H7), assume $\exists x \varphi(x) \in \Delta^*$, say $\in \Delta_{i_0}$, and let t be given. Choose $i \geqslant i_0$ such that $(b_i, \varphi_i, t_i) = (1, \exists x \varphi(x), t)$. By construction, $\Delta_{i+1} = \Delta_i \cup \{\varphi(t)\}$, so $\varphi(t) \in \Delta^*$.

The proofs of (H2)-(H6) are analogous. $\dashv$

The Henkin properties allow to "read-off" a model $\mathfrak{A}$ satisfying Γ^* and falsifying Δ^*. The universe A is the set of closed $L_0 \cup C$-terms. For $r \in \mathbb{N}$ and $f \in L_0$ an r-ary function symbol, the interpretation $f^{\mathfrak{A}}$ maps $(t_1, \ldots, t_r) \in A^r$ to $ft_1 \cdots t_r \in R$. For $r \in \mathbb{N}$ and $R \in L_0$ an r-ary relation symbol the interpretation is

$$R^{\mathfrak{A}} := \big\{(t_1, \ldots, t_r) \in A^r \mid Rt_1 \cdots t_r \in \Gamma^*\big\}.$$

Observe that $\mathfrak{A}$ interprets every term "by itself", namely $t^{\mathfrak{A}} = t$ for every closed $L_0 \cup C$-term t. Use induction: by definition $c^{\mathfrak{A}} = c$ for a constant $c \in L_0 \cup C$, and $(ft_1 \cdots t_r)^{\mathfrak{A}} = f^{\mathfrak{A}}(t_1^{\mathfrak{A}}, \ldots, t_r^{\mathfrak{A}}) = f^{\mathfrak{A}}(t_1, \ldots, t_r) = ft_1 \cdots t_r$.

We are left to show for every $\doteq$-free $L_0 \cup C$-sentence φ:

$$\begin{aligned} \varphi \in \Gamma^* &\implies \mathfrak{A} \models \varphi, \\ \varphi \in \Delta^* &\implies \mathfrak{A} \not\models \varphi. \end{aligned}$$

To prove this we use induction on the number n of occurrences of $\neg \wedge \exists$ in φ.

If $n = 0$, φ is atomic. Since φ is $\doteq$-free it is of the form $Rt_1 \cdots t_r$. Then $\varphi \in \Gamma^*$ implies $(t_1, \ldots, t_r) \in R^{\mathfrak{A}}$; but $t_i = t_i^{\mathfrak{A}}$, so $\mathfrak{A} \models Rt_1 \cdots t_r$. Further, $\varphi \in \Delta^*$ implies $\varphi \notin \Gamma^*$ by (H1), so $(t_1, \ldots, t_r) \in R^{\mathfrak{A}}$; by $t_i = t_i^{\mathfrak{A}}$, we get $\mathfrak{A} \not\models Rt_1 \cdots t_r$.

For $n > 0$, we distinguish cases:

- $\varphi = \neg\psi$.
  If $\varphi \in \Gamma^*$, then $\psi \in \Delta^*$ by (H2); by induction, $\mathfrak{A} \not\models \psi$, so $\mathfrak{A} \models \varphi$.
  If $\varphi \in \Delta^*$, then $\psi \in \Gamma^*$ by (H3); by induction, $\mathfrak{A} \models \psi$, so $\mathfrak{A} \not\models \varphi$.
- $\varphi = (\psi \wedge \chi)$.
  If $\varphi \in \Gamma^*$, then $\psi, \chi \in \Gamma^*$ by (H4); by induction, $\mathfrak{A} \models \psi$ and $\mathfrak{A} \models \chi$, so $\mathfrak{A} \models \varphi$.
  If $\varphi \in \Delta^*$, then $\psi \in \Delta^*$ or $\chi \in \Delta^*$ by (H5); by induction, $\mathfrak{A} \not\models \psi$ or $\mathfrak{A} \not\models \chi$, so $\mathfrak{A} \not\models \varphi$.
- $\varphi = \exists x \psi(x)$.
  If $\varphi \in \Gamma^*$, then $\psi(c) \in \Gamma^*$ for some $c \in C$ by (H6); by induction, $\mathfrak{A} \models \psi(c)$, so by the substitution lemma $\mathfrak{A} \models \psi[c^{\mathfrak{A}}]$, so $\mathfrak{A} \models \varphi$.
  If $\varphi \in \Delta^*$, then $\psi(t) \in \Delta^*$ for all closed $L_0 \cup C$-terms t by (H7); by induction, $\mathfrak{A} \not\models \psi(t)$, so, by the substitution lemma, $\mathfrak{A} \not\models \psi[t^{\mathfrak{A}}]$ for all closed $L_0 \cup C$-terms t; recalling $t^{\mathfrak{A}} = t$ and the definition of A, this implies $\mathfrak{A} \not\models \varphi[a]$ for all $a \in A$, that is, $\mathfrak{A} \not\models \varphi$. □

2.6.2 General case

Theorem 2.6.7 (LK completeness)**.** *A sequent is valid if and only if it is LK-provable. In particular, if $\varphi(\bar{x})$ is an L-formula and $\bar{c}$ a tuple of constants from C, then $\varphi(\bar{x})$ is valid if and only if $\Rightarrow \varphi(\bar{c})$ is LK-provable.*

The proof is by reduction to the $\doteq$-free case. To this end let $E \notin L$ be a new binary relation symbol intended to be used instead of $\doteq$. Recall the set of equality axioms Eq_L from Example 2.4.9.

Definition 2.6.8. For an L-formula φ let φ^E be obtained replacing atomic subformulas $t \dot{=} t'$ by Ett'. For a set Φ of L-formulas let $\Phi^E := \{\varphi^E \mid \varphi \in \Phi\}$.

Recall, a relation K is an *equivalence relation* on a set A if $K \subseteq A^2$ is reflexive, symmetric and transitive, i.e., for all $a, b, c \in A$, $(a, a) \in K$, and, $(a, b) \in K$ implies $(b, a) \in K$, and, $(a, b), (b, c) \in K$ implies $(a, c) \in K$.

Definition 2.6.9. Let $\mathfrak{A}$ be an L-structure. A *congruence on* $\mathfrak{A}$ is an equivalence relation K on A such that for all $r \in \mathbb{N}$, every r-ary function symbol $f \in L$, every r-ary relation symbol $R \in L$ and all $a_1, \dots a_r, b_1, \dots, b_r \in A$:

(K1) if $(a_1, b_1), \dots, (a_r, b_r) \in K$, then $(f^{\mathfrak{A}}(a_1, \dots, a_r), f^{\mathfrak{A}}(b_1, \dots, b_r)) \in K$;

(K2) $(a_1, \dots, a_r) \in R^{\mathfrak{A}} \iff (b_1, \dots, b_r) \in R^{\mathfrak{A}}$.

Remark 2.6.10. Let $\mathfrak{A}$ be an $L \cup \{E\}$-structure. $E^{\mathfrak{A}}$ is a congruence on $\mathfrak{A} \upharpoonright L$ if and only if $\mathfrak{A} \models Eq_L^E$.

The construction in the following lemma generalizes familiar quotient structures in algebra, like quotient vectorspaces, quotient groups, a ring modulo some ideal,...

Lemma 2.6.11. *Assume $\mathfrak{A}$ is an $L \cup \{E\}$-structure that satisfies Eq_L^E. Then there is an L-structure $\mathfrak{A}/E$ such that for all L-sentences φ:*

$$\mathfrak{A} \models \varphi^E \iff \mathfrak{A}/E \models \varphi.$$

Proof. The universe of $\mathfrak{A}/E$ is $\{a/E \mid a \in A\}$ where $a/E := \{a' \in A \mid (a, a') \in E^{\mathfrak{A}}\}$ is the $E^{\mathfrak{A}}$-equivalence class of $a \in A$. The interpretations of r-ary function and relation symbols $f, R \in L$ are given stipulating for all $a_1, \dots, a_r \in A$:

$$f^{\mathfrak{A}/E}(a_1/E, \dots, a_r/E) := f^{\mathfrak{A}}(a_1, \dots, a_r)/E,$$
$$(a_1/E, \dots, a_r/E) \in R^{\mathfrak{A}/E} \iff (a_1, \dots, a_r) \in R^{\mathfrak{A}}.$$

This is well-defined: by (K1) and (K2) the r.h.s. do not depend on the choice of $a_i \in a_i/E$.

By definition of $\mathfrak{A}/E$ we have $\pi : \mathfrak{A} \upharpoonright L \to_h \mathfrak{A}/E$ for $\pi(a) := a/E$. By Exercise 2.3.6, $\pi(t^{\mathfrak{A} \upharpoonright L}[\beta]) = t^{\mathfrak{A}/E}[\pi \circ \beta]$ for all L-terms t and assignments β in $\mathfrak{A}$.

It now suffices to show for all L-formulas φ and assignments β in $\mathfrak{A}$:

$$\mathfrak{A} \models \varphi^E[\beta] \iff \mathfrak{A}/E \models \varphi[\pi \circ \beta].$$

This is proved by induction on φ. The induction steps being straightforward we only verify the atomic case. If $\varphi = Rt_1 \cdots t_r$, note $\varphi^E = \varphi$, so $\mathfrak{A} \models \varphi^E[\beta]$ means $(t_1^{\mathfrak{A}}[\beta], \dots, t_r^{\mathfrak{A}}[\beta]) \in R^{\mathfrak{A}}$. By definition, this is equivalent to $(t_1^{\mathfrak{A}}[\beta]/E, \dots, t_r^{\mathfrak{A}}[\beta]/E) \in R^{\mathfrak{A}/E}$. Since $t_i^{\mathfrak{A}}[\beta]/E = t_i^{\mathfrak{A}/E}[\pi \circ \beta]$, this means $\mathfrak{A}/E \models \varphi[\pi \circ \beta]$. If $\varphi = t_0 \dot{=} t_1$, note $\varphi^E = Et_0t_1$, and argue

$$\mathfrak{A} \models \varphi^E[\beta] \iff t_0^{\mathfrak{A}}[\beta]/E = t_1^{\mathfrak{A}}[\beta]/E \iff t_0^{\mathfrak{A}/E}[\pi \circ \beta] = t_1^{\mathfrak{A}/E}[\pi \circ \beta] \iff \mathfrak{A}/E \models \varphi[\pi \circ \beta]. \quad \square$$

Corollary 2.6.12. *An L-formula φ is valid if and only if Eq_L^E implies φ^E.*

Proof. $\Rightarrow$: given an $L \cup \{E\}$-structure $\mathfrak{A} \vDash Eq_{L_0}^E$ and an assignment β in $\mathfrak{A}$ we have to show $\mathfrak{A} \vDash \varphi^E[\beta]$, equivalently, $\mathfrak{A}/E \vDash \varphi[\pi \circ \beta]$ (see the previous proof). This holds if φ is valid.

$\Leftarrow$: assume the r.h.s., and let $\mathfrak{A}$ be an L-structure and β an assignment in $\mathfrak{A}$. Let $\mathfrak{A}'$ be the $L \cup \{E\}$-expansion interpreting E by the identity $E^{\mathfrak{A}} := \{(a,a) \mid a \in A\}$. Then $\mathfrak{A}' \vDash Eq_L^E$, so $\mathfrak{A}' \vDash \varphi^E[\beta]$. By the previous proof, $\mathfrak{A}'/E \vDash \varphi^E[\pi \circ \beta]$. But, clearly, $\mathfrak{A}'/E \cong \mathfrak{A}$ via $a/E^{\mathfrak{A}} = \{a\} \mapsto a$, i.e. π^{-1}. Then $\mathfrak{A} \vDash \varphi[\pi^{-1} \circ \pi \circ \beta]$ (Exercise 2.3.19). □

Proof of Theorem 2.6.7. The second statement follows from the first noting $\varphi(\bar{x})$ is valid if and only if $\varphi(\bar{c})$ is valid (Exercise 2.5.5 with $T = \varnothing$).

$\Leftarrow$: (Soundness) clearly, all rules lead from valid premisses to valid conclusions. By a simple induction, all sequents in an LK proof are valid.

$\Rightarrow$: (Completeness) let $\Gamma \Rightarrow \Delta$, hence $(\bigwedge \Gamma \to \bigvee \Delta)$ be valid. This is an L_0-sentence for some finite $L_0 \subseteq L \cup C$. By the corollary, $Eq_{L_0}^E \vDash ((\bigwedge \Gamma)^E \to (\bigvee \Delta)^E)$, so $Eq_{L_0}^E \cup \Gamma^E \Rightarrow \Delta^E$ is valid. By Theorem 2.6.6 it is LK$^-$-provable. Replacing Ett' by $t \dot{=} t'$ throughout the proof gives an LK-proof of $Eq_{L_0} \cup \Gamma \Rightarrow \Delta$. Exercise 2.6.4 and Weakening, gives LK-proofs of $\Gamma \Rightarrow \Delta, \psi$ for all $\psi \in Eq_{L_0}$. Then $|Eq_{L_0}|$ many cuts give a proof of $\Gamma \Rightarrow \Delta$. □

Gentzen's Hauptsatz states Theorem 2.6.7 for LK without the Cut rule. A proof is outside the scope of this course.

Exercise 2.6.13 (Semi-decidability of Hilbert's Entscheidung)**.** There is an algorithm that, given an L-formula φ, halts in a finite number of steps if and only if φ is valid.

2.6.3 Compactness theorem and applications

Let T be an L-theory.

Theorem 2.6.14 (Compactness)**.** *T is satisfiable if and only if every finite subset of T is satisfiable.*

This could be called the "Fundamental Theorem of Model Theory". A proof is outside the scope of this course.

Definition 2.6.15. Let φ be an L-sentence. T *LK-proves* φ, symbolically $T \vdash \varphi$ if there is a finite $T_0 \subseteq T$ such that $T_0 \Rightarrow \varphi$ is LK-provable.

Theorem 2.6.16 (Deductive LK completeness)**.** *Let φ be an L-sentence. Then $T \vDash \varphi$ if and only if $T \vdash \varphi$.*

Proof. The following are equivalent: $T \vDash \varphi$, $T \cup \{\neg\varphi\}$ is unsatisfiable, $T_0 \cup \{\neg\varphi\}$ is unsatisfiable for some finite $T_0 \subseteq T$ (compactness), $T_0 \Rightarrow \neg\varphi$ is valid for some finite $T_0 \subseteq T$, $T_0 \Rightarrow \varphi$ is LK-provable for some finite $T_0 \subseteq T$ (Theorem 2.6.7), $T \vdash \varphi$. □

Theorem 2.6.17 (Löwenheim-Skolem upwards)**.** *Assume T has an infinite model. Let S be any set. Then T has a model whose universe contains S.*

Proof. It suffices to show tat T has a model such that there is an injection of S into its universe. We can assume $S \cap L = \varnothing$ and view S as a set of constants. We claim $T' := T \cup \{\neg s \doteq s' \mid s, s' \in S, s \neq s'\}$ is satisfiable. Then, if $\mathfrak{A} \models T'$, we have $\mathfrak{A} \restriction L \models T$ and $s \mapsto s^{\mathfrak{A}}$ is injective. To prove the claim, by compactness, it suffices to show that every finite $T'_0 \subseteq T'$ is satisfiable. Choose $s_1, \ldots, s_n \in S$ such that $T'_0 \subseteq T \cup \{\neg s_i \doteq s_j \mid i, j \in [n], i \neq j\}$. Choose an infinite $\mathfrak{A} \models T$ and pairwise distinct $a_1, \ldots, a_n \in A$. Then the $L \cup \{s_1, \ldots, s_n\}$-expansion of $\mathfrak{A}$ that interprets s_i by a_i models T'_0. □

Example 2.6.18 (Undefinability of connectedness)**.** Recall a graph $\mathfrak{G} = (G, E^{\mathfrak{G}})$ is *connected* if for all $g, g' \in G$ there is a path from g to g' in $\mathfrak{G}$. There does not exist an $\{E\}$-theory T whose models are precisely the connected graphs.

Proof. Assume T's models are precisely the connected graphs. Let c, d be constants, $n > 0$ and $\varphi_n := \neg \exists x_1 \cdots x_n (c \doteq x_1 \wedge d \doteq x_n \wedge \bigwedge_{i=1}^{n-1} E x_i x_{i+1})$. Then $T \cup \{\varphi_n \mid n > 0\}$ is unsatisfiable: if $\mathfrak{A}$ would be a model, then $\mathfrak{A} \restriction \{E\}$ is a graph with no path from $c^{\mathfrak{A}}$to $d^{\mathfrak{A}}$; but $\mathfrak{A} \restriction \{E\} \models T$, a contradiction. By compactness, there is $m > 0$ such that $T \cup \{\varphi_n \mid n < m\}$ is unsatisfiable. Take a graph that is a path of length m. This is connected, so satisfies T. Its $\{E, c, d\}$-expansion interpreting c, d by the endpoints models all $\varphi_n, n < m$ – contradiction. □

Exercise 2.6.19. Assume L is finite and T is a *decidable* L-theory, i.e., there exists an algorithm that, given an L-sentence φ, decides whether $\varphi \in T$. Describe an algorithm that, given an L-sentence φ, halts in a finite number of steps if and only if $T \vdash \varphi$.

Exercise 2.6.20. Let T be an L-theory. Assume for every $n \in \mathbb{N}$ there is $\mathfrak{A} \models T$ with $|A| \geqslant n$. Show T has an infinite model. E.g., there is no $\{E\}$-theory whose models are precisely the finite graphs.

Exercise 2.6.21. Find examples of satisfiable sentences without finite models (e.g. recall the ordering and pigeonhole principles from Section 1.8).

2.6.4 Gödel's first incompleteness theorem

The following result is outside the scope of this course but not hard to prove given a basic development of computability theory.

Theorem 2.6.22. *There is no algorithm deciding*

ARITHMETICAL TRUTH	
Input:	an L_{PA}-sentence φ.
Problem:	$\mathfrak{N} \models \varphi$?

This implies a weak version of Gödel's first incompleteness theorem. The full version concerns satisfiable instead of true (in $\mathfrak{N}$) theories.

Theorem 2.6.23 (Gödel's first incompleteness theorem)**.** *Assume T is a decidable L_{PA}-theory with $\mathfrak{N} \models T$. Then there exists an L_{PA}-sentence φ such that $T \nvdash \varphi$ and $T \nvdash \neg\varphi$.*

Proof. For contradiction, assume $T \vdash \varphi$ or $T \vdash \neg\varphi$ for every L_{PA}-sentence φ. Then $\mathfrak{N} \models \varphi$ if and only if $T \vdash \varphi$. Indeed, $\Leftarrow$ is clear by soundness and $\mathfrak{N} \models T$; $\Rightarrow$: if $\mathfrak{N} \models \varphi$, then $\mathfrak{N} \not\models \neg\varphi$, then $T \nvdash \neg\varphi$ (by $\Leftarrow$), so $T \vdash \varphi$ by assumption.

It suffices to give an algorithm deciding Arithmetical Truth. Run the algorithm from Exercise 2.6.19 in parallel on φ and $\neg\varphi$, i.e., do computation steps alternatingly on these two inputs. Exactly one of the two computations halts after a finite number of steps. If it is the one on φ output 1; if it is the one on $\neg\varphi$ output 0. $\square$

2.7 Universal theories

Skolemization reduces the satisfiability question to universal formulas. This gives special interest to universal formulas. We establish Łos and Tarski's semantic characterization of universal formulas and give an application in graph theory. We then give Herbrand's theorems showing that the satisfiability question for universal formulas is essentially propositional. This is the basis of logic programming (Section 2.9).

Let L be a language and $\mathfrak{A}$ be an L-structure with universe A.

2.7.1 Łos-Tarski theorem

Definition 2.7.1. Let $L(A) := L \cup \{c_a \mid a \in A\}$ where the c_a are pairwise distinct constants outside L. Let $\mathfrak{A}_A$ be the $L(A)$-expansion of $\mathfrak{A}$ that interprets c_a by a. The *algebraic diagram* $D_a(\mathfrak{A})$ *of* $\mathfrak{A}$ is a set of true (in $\mathfrak{A}_A$) $L(A)$-sentences, namely $\neg c_a \doteq c_{a'}$for $a \neq a'$ and those of the form

$$Rc_{a_1}\cdots c_{a_r},\ \neg Rc_{a_1}\cdots c_{a_r},\ fc_{a_1}\cdots c_{a_r} = c_a$$

where $r \in \mathbb{N}$, $f, R \in L$ are r-ary and $a_1, \ldots, a_r, a \in A$.

Lemma 2.7.2. *Let T be an L-theory. The following are equivalent.*

1. *$\mathfrak{A}$ satisfies every universal L-sentence φ implied by T.*
2. *$T \cup D_a(\mathfrak{A})$ is satisfiable.*
3. *Some extension of $\mathfrak{A}$ satisfies T.*

Proof. $1 \Rightarrow 2$: assume $T \cup D_a(\mathfrak{A})$ is unsatisfiable. By compactness, there are $\varphi_1, \ldots, \varphi_n \in D_a(\mathfrak{A})$ such that $T \cup \{\varphi_1, \ldots, \varphi_n\}$ is unsatisfiable. Write $\varphi_i = \varphi_i(\bar{c})$ for L-formulas $\varphi_i'(\bar{x})$ and $\bar{c}$ constants in $L(A) \setminus L$. Then T implies $\bigvee_{i=1}^n \neg\varphi_i'(\bar{c})$, hence also $\forall\bar{x} \bigvee_{i=1}^n \neg\varphi(\bar{x})$ (Exercise 2.5.5). This is a universal L-sentence that is false in $\mathfrak{A}$.

$2 \Rightarrow 3$: let $\mathfrak{C} \models T \cup D_a(\mathfrak{A})$. It suffices to show that $\mathfrak{C}\restriction L$ is isomorphic to an extension $\mathfrak{B} \supseteq \mathfrak{A}$. By Exercise 2.1.14 it suffices to show that there is an embedding of $\mathfrak{A}$ into $\mathfrak{C}\restriction L$.

We claim that the map $a \mapsto c_a^{\mathfrak{C}}$ is one. It is injective because $\neg c_a \doteq c_{a'} \in D_a(\mathfrak{A})$, so $\mathfrak{C} \models \neg c_a \doteq c_{a'}$, so $c_a^{\mathfrak{C}} \neq c_{a'}^{\mathfrak{C}}$ for $a \neq a'$.

For an r-ary function symbol $f \in L$, we have to show that $a := f^{\mathfrak{A}}(a_1, \ldots, a_r)$ is mapped to $f^{\mathfrak{C}}(c_{a_1}^{\mathfrak{C}}, \ldots, c_{a_r}^{\mathfrak{C}})$, i.e., $c_a^{\mathfrak{A}} = f^{\mathfrak{C}}(c_{a_1}^{\mathfrak{C}}, \ldots, c_{a_r}^{\mathfrak{C}})$, i.e., the sentence $c_a \doteq fc_{a_1}\cdots c_{a_r}$ is true in $\mathfrak{C}$; this holds because it is in $D_a(\mathfrak{A})$.

For an r-ary relation symbol $R \in L$, we show $(a_1, \dots, a_r) \in R^{\mathfrak{A}} \Leftrightarrow (c_{a_1}^{\mathfrak{C}}, \dots c_{a_r}^{\mathfrak{C}}) \in R^{\mathfrak{C} \upharpoonright L}$, i.e., $\mathfrak{A}_A \vDash Rc_{a_1} \cdots c_{a_r} \Leftrightarrow \mathfrak{C} \vDash Rc_{a_1} \cdots c_{a_r}$. This follows from $Rc_{a_1} \cdots c_{a_r}$, $\neg Rc_{a_1} \cdots c_{a_r} \in D_a(\mathfrak{A})$.

$3 \Rightarrow 1$: if $\mathfrak{A} \subseteq \mathfrak{B} \vDash T$ and $T \vDash \varphi$, then $\mathfrak{B} \vDash \varphi$. If φ is universal, then $\mathfrak{A} \vDash \varphi$ (Exercise 2.3.21). □

Exercise 2.7.3. Let $L' \supseteq L$ be a language such that $L' \setminus L$ contains only relation symbols. Let T' be a universal L'-theory. The following are equivalent.

1. $\mathfrak{A}$ satisfies every universal L-sentence φ implied by T'.
2. $\mathfrak{A}$ has an expansion $\mathfrak{A}' \vDash T'$.

Theorem 2.7.4 (Łos-Tarski). *Let T be an L-theory and $\varphi(\bar{x})$ be an L-formula. The following are equivalent.*

1. *For all $\mathfrak{A}, \mathfrak{B} \vDash T$ and all tuples $\bar{a}$ from A of suitable length:*

$$\mathfrak{A} \subseteq \mathfrak{B} \vDash \varphi[\bar{a}] \quad \Longrightarrow \quad \mathfrak{A} \vDash \varphi[\bar{a}].$$

2. *There exists a universal L-formula $\psi(\bar{x})$ such that $T \vDash \forall \bar{x}(\varphi(\bar{x}) \leftrightarrow \psi(\bar{x}))$.*

Proof. $2 \Rightarrow 1$: if $\mathfrak{B} \vDash \varphi[\bar{a}]$, then $\mathfrak{B} \vDash \psi[\bar{a}]$ since $\mathfrak{B} \vDash T$, then $\mathfrak{A} \vDash \psi[\bar{a}]$ by Exercise 2.3.21, then $\mathfrak{A} \vDash \varphi[\bar{a}]$ since $\mathfrak{A} \vDash\ T$.

$1 \Rightarrow 2$: choose new constants $\bar{c}$. Then for all $L' := L \cup \{\bar{c}\}$-structures $\mathfrak{A}, \mathfrak{B} \vDash T$ we have

$$\mathfrak{A} \subseteq \mathfrak{B} \vDash \varphi(\bar{c}) \implies \mathfrak{A} \vDash \varphi(\bar{c}).$$

Let U be the set of L'-sentences implied by $T' := T \cup \{\varphi(\bar{c})\}$. We claim:

$$T \cup U \vDash \varphi(\bar{c}).$$

Let $\mathfrak{A} \vDash T \cup U$. The lemma gives an L'-structure $\mathfrak{B}$ with $\mathfrak{A} \subseteq \mathfrak{B} \vDash T'$. This implies $\mathfrak{A} \vDash \varphi(\bar{c})$.

Then $T \cup U \cup \{\neg\varphi(\bar{c})\}$ is unsatisfiable. By compactness there is a finite $U_0 \subseteq U$ such that $T \cup U_0 \cup \{\neg\varphi(\bar{c})\}$ is unsatisfiable. Then $T \vDash (\bigwedge U_0 \to \varphi(\bar{c}))$. As $T \vDash (\varphi(\bar{c}) \to \psi)$ for all $\psi \in U$, we have $T \vDash (\bigwedge U_0 \leftrightarrow \varphi(\bar{c}))$. But $\bigwedge U_0$ is equivalent to $\psi(\bar{c})$ for some universal L-formula $\psi(\bar{x})$. By Exercise 2.5.5, $T \vDash \forall \bar{x}(\varphi(\bar{x}) \leftrightarrow \psi(\bar{x}))$. □

2.7.2 Forbidden subgraph characterizations

Suppose L is finite and relational (contains only relation symbols). For L-structures $\mathfrak{A}, \mathfrak{B}$ let $\mathfrak{B} \hookrightarrow \mathfrak{A}$ mean that there is an embedding of $\mathfrak{B}$ into $\mathfrak{A}$.

Proposition 2.7.5. *Let φ a universal L-sentence. There are $n \in \mathbb{N}$ and finite L-stuctures $\mathfrak{B}_1, \dots, \mathfrak{B}_n$ such that for all L-structures $\mathfrak{A}$:*

$$\mathfrak{A} \vDash \varphi \iff \mathfrak{B}_1 \not\hookrightarrow \mathfrak{A}, \dots, \mathfrak{B}_n \not\hookrightarrow \mathfrak{A}.$$

Proof. Write $\varphi = \forall \bar{x}\psi(\bar{x})$ for quantifier free $\psi(\bar{x})$. If $\mathfrak{A} \nvDash \varphi$, there is a tuple $\bar{a}$ over A of the length of $\bar{x}$ such that $\mathfrak{A} \vDash \neg\psi[\bar{a}]$. Then $\langle\bar{a}\rangle^{\mathfrak{A}} \vDash \neg\psi[\bar{a}]$. Let the $\mathfrak{B}_i$ list $\langle\bar{a}\rangle^{\mathfrak{A}}$ for all such $\mathfrak{A}, \bar{a}$ – up to isomorphism. □

Example 2.7.6. Recall, graphs are $\{E\}$-structures $\mathfrak{G}$ with $\mathfrak{G} \vDash \varphi_{graph} := \forall xy(\neg Exx \wedge (Exy \to Eyx))$. Equivalently, $\overset{\bullet}{\circlearrowleft}, \bullet\!\to\!\bullet \not\hookrightarrow \mathfrak{G}$ (where we depict E by $\to$).

We now focus on graphs. A prominent topic of graph theory is to characterize graph properties by forbidden graphs in one or another sense.

Definition 2.7.7. Let $\mathcal{P}$ be a class of graphs.

1. $\mathcal{P}$ is *subgraph closed* if it contains all subgraphs of every graph $\mathfrak{G} \in \mathcal{P}$.
2. $\mathcal{P}$ is *definable* if there is an $\{E\}$-sentence φ such that for all graphs $\mathfrak{G}$:
$$\mathfrak{G} \in \mathcal{P} \iff \mathfrak{G} \vDash \varphi.$$
3. Let $\mathcal{F}$ be a set of graphs. $\mathcal{P}$ is *characterized by forbidding* $\mathcal{F}$ if for all graphs $\mathfrak{G}$:
$$\mathfrak{G} \in \mathcal{P} \iff \text{no } \mathfrak{H} \in \mathcal{F} \text{ is isomorphic to a subgraph of } \mathfrak{G}.$$

Theorem 2.7.8. *Assume $\mathcal{P}$ is a class of graphs that is subgraph closed and definable. Then there is a finite set $\mathcal{F}$ of finite graphs such that $\mathcal{P}$ is characterized by forbidding $\mathcal{F}$.*

Proof. Let φ define $\mathcal{P}$. As $\mathcal{P}$ is closed under subgraphs implies it is closed under induced subgraphs, so closed under substructures – i.a.w., Theorem 2.7.4 (1)holds for φ and $T := \{\varphi_{Graph}\}$, Thus, there is a universal φ such that $\{\varphi_{Graph}\} \vDash (\varphi \leftrightarrow \psi)$, so the sentence $((\varphi \wedge \varphi_{Graph}) \leftrightarrow (\varphi_{Graph} \wedge \psi))$ is valid. Let χ be universal and equivalent to $(\varphi_{Graph} \wedge \psi)$. Then $\mathcal{P}$ is the class of models of χ.

Choose a list $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ according to Proposition 2.7.5 of $\{E\}$-structures. Delete $\mathfrak{B}_i$ if it is not a graph, or there is $j \neq i$ such that $\mathfrak{B}_j$ is isomorphic to a proper subgraph of $\mathfrak{B}_i$. Let $\mathfrak{B}_{i_1}, \ldots, \mathfrak{B}_{i_\ell}$ be the new (sub)list of finite graphs.

Let $\mathfrak{G}$ be a graph. Then $\mathfrak{G} \nvDash \chi$ if and only if $\mathfrak{B}_i \hookrightarrow \mathfrak{G}$ for some $i \in [n]$. We claim this holds if and only if there is $j \in [\ell]$ such that $\mathfrak{B}_{i_j}$ is isomorphic to some subgraph of $\mathfrak{G}$.

$\Rightarrow$: if $\mathfrak{B}_i \hookrightarrow \mathfrak{G}$, then $\mathfrak{B}_i$ is a graph: if $\mathfrak{B}_i \cong \mathfrak{H} \subseteq \mathfrak{G}$, then $\mathfrak{H} \vDash \varphi_{graph}$, so $\mathfrak{B}_i \vDash \varphi_{graph}$ (Exercises 2.3.21, 2.3.19). Hence, some $\mathfrak{B}_{i_j}$ is isomorphic to a subgraph of $\mathfrak{B}_i$. Then $\mathfrak{B}_{i_j}$ is isomorphic to some subgraph of $\mathfrak{H}$, hence of $\mathfrak{G}$.

$\Leftarrow$: if $\mathfrak{B}_{i_j}$ is isomorphic to some subgraph of $\mathfrak{G}$, then it is isomorphic to an induced subgraph of some subgraph $\mathfrak{G}'$ of $\mathfrak{G}$, i.e., $\mathfrak{B}_{i_j} \hookrightarrow \mathfrak{G}'$. Then $\mathfrak{G}' \nvDash \chi$, so $\mathfrak{G}' \notin \mathcal{P}$. As $\mathcal{P}$ is subgraph closed, $\mathfrak{G} \notin \mathcal{P}$, i.e., $\mathfrak{G} \nvDash \chi$. □

Example 2.7.9 (Lovász)**.** Let $k > 0$ and $\mathcal{P}_k$ be the class of graphs that contain a vertex cover of size $\leqslant k$. Then there exists a finite set $\mathcal{F}_k$ of finite graphs such that $\mathcal{P}_k$ is characterized by forbidding $\mathcal{F}_k$.

Proof. $\mathcal{P}_k$ is subgraph closed and defined by $\exists x_1 \cdots x_k \forall yz(Eyz \to \bigvee_{i=1}^{k}(x_i \doteq y \vee x_i \doteq z))$. □

2.7.3 Herbrand's theorems

Let L be a language that contains at least one constant.

Theorem 2.7.10. *Let T be a universal L-theory and $\varphi(\bar{x}, y)$ a quantifier free L-formula. If T implies $\forall\bar{x}\exists y\varphi(\bar{x}, y)$, then there are $n > 0$ and L-terms $t_1(\bar{x}), \dots, t_n(\bar{x})$ such that T implies $\forall\bar{x}\bigvee_{i=1}^n \varphi(\bar{x}, t_i(\bar{x}))$.*

Proof. Let $\bar{c}$ be a tuple of constants outside L. It suffices to show

$$T^* := T \cup \{\neg\varphi(\bar{c}, t(\bar{c})) \mid t(\bar{x}) \text{ is an } L\text{-term}\}$$

is unsatisfiable: then compactness gives $n > 0$ and L-terms $t_1(\bar{x}), \dots, t_n(\bar{x})$ such that $T \cup \{\neg\varphi(\bar{c}, t_1(\bar{c})), \dots, \neg\varphi(\bar{c}, t_n(\bar{c}))\}$ is unsatisfiable, i.e., T implies $\bigvee_{i=1}^n \varphi(\bar{c}, t_i(\bar{c}))$; as $\bar{c}$ does not appear in T, this yields our claim (Exercise 2.5.5).

For contradiction, assume there is an $L \cup \{\bar{c}\}$-structure $\mathfrak{A} \models T^*$. Since $L \cup \{\bar{c}\}$ contains at least one constant, $\langle\varnothing\rangle^{\mathfrak{A}} \subseteq \mathfrak{A}$ is defined. Its universe is the set of $t^{\mathfrak{A}}$ where t is a closed $L\cup\{\bar{c}\}$-term. As T is universal, $\langle\varnothing\rangle^{\mathfrak{A}} \models T$ (Exercise 2.3.21). Then $\langle\varnothing\rangle^{\mathfrak{A}} \models \exists y\varphi(\bar{c}, y)$, so there is $t^{\mathfrak{A}}$ satisfying $\varphi(\bar{c}, x)$ in $\langle\varnothing\rangle^{\mathfrak{A}}$, so $\langle\varnothing\rangle^{\mathfrak{A}} \models \varphi(\bar{c}, t)$. Write $t = t'(\bar{c})$ for some L-term $t'(\bar{x})$. Since φ is quantifier free, $\mathfrak{A} \models \varphi(\bar{c}, t'(\bar{c}))$. This contradicts $\mathfrak{A} \models T^*$. □

Remark 2.7.11. The above is true if L does not contain a constant and $\bar{x}$ is non-empty. We shall see important situations where we can ensure $n = 1$ (Corollary 2.7.20, Theorem 2.9.2).

Exercise 2.7.12. Formulate and prove a version of the above for a tuple of variables $\bar{y}$ instead of a single variable y.

Observe that $T := \varnothing \models \forall\bar{x}\exists y\varphi(\bar{x}, y)$ means that the existential formula $\exists y\varphi(\bar{x}, y)$ is valid, and the above yields that the quantifier free formula $\bigvee_{i=1}^n \varphi(\bar{x}, t_i(\bar{x}))$ is valid. We now aim to show that this validity is essentially propositional validity. We choose to equivalently talk about unsatisfiable universal formulas instead of valid existential ones.

Definition 2.7.13. Let $\psi(x_1, \dots, x_k)$ be a quantifier free L-formula. The *Herbrand expansion of* $\forall\bar{x}\psi$ is the set

$$\mathcal{H}(\forall\bar{x}\psi) := \{\psi(t_1, \dots, t_k) \mid t_1, \dots, t_k \text{ closed } L\text{-terms}\}.$$

We also set $\mathcal{H}(\psi(\bar{x})) := \mathcal{H}(\forall\bar{x}\psi)$ and $\mathcal{H}(T) := \bigcup_{\varphi\in T} \mathcal{H}(\varphi)$ for a universal L-theory T.

Definition 2.7.14. A quantifier free L-sentence φ is *propositionally satisfiable* if it is satisfiable as a propositional formula over propositional variables Var_L, the set of atomic L-sentences. A set of quantifier free sentences is *propositionally satisfiable* if it is satisfiable viewed as a set of propositional formulas over Var_L.

Remark 2.7.15. Clearly, satisfiable implies propositionally satisfiable – but not vice-versa: e.g., for distinct constants c, d, both $\neg c \doteq c$ and $(c \doteq d \wedge \neg d \doteq c)$ are unsatisfiable but propositionally satisfiable. In the $\doteq$-free case this cannot happen:

Lemma 2.7.16. *Let T be a set of universal $\doteq$-free L-sentences. Then T is satisfiable if and only if $\mathcal{H}(T)$ is propositionally satisfiable.*

Proof. If $\mathfrak{A} \models T$, then $\mathfrak{A} \models \psi$ for all $\psi \in \mathcal{H}(T)$. Define an assignment β to Var_L mapping every $\chi \in Var_L$ to its truth value in $\mathfrak{A}$. A straightforward induction shows that a quantifier free sentence ψ is propositionally satisfied by β if and only if it is true in $\mathfrak{A}$.

Conversely, if $\beta : Var_L \to \{0,1\}$ propositionally satisfies $\mathcal{H}(T)$, define $\mathfrak{A}$ with universe A the set of closed L-terms as follows: for an r-ary function symbol $f \in L$ let $f^{\mathfrak{A}}$ map $t_1, \ldots, t_r \in A$ to $ft_1 \cdots t_r \in A$; for an r-ary relation symbol $R \in L$ set

$$R^{\mathfrak{A}} := \big\{(t_1, \ldots, t_r) \in A^r \mid \beta(Rt_1 \cdots t_r) = 1\big\}.$$

Let $\varphi = \forall \bar{x} \psi(\bar{x}) \in T$ for quantifier free $\psi(\bar{x})$. We have to show $\mathfrak{A} \models \psi[\bar{t}]$, i.e., $\mathfrak{A} \models \psi(\bar{t})$ for all tuples $\bar{t}$ from A. As $\psi(\bar{t}) \in \mathcal{H}(\varphi) \subseteq \mathcal{H}(T)$ it suffices to show that a quantifier free $\doteq$-free sentence ψ is propositionally satisfied by β if and only if $\mathfrak{A} \models \psi$ – an easy induction. □

Recall the set of equality axioms Eq_L from Example 2.4.9 and note they are universal.

Theorem 2.7.17. *A universal L-sentence φ is unsatisfiable if and only if some finite subset of $\mathcal{H}(Eq_L \cup \{\varphi\})$ is propositionally unsatisfiable.*

Proof. The following are equivalent: φ is unsatisfiable, $\neg\varphi$ is valid, Eq_L^E implies $\neg\varphi^E$ (Corollary 2.6.12), $Eq_L^E \cup \{\varphi^E\}$ is unsatisfiable, $\mathcal{H}(Eq_L^E \cup \{\varphi^E\})$ is propositionally unsatisfiable (Lemma 2.7.16). But this is just a copy of $\mathcal{H}(Eq_L \cup \{\varphi\})$. Now apply compactness of propositional logic (Theorem 1.4.9). □

Exercise 2.7.18. Formulate and prove an analogue for a universal L-theory T in place of φ. Infer Theorem 2.7.10 from it.

2.7.4 Search algorithms from proofs

Intuitively, Theorem 2.7.10 has the following computational reading. We have a search problem: given $\bar{x}$ compute y such that $\varphi(\bar{x}, y)$. We assume this is well-defined – for all $\bar{x}$ there exist such y. Suppose you know more than mere truth of this, namely that a true universal theory T implies $\forall \bar{x} \exists y \varphi(\bar{x}, y)$. Then you can solve the problem by computing the values $y_1 := t_1(\bar{x}), \ldots, y_n := t_n(\bar{x})$ and check which one works; note n is a constant, i.e., independent of the input. To give a precise version of this idea, recall Example 2.1.8.

Definition 2.7.19. $\forall PV$ is the set of universal PV-sentences true in $\mathfrak{E}$.

Below note $\mathfrak{E} \models \forall x Rxfx$ means that the efficient algorithm f solves the search problem associated to R.

Corollary 2.7.20. *Let $R \in PV$ be a binary relation symbol. If $\forall PV$ implies $\forall x \exists y Rxy$, then there is $f \in PV$ such that $\mathfrak{E} \models \forall x Rxfx$.*

Proof. By Theorem 2.7.10 there are PV-terms $t_1(x), \dots, t_n(x)$ such that $\bigvee_{i=1}^n Rxt_i(x)$ is true in $\mathfrak{E}$. Consider the following algorithm $\mathbb{A}$: on input $a \in \{0,1\}^*$ compute $b_1 := t_1^{\mathfrak{E}}[a], \dots, b_n := t_n^{\mathfrak{E}}[a]$; output the first b_i such that $(a, b_i) \in R^{\mathfrak{E}}$.

The b_i are computed by composing the efficient algorithms denoted by the symbols in t_i. Hence $\mathbb{A}$ is efficient and PV contains a symbol f for it. Then $\mathfrak{E} \vDash \forall x Rxfx$. □

Remark 2.7.21. PV stands for "polynomially verifiable" and goes back to Cook (1975). He defined an equational theory in the language PV based on Cobham's historical characterization of efficient algorithms (1965); his theory can be seen as a subset of $\forall PV$. A superset has been considered by DeMillo and Lipton (1979). Cook's work initiates the field of *Bounded Arithmetic* whose foundations were developed by Buss (1986).

2.8 Formal reasoning II: Resolution

Let L be a language that contains at least one constant. Let us sum up how to reduce the satisfiability problem for L-sentences to propositional satisfiability. Observe the equality axioms can be written as universal quantifications of clauses:

$$x \doteq x,\ (\neg x \doteq y \vee y \doteq x),\ (\neg x \doteq y \vee \neg y \doteq z \vee x \doteq z),$$
$$\big(\bigvee_{i=1}^r \neg x_i \doteq y_i \vee fx_1 \cdots x_r = fy_1 \cdots y_r\big),\ \big(\bigvee_{i=1}^r \neg x_i \doteq y_i \vee \neg Rx_1 \cdots x_r \vee Ry_1 \cdots y_r\big).$$

Corollary 2.8.1. *There is an algorithm that computes for every L-sentence φ an $\doteq$-free CNF φ' (in some language $L' \supseteq L$) such that φ is unsatisfiable if and only if $\mathcal{H}(\varphi')$ is propositionally unsatisfiable.*

Proof. We can assume L is finite. Compute a prenex $\psi \equiv \varphi$ (Exercise 2.4.6). Compute a Skolemization ψ^{Sk}, say in a finite language $L_1 \supseteq L_0$. By Corollary 2.4.12, $\psi^{Sk} \equiv \forall \bar{x} \chi(\bar{x})$ where $\chi(\bar{x})$ is a CNF – and it should be clear that such χ can be computed. As seen in the proof of Theorem 2.7.17, $\forall \bar{x} \chi(\bar{x})$ is unsatisfiable if and only if $\mathcal{H}(Eq_{L_1}^E \cup \{\chi^E\})$ is propositionally unsatisfiable. Output the conjunction φ' of χ^E and the above clauses for the equality axioms (with Exy instead $x \doteq y$). □

2.8.1 Gilmore's algorithm

Gilmore's algorithm solves Exercise 2.6.13 based on the above corollary using Resolution for propositional unsatisfiability checks. We restrict attention to the $\doteq$-free case and, as in the propositional setting, write clauses as sets of literals, i.e., from now on an *(L-)clause* C is a finite set of $\doteq$-free L-literals. Whenever using a clause C (a set of clauses $\mathcal{C}$) in a context where an L-formula is expected, we mean $\bigvee C$ (resp. $\bigwedge_{C \in \mathcal{C}} \bigvee C$) – but we also allow the empty clause (empty set of clauses) understanding it to be unsatisfiable (valid).

To be clear, we spell out the semantics:

- $\mathfrak{A} \vDash C[\beta]$ means $\mathfrak{A} \vDash \lambda[\beta]$ for some $\lambda \in C$. Note this never holds for the empty clause $C = \emptyset$. Further, $\mathfrak{A} \vDash \forall \bar{x} C[\beta]$ means $\mathfrak{A} \vDash C[\beta[\bar{x}/\bar{a}]$ for all tuples $\bar{a}$ from A.

– $\mathfrak{A} \vDash \mathcal{C}[\beta]$ means $\mathfrak{A} \vDash C[\beta]$ for all $C \in \mathcal{C}$. Note this trivially holds for $\mathcal{C} = \varnothing$. Further, $\mathfrak{A} \vDash \forall \bar{x} \mathcal{C}[\beta]$ means $\mathfrak{A} \vDash C[\beta[\bar{x}/\bar{a}]$ for all tuples $\bar{a}$ from A and all $C \in \mathcal{C}$.

We write $\mathcal{C} = \mathcal{C}(\bar{x}), C = C(\bar{x})$ to indicate that variables are among $\bar{x}$. The substitution σ of terms $t_1, \ldots, t_k$ for $\bar{x} = x_1 \cdots x_k$ in $C(\bar{x})$ gives $C^\sigma = C(t_1, \ldots, t_k) = \{\lambda(t_1, \ldots, t_k) \mid \lambda(\bar{x}) \in C(\bar{x})\}$. The Herbrand expansion of $\mathcal{C}(\bar{x})$ is

$$\mathcal{H}(\mathcal{C}) \quad \coloneqq \quad \big\{C(t_1, \ldots, t_k) \mid C(x_1, \ldots, x_k) \in \mathcal{C}, t_1, \ldots, t_k \text{ closed } L\text{-terms}\big\},$$

In other words, $\mathcal{H}(\mathcal{C})$ is the set of closed instances of clauses in $\mathcal{C}$:

Definition 2.8.2. A substitution σ is *closed* if its image contains only closed L-terms. A *closed instance* of a clause C is C^σ for a closed σ defined on all variables in C.

We view clauses without variables as propositional clauses where the propositional variables are atomic L-sentences. We refer to a cut of two such clauses as a *propositional cut.*

Proposition 2.8.3. *There is an algorithm that, given a set of clauses $\mathcal{C}(\bar{x})$, halts in a finite number of steps if and only if $\forall \bar{x} \mathcal{C}(\bar{x})$ is unsatisfiable.*

Proof. By Lemma 2.7.16, $\forall \bar{x} \mathcal{C}(\bar{x})$ is unsatisfiable if and only if $\mathcal{H}(\mathcal{C})$ is propositionally unsatisfiable. Let $C_1, C_2 \ldots$ be an *effective* enumeration of $\mathcal{H}(\mathcal{C})$, i.e., $i \mapsto C_i$ is computable.

```
1. $i \leftarrow 1,\ \mathcal{D} \leftarrow \{C_1\}$
2. while $\varnothing \notin \mathcal{D}$
3.     $i \leftarrow i + 1,\ \mathcal{D} \leftarrow \mathcal{D} \cup \{C_{i+1}\}$
4.     $\mathcal{D} \leftarrow \mathcal{D} \cup$ set of propositional cuts of clauses $C, C' \in \mathcal{D}$
```

This algorithm halts only if $\forall \bar{x} \mathcal{C}(\bar{x})$ is unsatisfiable. Indeed, if $\mathfrak{A} \vDash \forall \bar{x} \mathcal{C}(\bar{x})$, then $\mathfrak{A} \vDash C_i$ for all i, and clearly $\mathfrak{A}$ satisfies every propositional cut of clauses it satisfies.

Conversely, assume $\forall \bar{x} \mathcal{C}(\bar{x})$ and hence $\mathcal{H}(\mathcal{C})$ is unsatisfiable. By propositional compactness, there $n \in \mathbb{N}$ such that $\{C_1, \ldots, C_n\}$ is propositionally unsatisfiable. By Theorem 1.6.10 there is a Resolution refutation of $\{C_1, \ldots, C_n\}$, say of length ℓ. Then the algorithm halts latestly when i reaches $n + \ell$. □

Example 2.8.4. Let E, P be binary and unary relation symbols, f, g unary function symbols, and c, d constants. For readability we write $E(x, y), P(x)$ instead of Exy, Px. Consider the set of clauses $\mathcal{C}$:

$$\{E(x, y), E(fc, z)\},\ \{\neg E(fx, gy), P(fy)\},\ \{\neg P(x)\}.$$

Applying respectively the closed substitutions $[x/fc, y/gd, z/gd], [x/c, y/d], [x/fd]$ gives the following closed instances in $\mathcal{H}(\mathcal{C})$:

$$\{E(fc, gd)\},\ \{\neg E(fc, gd), P(fd)\},\ \{\neg P(fd)\}.$$

A propositional cut on the first two gives $\{P(fd)\}$, and another cut gives the empty clause $\varnothing$. Hence, $\forall xy \mathcal{C}(xy)$ is unsatisfiable.

Observe in the first cut we use substitutions that *unify* $E(x,y), E(fc,z), E(fx,gy)$. Instead of d we could also use c or $ffgfd$ or any other closed term – and why not a variable u to indicate this freedom? This leads to the notion of a *most general unifier*. Intuitively, this is a unifying substitution with maximal freedom. First-order Resolution is based on the idea to speed-up Gilmore's algorithm by avoiding the generation of many useless closed instances but instead performing cuts after applying most general unifiers.

2.8.2 First-order Resolution

Definition 2.8.5. The *composition* of substitutions

$$\sigma = [x_1/t_1, \ldots, x_\ell/t_\ell, y_1/s_1, \ldots, y_m/s_m] \text{ and } \sigma' = [x_1/t'_1, \ldots, x_\ell/t'_\ell, z_1/r_1, \ldots, z_n/r_n]$$

with x_i, y_j, z_k pairwise distinct, is the substitution

$$\sigma\sigma' := [x_1/t_1^{\sigma'}, \ldots, x_\ell/t_\ell^{\sigma'}, y_1/s_1^{\sigma'}, \ldots, y_m/s_m^{\sigma'}, z_1/r_1, \ldots, z_n/r_n].$$

A set of literals L is *unifiable* if there is a *unifier* of L, that is, a substitution σ such that $L^\sigma := \{\lambda^\sigma \mid \lambda \in L\}$ has size 1. A unifier σ of L is *most general* if for every unifier τ of L there is a substitution σ' such that $\tau = \sigma\sigma'$.

Note $C^{\sigma\sigma'} = (C^\sigma)^{\sigma'}$ for every clause C.

Lemma 2.8.6 (Unification)**.** *There is an efficient algorithm that, given a set L of literals, outputs a most general unifier of L in case L is unifiable, and otherwise outputs "fail".*

Proof. On input L the algorithm works as follows.

```
1. $\sigma \leftarrow$ the empty substitution
2. **while** $|L^\sigma| > 1$
3.    choose distinct $\lambda_1^\sigma, \lambda_2^\sigma \in L^\sigma$; $i \leftarrow$ first position $i$ where $\lambda_1^\sigma, \lambda_2^\sigma$ differ
4.    **if** none of $\lambda_1^\sigma, \lambda_2^\sigma$ has a variable at position $i$, **then** output "fail"
5.    **else** choose such a variable $x$ (in one of the literals);
         $t \leftarrow$ the term starting at position $i$ in the other literal
6.       **if** $x$ occurs in $t$, **then** output "fail".
7.       **else** $\sigma \leftarrow \sigma[x/t]$
```

Observe that, after each while loop, the variables in the domain of σ do not occur in the terms in its image. Hence each while loop decreases the number of variables in L. It follows that the algorithm is efficient. To prove correctness it suffices to show that it outputs a most general unifier given a unifiable L. Say, τ unifies L. It suffices to show that the while-loop maintains the property of σ that $\tau = \sigma\sigma'$ for some σ'.

Assume this for σ. Then σ' unifies L^σ, in particular $x^{\sigma'} = t^{\sigma'}$. Hence line 7 is reached and σ is updated to $\sigma[x/t]$. We claim $\tau = (\sigma[x/t])\sigma''$ where σ'' is the restriction of σ' to its domain without x. For a variable $y \neq x$ we have

$$y^\tau = (y^\sigma)^{\sigma'} = (y^{\sigma[x/t]})^{\sigma'} = (y^{\sigma[x/t]})^{\sigma''};$$

indeed, the 2nd holds because x does not occur in the terms in the image of σ;the 3rd holds because x does not occur in t hence not in $y^{\sigma[x/t]}$.

For variable x, note x does not occur in t and σ is not defined on x, so

$$x^\tau = x^{\sigma\sigma'} = x^{\sigma'} = t^{\sigma'} = t^{\sigma''} = x^{(\sigma[x/t])\sigma''}. \qquad \square$$

Definition 2.8.7. A *renaming* is a substitution whose image contains only variables. For a literal $\lambda = \neg^b\alpha$ with α atomic and $b \in \{0,1\}$ let $\bar\lambda := \neg^{1-b}\alpha$ denote the *complementary literal.* Let C_0, C_1 be clauses and ρ_0, ρ_1 be renamings so that $C_0^{\rho_0}, C_1^{\rho_1}$ do not share variables. Let $\lambda_1, \ldots, \lambda_n \in C_0^{\rho_0}, \mu_1, \ldots, \mu_m \in C_1^{\rho_1}$ and σ be a most general unifier of $\{\lambda_1, \ldots, \lambda_n, \bar\mu_1, \ldots, \bar\mu_m\}$. Then a *(first-order) cut of* C_0, C_1 is the clause

$$\left(C_0^{\rho_0} \smallsetminus \{\lambda_1, \ldots, \lambda_n\}\right)^\sigma \cup \left(C_1^{\rho_1} \smallsetminus \{\mu_1, \ldots, \mu_m\}\right)^\sigma.$$

A *(first-order) Resolution proof from* a set of clauses $\mathcal{C}$ is a sequence $C_1, \ldots, C_\ell$ of clauses, for some $\ell \in \mathbb{N}$, such that for every $i \in [\ell]$, C_i is a (first-order) cut of two clauses in $\mathcal{C} \cup \{C_j \mid j < i\}$. It is a proof *of* C_ℓ; if $C_\ell = \varnothing$ it is a *(first-order) Resolution refutation of* $\mathcal{C}$.

Example 2.8.8. In Example 2.8.4 we have $\{E(x,y), E(fc,z)\}$, $\{\neg E(fx',gy'), P(fy')\}$ after renaming. The following table shows a computation of a most general unifier of $L := \{E(x,y), E(fc,z), E(fx',gy')\}$; the position i is underlined.

L^σ			σ
$E(\underline{x},y)$	$E(\underline{f}c,z)$	$E(fx',gy')$	$[x/fc]$
$E(fc,\underline{y})$	$E(fc,\underline{z})$	$E(fx',gy')$	$[x/fc][y/z]$
	$E(f\underline{c},z)$	$E(f\underline{x}',gy')$	$[x/fc][y/z][x'/c]$
	$E(fc,\underline{z})$	$E(fc,\underline{g}y')$	$[x/fc][y/z][x'/c][z/gy']$
		$E(fc,gy')$	

A most general unifier is $\sigma = [x/fc, y/gy', x'/c, z/gy']$. Thus, $\{P(fy')\}^\sigma = \{P(fy')\}$ is a first-order cut of our clauses. In a picture:

$$\begin{tikzcd}
E(x,y), E(fc,z) \arrow[d, dashed, "\sigma"] & & \neg E(fx',gy'), P(fy') \arrow[d, dashed, "\sigma"] \\
E(fc,gy') \arrow[dr] & & \neg E(fc,gy'), P(fy') \arrow[dl] \\
& P(fy') &
\end{tikzcd}$$

In Example 2.8.4 we had the propositional cut $\{P(fd)\}$ of closed instances. Note $\{P(fd)\}$ is a closed instance of $\{P(fy')\}$. We show next that this always happens.

Lemma 2.8.9 (Lifting)**.** *Let C_0, C_1 be clauses and C' a propositional cut of closed instances C'_0, C'_1. Then there is a first-order cut C of C_0, C_1 such that C' is a closed instance of C.*

Proof. Let ρ_0, ρ_1 be renamings so that $C_0^{\rho_0}, C_1^{\rho_1}$ do not share variables. Choose closed substitutions τ_0, τ_1 such that $C'_0 = C_0^{\rho_0\tau_0}$ and $C'_1 = C_1^{\rho_1\tau_1}$. Since the domains are disjoint, we can replace τ_0, τ_1 by $\tau := \tau_0\tau_1$. Say $C' = (C'_0 \smallsetminus \{\lambda\}) \cup (C'_1 \cup \{\bar{\lambda}\})$ is a propositional cut on λ. Let $\lambda_1, \dots, \lambda_n$ list all literals in $C_0^{\rho_0}$ with $\lambda_i^\tau = \lambda$ and let $\mu_1, \dots, \mu_m$ list all literals in $C_0^{\rho_1}$ with $\mu_i^\tau = \bar{\lambda}$. Then τ unifies $\{\lambda_1, \dots, \lambda_n, \bar{\mu}_1, \dots, \bar{\mu}_m\}$. Let σ be a most general unifier. Then $\tau = \sigma\sigma'$ for some σ'. Then $C^{\sigma'} = C'$ for $C := (C_0^{\rho_0} \smallsetminus \{\lambda_1, \dots, \lambda_n\})^\sigma \cup (C_1^{\rho_1} \smallsetminus \{\mu_1, \dots, \mu_m\})^\sigma$. □

$$\begin{tikzcd}
C_0^{\rho_0} \arrow[d, dashed, "\sigma"] & & C_1^{\rho_1} \arrow[d, dashed, "\sigma"] \\
C_0^{\rho_0\sigma} \arrow[d, dashed, "\sigma'"] \arrow[dr] & & C_1^{\rho_1\sigma} \arrow[d, dashed, "\sigma'"] \arrow[dl] \\
C'_0 \arrow[dr] & C \arrow[d, dashed, "\sigma'"] & C'_1 \arrow[dl] \\
& C' &
\end{tikzcd}$$

Theorem 2.8.10 (Refutation completeness)**.** *Let $\mathcal{C}(\bar{x})$ be a set of clauses. Then $\forall\bar{x}\mathcal{C}(\bar{x})$ is unsatisfiable if and only if there exists a Resolution refutation of $\mathcal{C}(\bar{x})$.*

Proof. $\Leftarrow$ (Soundness): it suffices to show that, if $C(\bar{x})$ is a cut of $C_0(\bar{x}), C_1(\bar{x})$, then $\forall\bar{x}C(\bar{x})$ is implied by $\{\forall\bar{x}C_0(\bar{x}), \forall\bar{x}C_1(\bar{x})\}$. Assume $\mathfrak{A} \nvDash \forall\bar{x}C(\bar{x})$, say $\mathfrak{A} \nvDash C[\beta]$ for some assignment β. In the notation of Definition 2.8.7, let $\lambda = \lambda_i^\sigma = \bar{\mu}_j^\sigma$. We have $\mathfrak{A} \vDash \lambda[\beta]$ or not. Assume the 1st (the 2nd case is similar). Then $\mathfrak{A} \nvDash \mu_j^\sigma[\beta]$. As $(C_1^{\rho_1} \smallsetminus \{\mu_1, \dots, \mu_m\})^\sigma \subseteq C$ this clause is not satisfied in $\mathfrak{A}$ under β. Hence $\mathfrak{A} \nvDash C_1^{\rho_1\sigma}[\beta]$ and thus $\mathfrak{A} \nvDash \forall\bar{x}C_1(\bar{x})$.

$\Rightarrow$ (Completeness): if $\forall\bar{x}\mathcal{C}(\bar{x})$ is unsatisfiable, then there exists a propositional Resolution refutation $C'_1, \dots, C'_\ell$ of $\mathcal{H}(\mathcal{C})$ (recall the proof of Proposition 2.8.3). We can assume it contains no weakening steps (Lemma 1.6.6).

For $i = 1, \dots, \ell$ we find C_i such that C'_i is a closed instance of C_i and $C_1, \dots, C_\ell$ is a first-order Resolution refutation of $\mathcal{C}(\bar{x})$. If C'_i is a closed instance of some $C \in \mathcal{C}$, set $C_i := C$. If C'_i is a propositional cut of C'_j, C'_k with $j, k < i$, choose C_i by the lifting lemma. □

Exercise 2.8.11 (Russell's paradox)**.** The job of a barber is to shave exactly those persons who do not shave themselves. Does a barber shave himself?

Let Bx mean "x is a barber" and let Sxy mean "x shaves y". Formalize "Every barber shaves everybody who does not shave himself" and "No barber shaves somebody who shaves himself" by clauses. Use Resolution to show that barbers do not exist.

2.9 Logic programs

2.9.1 Clark's theorem

Again, we fix a language L containing at least one constant.

Definition 2.9.1. A literal is *negative* if it starts with $\neg$ and otherwise *positive*. A *goal clause* contains only negative literals. A *logic program* $\mathcal{P}$ is a set of clauses each containing exactly one positive literal.

Let $G = G(\bar{x}, y_1, \ldots, y_k)$ be a goal clause and $A \notin L$ a k-ary relation symbol. A *computation of* $\mathcal{P}$ *on* G is a sequence of clauses $C_1, \ldots, C_\ell$ with $C_1 = G \cup \{Ay_1 \cdots y_k\}, C_\ell = \{At_1 \cdots t_k\}$ where $\ell > 0$ and $t_1, \ldots, t_k$ are terms such that for all $i \in [\ell - 1]$, C_{i+1} is a first-order cut of C_i and some clause in $\mathcal{P}$. The *output* of the computation is $t_1 \cdots t_k$.

For $\bar{t} = t_1 \cdots t_k$ write $G(\bar{x}, \bar{t})$ for $G(\bar{x}, t_1, \ldots, t_k)$. Writing $\bar{t} = \bar{t}(\bar{x})$ means all t_i have variables among $\bar{x}$. Further, we write $\bar{t}^\sigma := t_1^\sigma \cdots t_k^\sigma$ for a substitution σ.

Intuition: say $k = 1$ and we know (or correctly hope) that, in a situation satisfying $\mathcal{P}$, there exists an object y satisfying a wishlist of properties. Our search problem is to compute such a y given input $\bar{x}$. Let the goal clause $G(\bar{x}, y)$ negate the wishlist, so $\forall \bar{x} \mathcal{P}$ implies $\exists y \neg \bigvee G$. Theorem 2.7.10 gives many terms $t(\bar{x})$ such that *some* $y := t(\bar{x})$ works – which one depends on $\bar{x}$. In the present context, we even get a single term $t(\bar{x})$ and $\mathcal{P}$ can compute it. In fact, $\mathcal{P}$ can compute *any* such $t(\bar{x})$:

Theorem 2.9.2 (Clark)**.** *Let $\mathcal{P}(\bar{x})$ be a logic program, and $G(\bar{x}, \bar{y})$ a goal clause such that $\forall \bar{x} \mathcal{P}(\bar{x})$ implies $\exists \bar{y} \neg \bigvee G(\bar{x}, \bar{y})$. Then there exists a computation of $\mathcal{P}$ on G. Moreover,*

1. *$\forall \bar{x} \mathcal{P}(\bar{x})$ implies $\neg \bigvee G(\bar{x}, \bar{t})$ for every output $\bar{t}$ of a computation of $\mathcal{P}$ on G.*
2. *If $\forall \bar{x} \mathcal{P}(\bar{x})$ implies $\neg \bigvee G(\bar{x}, \bar{s}(\bar{x}))$ for certain terms $\bar{s}(\bar{x})$, then there is a computation of $\mathcal{P}$ on G with some output $\bar{t}$ and a substitution σ such that $\bar{s} = \bar{t}^\sigma$.*

Proof. Let $\bar{c}$ be new constants and work in the language $L \cup \{\bar{c}\}$. Assume $\forall \bar{x} \mathcal{P}(\bar{x}) \cup \{\forall \bar{y} G(\bar{c}, \bar{y})\}$ is unsatisfiable. By Corollary 1.6.18 we get a propositional SLD refutation $C_1', \ldots, C_\ell'$ of $\mathcal{H}(\mathcal{P}) \cup \mathcal{H}(G)$. As C_1' is negative, it is in $\mathcal{H}(G)$, say $C_1 = G(\bar{c}, \bar{s})$ for certain closed $L \cup \{\bar{c}\}$-terms $\bar{s}$. As all C_i' are negative, the side clauses are in $\mathcal{H}(\mathcal{P})$. Add $A\bar{s}$ to all C_i' and, as in Theorem 2.8.10, lift to a first-order resolution proof $C_1, \ldots, C_\ell$. Then $C_\ell = \{A\bar{t}\}$ for certain $L \cup \{\bar{c}\}$-terms $\bar{t}$ and $\bar{t}^\sigma = \bar{s}$ for some σ. We can assume $\bar{x}$ appear nowhere, and replace $\bar{c}$ back by $\bar{x}$ to get L-terms.

2: $\forall \bar{x} \mathcal{P}(\bar{x}) \cup \{G(\bar{c}, \bar{s}(\bar{c}))\}$ is unsatisfiable, and $\mathcal{H}(G(\bar{c}, \bar{s}(\bar{c}))) = \{G(\bar{c}, \bar{s}(\bar{c}))\}$, so $C_1' = G(\bar{c}, \bar{s}(\bar{c}))$ above.

1: let $\bar{t}$ be the output of a computation of $\mathcal{P}$ on $G(\bar{x}, \bar{y})$. Then $\forall \bar{x} \mathcal{P}(\bar{x}) \cup \{\forall \bar{y}(G(\bar{x}, \bar{y}) \cup \{A\bar{y}\})\}$ implies $A\bar{t}$ by soundness (see Theorem 2.8.10). Let $\mathfrak{B} \models \forall \bar{x} \mathcal{P}(\bar{x})$ and β an assignment. Let $\bar{a}$ be the values of $\bar{t}$ in $\mathfrak{B}$ under β. Let $\mathfrak{B}'$ be the $L \cup \{A\}$-expansion of $\mathfrak{B}$ with $A^{\mathfrak{B}'} := B^k \setminus \{\bar{a}\}$. Then $\mathfrak{B}' \not\models A\bar{t}[\beta]$. Then there is $\bar{b} \in B^k$ such that $\mathfrak{B}' \models (\neg \bigvee G(\bar{x}, \bar{y}) \wedge \neg A\bar{y})[\beta[\bar{y}/\bar{b}]]$. Then $\bar{b} = \bar{a}$ and $\mathfrak{B}' \models \neg \bigvee G(\bar{x}, \bar{y})[\beta[\bar{y}/\bar{a}]]$, so $\mathfrak{B} \models \neg \bigvee G(\bar{x}, \bar{t})[\beta]$. □

Remark 2.9.3. PROLOG is a declarative programming language. The user writes a logic program and a goal and the algorithm looks for a computation. Albeit this is known to be uncomputable, various engineering tricks make it work well on real life instances.

2.9.2 Examples: computing, planning and proving

Many recursively defined functions are naturally evaluated by logic programs. In fact, it is known that every computable function can, in a reasonable sense, be computed by a logic program. We illustrate this for addition.

Example 2.9.4. Addition is recursively defined by $x + 0 = x, x + s(x) = s(x + y)$ where s is the successor. We write this as a logic program using a unary function symbol s, a constant $\underline{0}$ and a ternary relation symbol S intended for the graph of addition:

$$\{Sx\underline{0}x\}, \quad \{\neg Sxyz, Sxsysz\}.$$

We whish to compute "$u = 2 + 2$", so write the goal clause $G = \{\neg Sss0ss0u\}$. Set $C_1 := \{Au, \neg Sss\underline{0}ss\underline{0}u\}$ and $\sigma_1 := [x/ss\underline{0}, y/s\underline{0}, u/sz]$. This is a most general unifier of $Sss\underline{0}ss\underline{0}u$ and $Sxsysz$ from the 2nd program clause. A cut gives $C_2 := \{Asz, \neg Sss\underline{0}s\underline{0}z\}$. Next, rename the 2nd program clause to $\{\neg Sxyz', Sxsysz'\}$. Then $\sigma_2 := [x/ss\underline{0}, y/\underline{0}, z/sz']$ is a most general unifier of $Sss\underline{0}s\underline{0}z$ and $Sxsysz'$. A cut gives $C_3 := \{Assz', \neg Sss\underline{0}\underline{0}z'\}$. The 1st program clause with $\sigma_3 := [x/ss\underline{0}, z'/ss\underline{0}]$ gives the cut $C_4 := \{Assss\underline{0}\}$.

The output of the computation is $ssss\underline{0}$ – as expected.

Exercise 2.9.5. Write logic programs for multiplication, exponentiation, the Fibonacci sequence and the Ackermann function.

A planning problem is given by a set of situations and actions that change situations. It asks for a sequence of actions that change a given start situation to one out of given goal situations. Many planning problems can be naturally formulated and solved by logic programs. We illustrate this with the following frequently used toy problem.

Example 2.9.6 (Monkey-banana-problem)**.** A situation determines positions of the monkey, the chair and the banana (hanging from the top). The monkey has actions walk, push (the chair), jump (on the chair), and grasp (the banana) to change situations.

We use a 4-ary relation symbol P for the 3 positions determined by a situation, unary relation symbols Q, G indicating being on the chair or being a goal, constants a, b, c, s for positions of monkey, chair, banana in the start situation, and function symbols w, p, j, g for the actions; w, p are ternary, j, g unary. We use some extra parentheses for readability.

1. $\{Pabcs\}$ Monkey, banana, chair are at positions a, b, c in start situation s.
2. $\{\neg Pxyxu, Qju\}$ If the monkey is at the chair, it can jump on it.
3. $\{\neg Pxyzu, Pxyzju\}$ Jumping does not change positions.
4. $\{\neg Pxxxu, \neg Qu, Ggu\}$ The goal is reached, if monkey, chair and banana align and the monkey is on the chair and grasps.
5. $\{\neg Pxyzu, Px'yzw(uxx')\}$ The monkey can walk anywhere.
6. $\{\neg Pzyzu, Pz'yz'p(uzz')\}$ If the monkey is at the chair, then it can push it anywhere.

This is a logic program $\mathcal{P}$. Here is a computation of $\mathcal{P}$ on $\{\neg G(u)\}$:

computation	$\mathcal{P}$	σ
$Au, \neg Gu$	4	$[u/gu_0]$
$Agu_0, \neg Px_0x_0x_0u_0, \neg Qu_0$	2	$[u_0/ju_1]$
$Agju_1, \neg Px_0x_0x_0ju_1, \neg Px_1y_1x_1u_1$	3	$[x_2y_2z_2u_2/x_0x_0x_0u_1]$
$Agju_1, \neg Px_1y_1x_1u_1, \neg Px_0x_0x_0u_1$	6	$[x_1y_1z'_3y_3/x_0x_0x_0x_0][u_1/p(u_3z_3x_0)]$
$Agjp(u_3z_3x_0), \neg Pz_3x_0z_3u_3$	5	$[x'_4y_4z_4/z_3x_0z_3][u_3/w(u_4x_4z_3)]$
$Agjp(w(u_4x_4z_3)z_3x_0), \neg Px_4x_0z_3u_4$	1	$[x_4x_0z_3u_4/abcs]$
$Agjp(w(sac)cb)$		

The 1st column shows the computation, a sequence of goal clauses plus $A\cdots$. The 2nd column shows the clause from $\mathcal{P}$ used in each cut. We use canonical renamings of the program clauses: $\{\neg Px_0x_0x_0u_0, \neg Qu_0, Ggu_0\}$ in the 1st step, $\{\neg Px_1y_1x_1u_1, Qju_1\}$ in the 2nd, and so on. The 3rd column gives the most general unifier σ used in the cut to yield the clause in the next line. E.g. consider line 4: clause 6 is renamed to $\{\neg Pz_3y_3z_3u_3, Pz'_3y_3z'_3p(u_3z_3z'_3)\}$. The displayed σ unifies $Pz'_3y_3z'_3p(u_3z_3z'_3)$ with $Px_0x_0x_0u_1, Px_1y_1x_1u_1$ in line 3. Then the cut yields the clause in line 5.

The output is the term $gjp(w(sac)cb)$. It means: in the start situation s, walk from a to c, then push the chair from c to b, then jump, then grasp – as expected.

Finally, we illustrate the use of logic programs in automated theorem proving.

Example 2.9.7. We want to derive the existence of right inverses

$$\varphi := \exists x \forall uy \exists z (Pxuu \wedge Pyzx)$$

from the group axioms in Example 2.5.8. Write them as a logic program $\mathcal{P}$:

$$C_1 := \{Pxygxy\}, \begin{array}{l} C_2 := \{\neg Pxyu, \neg Pyzv, \neg Pxvw, Puzw\} \\ C'_2 := \{\neg Pxyu, \neg Pyzv, \neg Puzw, Pxvw\} \end{array}, \ C_3 := \{Peuu\}, \ C_4 := \{Piyye\}.$$

Prenex and skolemize $\neg\varphi$ to get the goal clause $G = \{\neg Pxjxjx, \neg Pkxzx\}$ with new unary function symbols k, j. It suffices to refute $\mathcal{P} \cup \{G\}$, i.e., to find a computation of $\mathcal{P}$ on G. Here it is – omitting the 0-ary A:

computation	$\mathcal{P}$	σ
$\neg Pxjxjx, \neg Pkxzx$	C_3	$[x/e, u/jx]$
$\neg Pkeze$	$C_2[z/z']$	$[u/ke, z'/z, w/e]$
$\neg Pxyke, \neg Pyzv, \neg Pxve$	$C_4[y/y']$	$[u/ke, z'/z, w/e]$
$\neg Piy'yke, \neg Pyzv$	C_3	$[x/iy', v/e, w/ke]$
$\neg Piy'eke$	C'_2	$[x/iy', v/e, w/ke]$
$\neg Piy'yu, \neg Pyze, \neg Puzke$	$C_3[u/u'] = \{Peu'u'\}$	$[u'/ke, z/ke, u/e]$
$\neg Piy'ye, \neg Pykee$	$C_4[y/z] = \{Pizze\}$	$[y/ike, z/ke]$
$\neg Piy'ikee$	C_4	$[y/ike, y'/ike]$
$\varnothing$		

Chapter 3

Model-Checking

Generically speaking *model-checking* refers to the study of the computational complexity of the problem to decide whether $W \models \varphi$ for a given world W and a given sentence φ of some logic. Typically the problem is computationally hard and one asks for efficiently solvable restrictions to a class of worlds $\mathcal{W}$ and a set of sentences Φ, i.e, considering only inputs with $W \in \mathcal{W}$ and $\varphi \in \Phi$.

In computer science model-checking is studied from two main perspectives: database theory and formal verification. In database theory, $\mathcal{W}$ is a class of databases, i.e., relational structures (recall Example 2.1.7) and Φ a set of *queries*, typically formalized in first-order logic. In formal verification $\mathcal{W}$ is a class of reactive or concurrent systems, and Φ a collection of correctness specifications one intends the systems to satisfy.

These two perspectives lead into orthogonal directions to look for efficiently solvable restrictions of interest. As a rule of thump, the perspective from database theory aims at rich classes $\mathcal{W}$ and targets efficiency for highly restrictive classes Φ of first-order formulas; from the formal verification perspective very special structures $\mathcal{W}$ fall on the table and one aims at strong logics tailored to reason about the reactive systems at hand.

The bad news is that, unless P = NP, there are no efficient algorithms even when severely restricting both $\mathcal{W}$ and Φ. But there are also good news: for example, we shall find a model-checker for linear time temporal logic over concurrent systems that runs in time $2^{O(|\varphi|)} \cdot |W|$ even though the problem is NP-hard (even PSPACE-complete). Such a runtime can be considered feasible and, in fact, works well in practice. The point is that, from both perspectives, typical instances of the problem have a large W and a relatively small φ. An adequate complexity analysis has to take this asymmetry into account. We do so, aiming at runtimes that might depend badly on $|\varphi|$ but not on $|W|$.

Hardness results in terms of classical complexity theory like the PSPACE-completeness mentioned above are meaningless. It is *parameterized complexity* that provides an adequate theoretical frame but a development of this theory is outside the scope of this course.

3.1 Monadic second-order logic

We define *monadic second order logic* as an extension of first-order logic by quantifiers $\exists X$ that range over subsets of the universe. Full second-order logic would also quantify over relations X of higher arity. The definition is straightforward adding new rules how to construct formulas and how to evaluate them. Recall Definitions 2.2.5 and 2.3.8.

Definition 3.1.1 (Monadic second-oder logic)**.** Let L be a language. Add to the alphabet of first-oder L-formulas a set VAR of *set variables* $X_1, X_2, \ldots$. The set of *monadic second order (MSO) L-formulas* is the smallest set F satisfying (F1)-(F5) and

(F6) if $X \in \mathit{VAR}$ and t is an L-term, then $Xt \in F$;

(F7) if $\varphi \in F$ and X is a set variable, then $\exists X\varphi \in F$.

Let $\mathfrak{A}$ be an L-structure. An *(MSO) assignment in* $\mathfrak{A}$ is a function β with domain $\mathit{Var} \cup \mathit{VAR}$ that maps every individual variable $x \in \mathit{Var}$ to an element $\beta(x) \in A$ and every set variable $X \in \mathit{VAR}$ to a subset $\beta(X) \subseteq A$.

For MSO L-formulas, we define $\mathfrak{A} \models \varphi[\beta]$ by (T1)-(T4) and

(T5) if $\varphi = Xt$ for $X \in \mathit{VAR}$ and t an L-term, then

$$\mathfrak{A} \models \varphi[\beta] \iff t^{\mathfrak{A}}[\beta] \in \beta(X)$$

(T6) if $\varphi = \exists X\psi$ for $X \in \mathit{VAR}$ and MSO L-formula ψ, then

$$\mathfrak{A} \models \varphi[\beta] \iff \text{there is } B \subseteq A\text{: } \mathfrak{A} \models \psi\big[\beta[X/B]\big],$$

where $\beta[X/B]$ is the MSO assignment that maps X to B and otherwise agrees with β.

Notation: For $X \in \mathit{VAR}$ we let $\forall X\varphi$ abbreviate $\neg\exists X\neg\varphi$. Then

$$\mathfrak{A} \models \forall X\varphi[\beta] \iff \text{for all } B \subseteq A\text{: } \mathfrak{A} \models \varphi[\beta[X/B]]$$

Remark 3.1.2. The Lemma 2.2.6 on unique readability is adjusted adding case 6: $\varphi = \exists X\psi$ for some set variable and an MSO L-formula ψ. This justifies the definition of $\models$ by recursion on syntax. Similarly, one defines the set of free variables $\mathit{free}(\varphi) \subseteq \mathit{Var} \cup \mathit{VAR}$ of an MSO L-formula φ as before plus: if $\varphi = \exists X\psi$ then $\mathit{free}(\varphi) := \mathit{free}(\psi) \setminus \{X\}$. The coincidence Lemmas 2.3.12, 2.3.13 hold for MSO L-formulas with the same proof.

Let $\bar{x} = x_1 \cdots x_n$ and $\bar{X} = X_1 \cdots X_m$ be tuples from Var (individual variables) and VAR (set variables). We write an MSO formula φ and $\varphi(\bar{X}, \bar{x})$ to indicate that the free variables of φ are among $\bar{X}, \bar{x}$. If $\bar{a} = (a_1, \ldots, a_n) \in A^n$ and $\bar{B} = (B_1, \ldots, B_m) \in P(A)^m$ we write

$$\mathfrak{A} \models \varphi[\bar{B}, \bar{a}]$$

to express $\mathfrak{A} \models \varphi[\beta]$ for some (equivalently, all) MSO assignments β in $\mathfrak{A}$ mapping x_i to a_i and X_j to B_j. An *MSO L-sentence* is a MSO L-formula φ with $\mathit{free}(\varphi) = \varnothing$. We write $\mathfrak{A} \models \varphi$ and say φ is *true in* $\mathfrak{A}$ if $\mathfrak{A} \models \varphi[\beta]$ for some (equivalently, all) β.

Example 3.1.3. Recall Example 2.6.18. Define

$$\varphi := \forall xy \forall X \big(Xx \wedge \forall uv((Xu \wedge Euv) \to Xv)) \to Xy \big).$$

Intuitively, $\forall uv((Xu \wedge Euv) \to Xv)$ states that X is closed under taking E-steps; that *every* such X containing x also contains y means that there is a path from x to y.

More precisely, φ is true in a graph $\mathfrak{G}$ if and only if $\mathfrak{G}$ is connected.

This example shows that MSO is more expressive than first-oder logic. It also shows that the compactness theorem is false for MSO. Also the upward Löwenheim-Skolem Theorem 2.6.17 fails for MSO:

Exercise 3.1.4. There is an MSO L_{PA}-sentence $\varphi_{\mathfrak{N}}$ that is true in an L_{PA}-structure $\mathfrak{A}$ if and only if $\mathfrak{A} \cong \mathfrak{N}$.

Hint: $\varphi_{\mathfrak{N}}$ states that $+, \cdot$ satisfy their recursive definitions and

$$\forall X((X\underline{0} \wedge \forall x(Xx \to Xsx)) \to \forall x Xx).$$

Corollary 3.1.5. *There is no algorithm that, given an MSO L_{PA}-sentence φ, halts in a finite number of steps if and only if φ is valid (i.e., true in all L_{PA}-structures).*

Proof. If $\mathbb{A}$ is such an algorithm, we can decide ARITHMETICAL TRUTH contradicting Theorem 2.6.22: given a first-order L_{PA}-sentence φ, run $\mathbb{A}$ in parallel on $(\varphi_{\mathfrak{N}} \to \varphi)$ and $(\varphi_{\mathfrak{N}} \to \neg\varphi)$, i.e., do steps of both computations alternatingly.

To see correctness, observe that $(\varphi_{\mathfrak{N}} \to \varphi)$ is valid if and only $\mathfrak{N} \models \varphi$. □

Informally speaking, this implies that there does not exist a sound and complete calculus for MSO. Hence the gain in expressiveness comes at a huge price, in some sense, MSO is out of control. However, restricting to certain classes of special structures, important for computer science, some control can be established.

In particular, this is so for word structures $\mathfrak{S}(w)$ – recall Example 2.1.6. Recall, an *alphabet* A is a non-empty set of *letters*, and a *language* L is a set of non-empty words, i.e., $L \subseteq A^+ := A^* \setminus \{\epsilon\}$. The restriction to non-empty words is inessential but convenient because $\mathfrak{S}(\epsilon)$ is not defined.

Definition 3.1.6. Let A be an alphabet. A language $L \subseteq A^+$ is *MSO-definable* if $L = L(\varphi)$ for some MSO L_A-sentence φ. Here, $L(\varphi)$ is the *language defined by* φ:

$$L(\varphi) := \big\{ w \in A^+ \mid \mathfrak{S}(w) \models \varphi \big\}.$$

Exercise 3.1.7. Show that the set PAR $\subseteq \{0,1\}^+$ of binary strings with an odd number of 1s is MSO-definable.

3.2 Finite automata

A language L is *regular* if $L = L(\mathbb{A})$ for some finite automaton $\mathbb{A}$ – we recall the definitions:

Definition 3.2.1. A *(nondeterministic) finite automaton* is a tuple $\mathbb{A} = (S, I, F, A, \Delta)$ where S is a set of *states*, $I \subseteq S$ a set of *initial states*, $F \subseteq S$ a set of *final states*, A is an alphabet, and $\Delta \subseteq S \times A \times S$ is the *transition relation.* $\mathbb{A}$ is *deterministic* if $|I| \leqslant 1$ and for all $(s,a) \in S \times A$ there is exactly one $s' \in S$ such that $(s,a,s') \in \Delta$, i.e., $s' = \Delta(s,a)$ and $\Delta : S \times A \to S$ is a function.

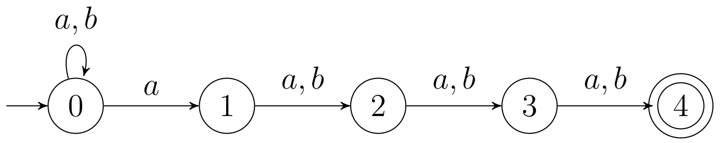
A *computation of* $\mathbb{A}$ *on* $w = a_1 \dots a_n \in A^n$ is a sequence $s_0, \dots, s_n$ of states such that $s_0 \in I$ and $(s_i, a_i, s_{i+1}) \in \ \Delta$ for all $i < n$. The computation is *from* s_0 and *to* s_n; it is *accepting* if $s_0 \in I$ and $s_n \in F$. If such a computation exists we say $\mathbb{A}$ *accepts* w.

The *language of* $\mathbb{A}$ is the set $L(\mathbb{A})$ of $w \in A^+$ accepted by $\mathbb{A}$. Two finite automata are *equivalent* if they have the same language.

Remark 3.2.2. If $\mathbb{A}$ is deterministic, then for every $w \in A^+$ there is exactly one computation of $\mathbb{A}$ on w.

Example 3.2.3. We depict a finite automaton $\mathbb{A}$ as usual, exemplified below. We have $S = \{0,1,2,3,4\}$, $S_0 = \{0\}$ indicated by the tail-less arrow, $F = \{4\}$ indicated by the double circle, $A = \{a,b\}$, and Δ is given by the A-labeled arrows; e.g., the loop arrow means $(0,a,0), (0,b,0) \in \Delta$ and the arrow from 0 to 1 means $(0,a,1) \in \Delta$. $\mathbb{A}$ is not deterministic because there are two a-labeled arrows leaving state 0.

The language $L(\mathbb{A})$ is the set of words $w \in A^+$ whose 4th letter from right is a.



Exercise 3.2.4. Draw an automaton whose language is Par, the set of binary strings with an odd number of 1s.

Lemma 3.2.5. *Let $k_1, k_2 \in \mathbb{N}$ and $\mathbb{A}_1, \mathbb{A}_2$ be finite automata with alphabet A and k_1, k_2 states. There are finite automata $\mathbb{A}_1 \otimes \mathbb{A}_2, \mathbb{A}_1 \oplus \mathbb{A}_2, \bar{\mathbb{A}}_1$ such that*

1. *$\mathbb{A}_1 \otimes \mathbb{A}_2$ has $k_1 \cdot k_2$ states and $L(\mathbb{A}_1 \otimes \mathbb{A}_2) = L(\mathbb{A}_1) \cap L(\mathbb{A}_2)$.*

2. *$\mathbb{A}_1 \oplus \mathbb{A}_2$ has $k_1 + k_2$ states and $L(\mathbb{A}_1 \oplus \mathbb{A}_2) = L(\mathbb{A}_1) \cup L(\mathbb{A}_2)$.*

3. *$\bar{\mathbb{A}}_1$ has 2^{k_1} states and $L(\bar{\mathbb{A}}_1) = A^+ \smallsetminus L(\mathbb{A}_1)$.*

Proof. Write $\mathbb{A}_1 = (S_1, I_1, F_1, A, \Delta_1)$ and $\mathbb{A}_2 = (S_2, I_2, F_2, A, \Delta_2)$. For 1, define $\mathbb{A}_1 \otimes \mathbb{A}_2 = (S_1 \times S_2, I_1 \times I_2, F_1 \times F_2, A, \Delta)$ where Δ contains $\big((s_1,s_2), a, (s_1', s_2')\big)$ if both $(s_1, a, s_1') \in \Delta_1$ and $(s_2, a, s_2') \in \Delta_2$. Then $(s_0^1, s_0^2), \dots, (s_n^1, s_n^2)$ is a computation of $\mathbb{A}_1 \otimes \mathbb{A}_2$ on $w \in A^n$ if and only if, for $i = 1, 2$, $s_0^i, \dots, s_n^i$ is a computation of $\mathbb{A}_i$ on w.

For 2, assume S_1, S_2 are disjoint and set $\mathbb{A}_1 \oplus \mathbb{A}_2 := (S_1 \cup S_2, I_0 \cup I_1, F_1 \cup F_2, A, \Delta_1 \cup \Delta_2)$.

For 3, the next lemma gives a deterministic finite automaton (S, I, F, A, Δ) with $|S| = 2^{k_1}$ equivalent to $\mathbb{A}_1$; set $\bar{\mathbb{A}} := (S, I, S \smallsetminus F, A, \Delta)$. $\square$

Lemma 3.2.6. *Let $k \in \mathbb{N}$. For every finite automaton with k states there is an equivalent deterministic finite automaton with 2^k states.*

Proof. Given $\mathbb{A} = (S, I, F, A, \Delta)$ define a deterministic $\mathbb{A}' := (S', I', F', A, \Delta')$ by $S' := P(S), I' := \{I\}, F' := \{X \subseteq S \mid X \cap F \neq \varnothing\}$ and $\Delta' : S' \times A \to S'$ by

$$\Delta'(X, a) := \{s' \in S \mid (s, a, s') \in \Delta, s \in X\}$$

for $X \in S', a \in A$. It is straightforward to check that for the computation $X_0, \ldots, X_n$ of $\mathbb{A}'$ on $w \in A^n$, the set $X_0 \times \cdots \times X_n$ is the set of computations of $\mathbb{A}$ on w. □

This exponential blow-up can, in general, not be avoided:

Proposition 3.2.7. *For every $k > 0$ there is a finite automaton with $k+1$ states such that every equivalent deterministic automaton has $\geqslant 2^k$ states.*

Proof. As in Example 3.2.3 one defines a finite automaton with $k+1$ states that accepts the words whose kth letter from the right is a. Let $\mathbb{A}$ be an equivalent deterministic automaton and assume it has $< 2^k$ states. Then there are distinct $w = a_1 \cdots a_k, w' = a'_1 \cdots a'_k \in \{a, b\}^k$ such that, if there exist computations of $\mathbb{A}$ on both w, w', then they end in the same state. Choose $i \in [k]$ minimal such that $a_i \neq a'_i$. Then $\mathbb{A}$ accepts $a_1 \cdots a_k b^i$ if and only if $\mathbb{A}$ accepts $a'_1 \cdots a'_k b^i$. But exactly one is in $L(\mathbb{A})$ – contradiction. □

Lemma 3.2.8 (Pumping). *Let A be an alphabet and $L \subseteq A^+$ be regular. There is $p \in \mathbb{N}$ such that every sufficiently long $w \in L$ equals uvw' for certain $u, v, w' \in A^*$ and $|uv| \leqslant p$ and $v \neq \epsilon$ and such that $uv^n w' \in L$ for all $n \in \mathbb{N}$.*

Proof. Choose a finite automaton $\mathbb{A}$ with $L = L(\mathbb{A})$. Let p be its number of states. Let $w = a_1 \cdots a_n \in L$ with $n \geqslant p$. Choose an accepting computation $s_0, \ldots, s_n$ of $\mathbb{A}$ on w. Choose $i < j \leqslant p$ such that $s_i = s_j$. Set $u := a_1 \cdots a_i$ (empty if $i = 0$) and $v := a_i \cdots a_j$ (not empty) and $w' := a_{j+1} \cdots a_n$ (empty if $j = n$). □

Exercise 3.2.9. Let A, A' be alphabets.

1. Assume there is a bijection $\alpha : A \to A'$; for every finite automaton $\mathbb{A}$ with alphabet A there is a finite automaton $\mathbb{A}'$ with alphabet A' such that

$$L(\mathbb{A}') = \bigcup_{n>0} \left\{\alpha(a_1) \cdots \alpha(a_n) \in (A')^+ \mid a_1 \cdots a_n \in L(\mathbb{A})\right\}.$$

2. For every finite automaton $\mathbb{A}'$ with alphabet $A \times A'$ there is a finite automaton $\mathbb{A}$ with alphabet A such that

$$L(\mathbb{A}) = \bigcup_{n>0} \left\{a_1 \cdots a_n \in A^+ \mid (a_1, a'_1) \cdots (a_n, a'_n) \in L(\mathbb{A}') \text{ for some } a'_1 \cdots a'_n \in (A')^+\right\}.$$

3. For every finite automaton $\mathbb{A}$ with alphabet A there is a finite automaton $\mathbb{A}'$ with alphabet $A \times A'$ such that

$$L(\mathbb{A}') = \bigcup_{n>0} \left\{(a_1, a'_1) \cdots (a_n, a'_n) \in (A \times A')^+ \mid a_1 \cdots a_n \in L(\mathbb{A})\right\}.$$

Exercise 3.2.10. There is an efficient algorithm deciding

NFA EMPTINESS	
Input:	a finite automaton $\mathbb{A}$.
Problem:	$L(\mathbb{A}) = \varnothing$?

3.3 Büchi's theorem for words

Fix an alphabet A. This section aims to prove:

Theorem 3.3.1 (Büchi – finite)**.** *A language L is regular if and only if it is MSO-definable.*

For the proof we need to extend Definition 3.1.6 to MSO L_A-formulas, say $\varphi(X_1, X_2)$ with free set variables X_1, X_2. The idea is to use alphabet $A \times \{0,1\}^2$; e.g. a length 6 word w looks as follows, for $a_1, \dots, a_6 \in A$ and writing letters as columns,

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

The binary rows code two sets, namely the first $B_1 := \{2,4\}$ and the second $B_2 := \{1,4,5\}$. Per definition, w satisfies $\varphi(X_1, X_2)$ if and only if $\mathfrak{S}(a_1 \cdots a_6) \models \varphi[B_1, B_2]$.

Definition 3.3.2. Let $\varphi(\bar{X})$ with $\bar{X} = X_1 \cdots X_k$ be an MSO L_A-formula. Let

$$w = (a_1, b_{11} \cdots b_{1k}) \cdots (a_n, b_{n1} \cdots b_{nk})$$

be a word of length $n > 0$ over the alphabet $(A \times \{0,1\}^k)$; here, $a_i \in A$ and $b_{ij} \in \{0,1\}$. We say w *satisfies* φ if $\mathfrak{S}(w) \models \varphi[B_1, \dots, B_k]$ where $B_j := \{i \in [n] \mid b_{ij} = 1\}$ for all $j \in [k]$.

Lemma 3.3.3. *Let $\varphi(\bar{X})$ with $\bar{X} = X_1 \cdots X_k$ be an MSO L_A-formula, $i \in [k]$, and $\mathbb{A}$ a finite automaton with $L(\varphi(\bar{X})) = L(\mathbb{A})$. Then there is a finite automaton $\mathbb{A}'$ with*

$$L(\exists X_i \varphi(\bar{X})) = L(\mathbb{A}').$$

Proof. Let $\alpha : A \times \{0,1\}^k \to (A \times \{0,1\}^{k-1}) \times \{0,1\}$ map $(a, b_1 \cdots b_k)$ to $((a, b_1 \cdots b_{i-1} b_{i+1} \cdots b_{k-1}), b_i)$. For $\mathbb{A}$ choose $\mathbb{A}''$ according to Exercise 3.2.9 (1). For $\mathbb{A}''$ choose $\mathbb{A}'$ according to Exercise 3.2.9 (2) plugging $(A \times \{0,1\}^{k-1})$ for A and $\{0,1\}$ for A'. Then

$$\begin{aligned}
&(a_1, b_{11} \cdots b_{1(k-1)}) \cdots (a_n, b_{n1} \cdots b_{n(k-1)}) \in L(\mathbb{A}') \\
\iff &((a_1, b_{11} \cdots b_{1(k-1)}), c_1) \cdots ((a_n, b_{n1} \cdots b_{n(k-1)}), c_n) \in L(\mathbb{A}'') \text{ for some bits } c_j \\
\iff &(a_1, b_{11} \cdots b_{1(i-1)} c_1 b_{1i} \cdots b_{1(k-1)}) \cdots (a_n, b_{n1} \cdots b_{n(i-1)} c_n b_{ni} \cdots b_{n(k-1)}) \in L(\mathbb{A}) \text{ for some bits } c_j \\
\iff &\mathfrak{S}(a_1 \cdots a_n) \models \varphi[B_1, \dots, B_{i-1}, C, B_i, \dots, B_{k-1}] \text{ for some } C \subseteq [n],
\end{aligned}$$

where $B_i = \{j \in [n] \mid b_{ij} = 1\}$; for the last equivalence, $C = \{j \in [n] \mid c_j = 1\}$. □

Proof of Theorem 3.3.1. $\Rightarrow$: given a finite automaton $\mathbb{A} = (S, I, F, A, \Delta)$ we want an MSO L_A-sentence $\varphi_{\mathbb{A}}$ such that for all $w \in A^+$ we have $\mathfrak{S}(w) \models \varphi_{\mathbb{A}}$ if and only if $\mathbb{A}$ accepts w. Intuitively, the sentence $\varphi_{\mathbb{A}}$, evaluated in $\mathfrak{S}(w)$, expresses that $\mathbb{A}$ accepts w. We can assume $S = [k]$ for some $k \in \mathbb{N}$ and use set variables X_i; intuitively, $X_i x$ means $\mathbb{A}$ is in state i at step x. We define

$$\varphi_{\mathbb{A}} := \exists X_1 \cdots X_k \big(\mathit{Part} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{Acc} \big)$$

where *Part* states the X_i form a partition, *Start* states the first state is reached from an initial state, *Step* states the states accord to Δ, and *Acc* states the last state is final:

$$\begin{aligned}
\mathit{Part} &:= \forall x \Big(\bigvee_{i \in [k]} X x \wedge \bigwedge_{1 \leqslant i < j \leqslant k} (\neg X_i x \vee \neg X_j x) \Big); \\
\mathit{Start} &:= \forall x \Big(\forall z \neg x < z \to \bigvee_{(i,a,j) \in \Delta, i \in I} (P_a x \wedge X_j x) \Big); \\
\mathit{Step} &:= \forall xy \Big((x < y \wedge \forall z \neg (x < z \wedge z < y)) \to \bigvee_{(i,a,j) \in \Delta} \big(X_i x \wedge P_a y \wedge X_j y \big); \\
\mathit{Acc} &:= \forall x \Big(\forall z \neg x < z \to \bigvee_{i \in F} X_i x \Big).
\end{aligned}$$

$\Leftarrow$: we first write MSO L_A-formulas in *translatable* form: formulas built by means of $\vee, \neg$ and $\exists X$ from the the following formulas for any $X, Y \in \mathit{VAR}$:

$$\begin{aligned}
\mathit{Sing}(X) &:= \exists x (X x \wedge \forall y (X y \to x \doteq y)); \\
\mathit{Before}(X, Y) &:= \forall xy ((X x \wedge Y y) \to x < y); \\
\mathit{Letter}_a(X) &:= \forall x (X x \to P_a x).
\end{aligned}$$

For individual variables $x_1, x_2, \ldots \in \mathit{Var}$ reserve set variables $Y_1, Y_2, \ldots \in \mathit{VAR}$.

Claim: for every MSO L_A-formula $\varphi(X_1 \cdots X_\ell, x_1 \cdots x_k)$ there is a translatable MSO L_A-formula $\varphi^*(X_1 \cdots X_k, Y_1 \cdots Y_k)$ such that for all words $w \in A^+$ and all $\bar{B} = B_1 \cdots B_\ell \in P([n])^\ell$ and all $\bar{\imath} = i_1 \cdots i_k \in [n]^k$:

$$\mathfrak{S}(w) \models \varphi[\bar{B}, \bar{\imath}] \iff \mathfrak{S}(w) \models \varphi^*[\bar{B}, \{i_1\}, \ldots, \{i_k\}].$$

Proof of the claim. We define $\varphi \mapsto \varphi^*$ by a straightforward recursion: $(x_i < x_j)^* := \mathit{Before}(Y_i, Y_j)$, $(P_a x_i)^* := \mathit{Letter}_a(Y_i)$, $((\varphi \vee \psi))^* := (\varphi^* \vee \psi^*)$, $(\neg\varphi)^* := \neg\varphi^*$, $(\exists x_i \varphi)^* := \exists Y_i (\mathit{Sing}(Y_i) \wedge \varphi^*)$, $(\exists X \varphi)^* := \exists X \varphi^*$. $\dashv$

It thus suffices to find, given a translatable MSO L_A-formula $\varphi(\bar{X})$ where all set variables occurring in φ (free or bound) are among $\bar{X} := X_1 \cdots X_k$, a finite automaton $\mathbb{A}_{\varphi(\bar{X})}$ with $L(\mathbb{A}_{\varphi(\bar{X})}) = L(\varphi(\bar{X}))$. Indeed: if φ is a translatable MSO L_A-sentence, then Exercise 3.2.9 (2) applied on $\mathbb{A}_{\varphi(\bar{X})}$ (with $A' := \{0,1\}^k$) yields $\mathbb{A}_\varphi$ with $L(\varphi) = L(\mathbb{A}_\varphi)$.

We construct $\mathbb{A}_{\varphi(\bar{X})}$ by recursion on φ. We leave it to the reader to construct $\mathbb{A}_{\varphi(\bar{X})}$ for $\varphi(\bar{X})$ one of $\mathit{Sing}(X_i), \mathit{Before}(X_i, X_j), \mathit{Letter}_a(X_i)$.

Using Lemma 3.2.5, we set $\mathbb{A}_{\neg\varphi(\bar{X})} := \bar{\mathbb{A}}_{\varphi(\bar{X})}$ and $\mathbb{A}_{(\varphi(\bar{X}) \wedge \psi(\bar{X}))} := \mathbb{A}_{\varphi(\bar{X})} \otimes \mathbb{A}_{\psi(\bar{X})}$.

To construct $\mathbb{A}_{\exists X_i \varphi(\bar{X})}$, Lemma 3.3.3 gives $\mathbb{B}$ with $L(\mathbb{B}) = L(\exists X_i \varphi(\bar{X}))$. Its alphabet is $A \times \{0,1\}^{k-1}$. Exercise 3.2.9 (3) gives $\mathbb{A}$ with alphabet $A \times \{0,1\}^k$, intuitively, padding with arbitrary k-th bits. Use Exercise 3.2.9 (1) with α that swaps this padding to place i, namely, α maps $(a, b_1 \cdots b_k)$ to $(a, b_1 \cdots b_{i-1} b_k b_i \cdots b_{k-1})$. □

Exercise 3.3.4. Show that the set of binary palindromes is not regular. Infer that there does not exist an MSO $L_{\{0,1\}}$-formula $\varphi(x,y,z)$ such that for all $w \in \{0,1\}^+$ and $i,j,k \in [|w|]$:

$$\mathfrak{S}(w) \models \varphi[i,j,k] \iff i + j = k.$$

3.3.1 Corollaries

The proof of Büchi's theorem actually establishes the following effective version.

Corollary 3.3.5.

1. *There is an algorithm that given an MSO L_A-sentence φ outputs a finite automaton $\mathbb{A}_\varphi$ such that $L(\varphi) = L(\mathbb{A}_\varphi)$.*
2. *There is an algorithm that, given finite automaton $\mathbb{A}$, outputs an MSO L_A-sentence $\varphi_{\mathbb{A}}$ such that $L(\mathbb{A}) = L(\varphi_{\mathbb{A}})$.*

Mapping ψ to $\varphi_{\mathbb{A}_\psi}$ we see:

Corollary 3.3.6. *There is an algorithm that, given an MSO L_A-sentence ψ, outputs an MSO L_A-sentence of the form $\exists \bar{X} \chi$ where no set quantifiers occur in χ and such that*

$$L(\psi) = L(\exists \bar{X} \chi).$$

Corollary 3.3.7. *There is a computable function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides*

MC(MSO,A^+)	
Input:	an MSO L_A-sentence φ and a word $w \in A^+$.
Problem:	$\mathfrak{S}(w) \models \varphi$?

in time polynomial in $f(|\varphi|) \cdot |w|$.

Proof. Given φ, w, compute a deterministic finite automaton $\mathbb{A}$ equivalent to $\mathbb{A}_\varphi$ (see Lemma 3.2.6) and check whether it accepts w.

Choose a computable function f' such that the computation of $\mathbb{A}$ takes $f'(|\varphi|)$ many steps. The check takes $|w|$ many evaluations of the transition function of $\mathbb{A}$, each done by scanning $\mathbb{A}$, so is efficient given $\mathbb{A}$. □

Exercise 3.3.8. There are algorithms that decide whether a given MSO L_A-sentence is true in some (resp. all) word structures.

Remark 3.3.9. In the beginning of this chapter, we announced to aim for runtimes as in Corollary 3.3.7. But how fast does f grow? From our proof, $f(|\varphi|)$ is at least the number of states of $\mathbb{A}_\varphi$ and this is not elementary, that is, not bounded by $2^{\cdot^{\cdot^{2^{|\varphi|}}}}$ for any constant height tower of 2s: our construction has an exponential blow-up for every $\neg$ in φ. Frick and Grohe proved in 2004 that Corollary 3.3.7 fails for elementary f (assuming P $\neq$ NP), even when we restrict to first-order φ (assuming a certain hypothesis from parameterized complexity theory, namely FPT $\neq$ AW$[*]$). This is outside the scope of this course.

3.4 Büchi's theorem for ω-words

Fix an alphabet A.

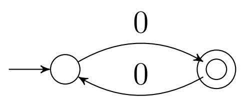
Definition 3.4.1. A^ω is the set of *ω-words (over A)*, namely functions $w : \mathbb{N}_{>0} \to A$. A *Büchi automaton* is a finite automaton $\mathbb{A} = (S, I, F, A, \Delta)$. A *computation* of $\mathbb{A}$ on $w \in A^\omega$ is a function $s : \mathbb{N} \to S$ such that $s(0) \in I$ and $(s(i), w(i+1), s(i+1)) \in \Delta$ for all $i \in \mathbb{N}$; it is *accepting* if $\{i \in \mathbb{N} \mid s(i) \in F\}$ is infinite. If such a computation exists, $\mathbb{A}$ *accepts* w.

An ω-language is a subset of A^ω. The *ω-language of* $\mathbb{A}$ is the set $L_\omega(\mathbb{A})$ of ω-words accepted by $\mathbb{A}$. $L \subseteq A^\omega$ is *ω-regular* if $L = L_\omega(\mathbb{A})$ for some Büchi automaton $\mathbb{A}$. Two Büchi automata are *equivalent* if they have the same ω-language.

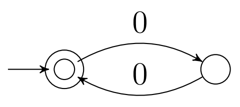
We often write w, s as sequences $a_1\ a_2 \cdots$ resp. $s_0\ s_1\ s_2 \cdots$ of letters resp. states, understanding $w(i) = a_i, s(i) = s_i$.

Example 3.4.2. Let $A := \{0, 1\}$. $\mathbb{A}_1, \mathbb{A}_2$ are equivalent as Büchi automata but not as finite automata.

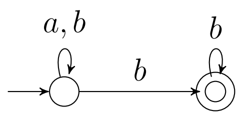
$\mathbb{A}_1$



$\mathbb{A}_2$



$\mathbb{A}_3$



$L(\mathbb{A}_1) = \{0^{2n+1} \mid n \in \mathbb{N}\}$
$L_\omega(\mathbb{A}_1) = \{000\cdots\}$

$L(\mathbb{A}_2) = \{0^{2n} \mid n \in \mathbb{N}_{>0}\}$
$L_\omega(\mathbb{A}_2) = \{000\cdots\}$

$L(\mathbb{A}_3) = \{wb \mid w \in A^*\}$
$L_\omega(\mathbb{A}_3) = \{wbbb\cdots \in A^\omega \mid w \in A^*\}$

Exercise 3.4.3. Find equivalent finite automata, not equivalent as Büchi automata.

Determinization (Lemma 3.2.6) fails:

Proposition 3.4.4. *$\mathbb{A}_3$ is not equivalent to any deterministic Büchi aiutomaton.*

Proof. Assume $\mathbb{A}$ is deterministic and equivalent to $\mathbb{A}_3$. Then its accepting run on $bbb\cdots$ is in a final state, say at step n_1, so after reading b^{n_1}. Being deterministic $\mathbb{A}$ visits the same state in its accepting run on $b^{n_1}abbb\cdots$. Choose n_2 such that this run is in a final state after reading $b^{n_1}ab^{n_2}$. Continuing like this gives an accepting run on a word with infinitely many as – contradiction. $\square$

Exercise 3.4.5. For Büchi automata $\mathbb{A}, \mathbb{B}$ over the same alphabet there is a *product automaton* $\mathbb{A} \otimes \mathbb{B}$ such that $L_\omega(\mathbb{A} \otimes \mathbb{B}) = L_\omega(\mathbb{A}) \cap L_\omega(\mathbb{B})$.

Closure under complementation requires a new proof outside the scope of this course:

Theorem 3.4.6 (McNaughton, Safra). *There is an algorithm that, given a Büchi automaton $\mathbb{A}$, outputs a Büchi automaton $\bar{\mathbb{A}}$ such that $L_\omega(\bar{\mathbb{A}}) = A^\omega \setminus L_\omega(\mathbb{A})$.*

Exercise 3.4.7. A *generalized Büchi automaton* $\mathbb{A} = (S, I, \mathcal{F}, A, \Delta)$ is defined like a Büchi automaton but $\mathcal{F} \subseteq P(S)$ (instead of $F \in P(S)$). A computation (defined a before) s of $\mathbb{A}$ on $w \in A^\omega$ is *accepting* if $\{i \mid s(i) \in F\}$ is infinite for all $F \in \mathcal{F}$; $\mathbb{A}$ *accepts* w if such a computation exists. Show there is an efficient algorithm that, given such $\mathbb{A}$, outputs a Büchi automaton $\mathbb{B}$ that accepts the same ω-words.

Definition 3.4.8. Let $w \in A^\omega$. The *word structure* $\mathfrak{S}(w)$ is the L_A-structure with universe $\mathbb{N}_{>0}$, $<^{\mathfrak{S}(w)}$ the natural order, and $P_a^{\mathfrak{S}(w)} = \{i \in \mathbb{N}_{>0} \mid w(i) = a\}$ for $a \in A$. An ω-language $L \subseteq A^\omega$ is *MSO-definable* if there is an MSO L_A-sentence φ such that

$$L = L_\omega(\varphi) := \{w \in A^\omega \mid \mathfrak{S}(w) \models \varphi\}.$$

Theorem 3.4.9 (Büchi – infinite). *An ω-language is ω-regular if and only if it is MSO-definable.*

Proof. $\Rightarrow$: given a Büchi automaton $\mathbb{A}$ we define $\varphi_\mathbb{A}$ as before but with

$$Acc := \forall x \exists y (x < y \wedge \textstyle\bigvee_{i \in F} X_i y).$$

$\Leftarrow$ has the same proof as before – this also holds for Lemmas 3.2.5 (2) and 3.3.3 and Exercise 3.2.9 used therein. □

Remark 3.4.10. It should be clear that the L_ω-analogues of Corollaries 3.3.5, 3.3.6 follow. We shall need in particular that there exists an algorithm that given an MSO L_A-sentence φ outputs a Büchi automaton $\mathbb{A}_\varphi$ such that $L_\omega(\varphi) = L_\omega(\mathbb{A}_\varphi)$.

Exercise 3.4.11. There is an efficient algorithm deciding

NBA EMPTINESS	
Input:	a Büchi automaton $\mathbb{A}$.
Problem:	$L_\omega(\mathbb{A}) = \emptyset$?

Infer that there exists an algorithm that, given MSO L_A-sentences φ, ψ, decides whether $L_\omega(\varphi) = L_\omega(\psi)$, i.e., φ and ψ are equivalent over ω-words.

3.5 Transition systems and linear time properties

An ATM is a transition system that switches from state to state depending on actions taken by the user. This can be seen as a deterministic automaton with letters representing user actions. Once designed the system should satisfy certain correctness specifications, e.g., "whenever the user inserts a card, there is a later state where the card is output". The formal verification perspective on model-checking aims at as efficient as possible algorithms to check as many as possible correctness specifications. Formally, the specifications are written in logics tailored for such specifications.

Algorithms are needed because, in practice, huge transition systems arise naturally and it is a superhuman task to check correctness specifications. E.g., a sensor might be modeled by a small automaton where the environment takes actions to change states, and a robot might switch states according to switches of states of many sensors. This can be modeled by taking $\otimes$ of many small automata and the result can be huge. Or, imagine concurrent processes modifying the values of some common variables, the states are given by all or all reasonable values of the variables; e.g. various secretariats enter student grades into a university database, or users of a webpage enter text content in chat room. This typically gives rise to very large, possibly infinite transition systems.

Definition 3.5.1. Let A be an alphabet. A *(finite) transition system over* A is a tuple $T = (\mathfrak{G}, \ell)$ wher $\mathfrak{G} = (G, E^{\mathfrak{G}})$ is a (finite) directed graph and $\ell : G \to A$ is a *labeling*. An *execution* of T is an infinite sequence $g_1, g_2 \ldots$ of vertices such that $(g_i, g_{i+1}) \in E^{\mathfrak{G}}$ for all $i \in \mathbb{N}$; its *trace* t is the ω-word $\ell(g_1)\ \ell(g_2)\ \cdots \in A^\omega$, i.e., $t : \mathbb{N}_{>0} \to A$ with $t(i) := \ell(g_i)$. A *partial trace* is a finite *prefix* of a trace, i.e., $t{\upharpoonright}[n]$ for some trace t and some $n \in \mathbb{N}$, i.e., $t(1)\cdots t(n) \in A^n$. A (partial) trace *of* T is a (partial) trace of an execution of T.

Definition 3.5.2. Let A be an alphabet and $P \subseteq A^\omega$. A transition system T *satisfies* P if every trace of T is in P. The *closure of* P is

$$\mathrm{cl}(P) := \big\{ t \in A^\omega \mid \text{ every prefix of } t \text{ is a prefix of some } s \in P \big\}.$$

P is a *safety property* if $\mathrm{cl}(P) = P$, and a *liveness property* if $\mathrm{cl}(P) = A^\omega$.

Remark 3.5.3.

1. For $P, Q \subseteq A^\omega$ one easily checks

$$P \subseteq \mathrm{cl}(P),\ \mathrm{cl}(\mathrm{cl}(P)) = \mathrm{cl}(P),\ \mathrm{cl}(P \cup Q) = \mathrm{cl}(P) \cup \mathrm{cl}(Q).$$

   In fact, cl is topological closure in a natural topology on A^ω (which one?).

2. A safety property P states "something bad never happens": $t \in P$ if and only if every $t \notin P$ has a prefix $w \in A^*$ that is P-*bad*, i.e., no $s \in P$ has prefix w.

3. A liveness property P states "something good will happen": every $w \in A^*$ is a prefix of some $t \in P$.

4. Every $P \subseteq A^\omega$ is the intersection of a safety and a liveness property.

   Indeed, $P = \mathrm{cl}(P) \cap (P \cup (A^\omega \setminus \mathrm{cl}(P)))$; note

$$\mathrm{cl}(P \cup (A^\omega \setminus \mathrm{cl}(P))) = \mathrm{cl}(P) \cup \mathrm{cl}(A^\omega \setminus \mathrm{cl}(P)) = A^\omega.$$

Proposition 3.5.4. *Let T, T' be finite transition systems over an alphabet A. Then T and T' satisfy the same safety properties if and only if T and T' have the same traces.*

Proof. $\Leftarrow$ is trivial. $\Rightarrow$: first note that T, T' have the same partial traces. Indeed, assume $w \in A^*$ is a partial trace of T but not of T'; then consider the safety property P of ω-words that do not have w as a prefix.

We show every trace t' of T' is a trace of $T = (\mathfrak{G}, \ell)$ (the converse is analogous). Consider all finite sequences $g_1, \dots, g_n$ in $\mathfrak{G}$ where $n \in \mathbb{N}$ and $(g_i, g_{i+1}) \in E^{\mathfrak{G}}$ for all $i \in [n-1]$ and $\ell(g_1)\cdots\ell(g_n) = t' \restriction [n]$. By the above such sequences exist for all n. Hence the set of these sequences is an infinite tree over G. By Exercise 1.4.10, it contains an infinite branch $g_1, g_2, \dots$. This is an execution of T with trace t'. □

3.5.1 Model-checking MSO

Below, let $|T|$ be the length of the binary string encoding a (finite) transition system T – in some reasonable sense, the details are irrelevant for us.

Theorem 3.5.5. *There are a computable function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides*

MC(MSO,TS)	
Input:	A transition system T over an alphabet A and an L_A-sentence φ.
Problem:	does T satisfy $L_\omega(\varphi)$?

in time polynomial in $f(|\varphi|) \cdot |T|$.

Proof. Given $T = (\mathfrak{G}, \ell)$, a (finite) transition system over A, we first compute the *trace automaton* $\mathbb{A}_T := (G, G, G, A, \Delta)$ where Δ contains (g, a, g') if $\ell(g) = a$ and $(g, g') \in E^{\mathfrak{G}}$. Then $L_\omega(\mathbb{A}_T)$ is the the set of traces of T. Then compute $\mathbb{A}_{\neg\varphi}$ from Remark 3.4.10, and then the product automaton $\mathbb{A}_T \otimes \mathbb{A}_{\neg\varphi}$ from Exercise 3.4.5. Note, T satisfies $L_\omega(\varphi)$ if and only if $L_\omega(\mathbb{A}_T \otimes \mathbb{A}_{\neg\varphi}) = \emptyset$. Now use Exercise 3.4.11. □

Recalling Remark 3.3.9, f grows very fast and it is dubious whether the above algorithm should be considered feasible. This motivates the search for other, possibly less expressive logics with a faster model-checker.

3.6 Linear time temporal logic

Definition 3.6.1. Let $k \in \mathbb{N}$ and $\bar{X} = X_1 \cdots X_k$ a tuple of propositional variables. A *transition system over $\bar{X}$* is a transition system over $A(\bar{X})$ where $A(\bar{X})$ is the set of assignments to $\bar{X}$.

In the literature such transition systems are often called *Kripke structures*. Linear time temporal logic (LTL) is a logic intended to express linear time properties of them. It extends the Definitions 1.1.2 and 1.2.2 of syntax and semantics of propositional logic.

Definition 3.6.2 (Syntax and semantics of LTL)**.** Add to the alphabet of propositional logic the letters X and U. The set of *linear time temporal logic (LTL) formulas* is the smallest set F of words that contains all propositional variables $X_0, X_1, \dots$ and such that for all $\varphi, \psi \in F$:

$$\neg\varphi \in F,\ (\varphi \wedge \psi) \in F,\ \mathsf{X}\varphi \in F,\ (\varphi \mathsf{U} \psi) \in F.$$

Writing $\varphi = \varphi(\bar{X})$ means all variables occurring in φ are among $\bar{X}$. Let $t \in A(\bar{X})^\omega$, i.e., $t(i)$ is an assignment to $\bar{X}$ for all $i > 0$. We define $\varphi(\bar{X})$ *is true in* t *at time* i, symbolically $t, i \vDash \varphi$, by recursion on φ, stipulating for all LTL formulas ψ, χ and variables X from $\bar{X}$:

(T1) $t, i \vDash X \iff t(i)(X) = 1$;

(T2) $t, i \vDash \neg\psi \iff t, i \nvDash \psi$;

(T3) $t, i \vDash (\psi \wedge \chi) \iff t, i \vDash \psi$ and $t, i \vDash \chi$;

(T4) $t, i \vDash \mathsf{X}\psi \iff t, i + 1 \vDash \psi$;

(T5) $t, i \vDash (\psi \mathsf{U} \chi) \iff$ there is $j \geqslant i$ such that for all $i \leqslant k < j$: $t, j \vDash \chi$ and $t, k \vDash \psi$;

Remark 3.6.3. $\mathsf{X}\varphi$ is read "next φ", and $(\varphi \mathsf{U} \psi)$ is read "φ until ψ". As usual, the definition by recursion on syntax is justified by a straightforward lemma on unique readability. Also the set $sub(\varphi)$ of *subformulas* of a formula φ is defined by recursion by adding, in Example 1.1.8, the conditions: $sub(\mathsf{X}\varphi) := \{\mathsf{X}\varphi\} \cup sub(\varphi)$ and $sub((\varphi \mathsf{U} \psi)) := \{(\varphi \mathsf{U} \psi)\} \cup sub(\varphi) \cup sub(\psi)$.

Exercise 3.6.4 (Coincidence lemma)**.** Let $\varphi(\bar{X})$ be an LTL formula and assume $t, t' \in A(\bar{X})^\omega$ are such that, for all $i \in \mathbb{N}_{>0}$, the assignments $t(i), t'(i)$ agree on all variables occurring in φ (a subset of $\bar{X}$). Then for all $i \in \mathbb{N}_{>0}$: $t, i \vDash \varphi \iff t', i \vDash \varphi$.

Remark 3.6.5. let X be a variable and write $\top := (X \vee \neg X)$. $\mathsf{F}\varphi := (\top \mathsf{U} \varphi)$ reads "finally φ" and $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$ reads "globally φ". Then

- $t, i \vDash \mathsf{F}\varphi \iff$ there is $j \geqslant i$: $t, j \vDash \varphi$.
- $t, i \vDash \mathsf{G}\varphi \iff$ for all $j \geqslant i$: $t, j \vDash \varphi$.

Example 3.6.6. Consider a transition system T containing a traffic light, which lights red, yellow or green in states. Formally, T is over variables $\bar{X}$ including R, Y, G. Then "once red the light turns eventually green" is formalized by $\mathsf{G}(R \to \mathsf{F}G)$. Or, "once red, the light turns eventually green after being yellow for some time" is formalized by

$$\mathsf{G}(R \to (R \mathsf{U}(Y \wedge \mathsf{X}(Y \mathsf{U} G)))).$$

Definition 3.6.7. The *ω-language defined by* an LTL formula $\varphi = \varphi(\bar{X})$ is

$$L_\omega(\varphi) := \{t \in A(\bar{X})^\omega \mid t, 1 \models \varphi\}.$$

LTL-formulas $\varphi(\bar{X}), \psi(\bar{X})$ are *equivalent*, symbolically $\varphi \equiv \psi$, if $L_\omega(\varphi) = L_\omega(\psi)$.

Exercise 3.6.8. For all LTL-formulas φ, ψ, χ:

Dualities: $\neg \mathsf{G}\varphi \equiv \mathsf{F}\neg\varphi$, $\neg\mathsf{X}\varphi \equiv \mathsf{X}\neg\varphi$

Idempotencies: $\mathsf{GG}\varphi \equiv \mathsf{G}\varphi$, $(\varphi\mathsf{U}(\varphi\mathsf{U}\psi)) \equiv (\varphi\mathsf{U}\psi)$

Absorption laws: $\mathsf{FGF}\varphi \equiv \mathsf{GF}\varphi$, $\mathsf{GFG}\varphi \equiv \mathsf{FG}\varphi$

Distribution laws: $\mathsf{G}(\varphi\wedge\psi) \equiv (\mathsf{G}\varphi\wedge\mathsf{G}\psi)$, $\mathsf{F}(\varphi\vee\psi) \equiv (\mathsf{F}\varphi\vee\mathsf{F}\psi)$, $\mathsf{X}(\varphi\mathsf{U}\psi) \equiv (\mathsf{X}\varphi\ \mathsf{U}\ \mathsf{X}\psi)$.

Expansion law: $(\varphi\mathsf{U}\psi) \equiv (\psi \vee (\varphi \wedge \mathsf{X}(\varphi\mathsf{U}\psi))$

We now observe that LTL is subsumed in first-order logic. *Kamp's theorem* states a certain converse. This is outside the scope of this course.

Proposition 3.6.9. *For every LTL formula $\varphi(\bar{X})$ there is a first-order $L_{A(\bar{X})}$-formula $\varphi^*(x)$ such that for all $t \in A(\bar{X})^\omega$ and all $i \in \mathbb{N}_{>0}$:*

$$t, i \models \varphi \iff \mathfrak{S}(t) \models \varphi^*[i].$$

Proof. We define $\varphi^*(x)$ by recursion on φ: for X a variable from $\bar{X}$,

$$X^* := \bigvee_{\beta\in A(\bar{X}),\beta(X)=1} P_\beta x,\ (\neg\varphi)^* := \neg\varphi^*(x),\ (\varphi\wedge\psi)^* := (\varphi^*(x)\wedge\psi^*(x))$$

and, writing $x \leqslant y$ for $(x < y \vee x \doteq y)$,

$$\begin{array}{rcl} (\mathsf{X}\varphi)^* & := & \exists y(x < y \wedge \forall z \neg(x < z \wedge z < y) \wedge \varphi^*(y)), \\ (\varphi\mathsf{U}\psi)^* & := & \exists y(x \leqslant y \wedge \psi^*(y) \wedge \forall z(x \leqslant z \wedge z < y) \to \varphi^*(z)). \end{array}$$

□

3.6.1 Model-checking LTL

Theorem 3.6.10 (Vardi, Wolper)**.** *There is an algorithm that maps an LTL formula φ to a Büchi automaton $\mathbb{A}_\varphi$ with $\leqslant 2^{|\varphi|}$ states such that $L_\omega(\varphi) = L_\omega(\mathbb{A}_\varphi)$.*

Proof. Let $\bar{X}$ list the variables in φ and $\Phi := sub(\varphi)$. A *type* is a set $s \subseteq \Phi$ such that

(t1) $(\psi_0 \wedge \psi_1) \in s \iff \psi_0 \in s$ and $\psi_1 \in s$;

(t2) $\neg\psi \iff \psi \notin s$;

(t3) $\psi_1 \in s \implies (\psi_0\mathsf{U}\psi_1) \in s$;

(t4) $(\psi_0\mathsf{U}\psi_1) \in s \implies \psi_0 \in s$ or $\psi_1 \in s$.

hold *for formulas in* Φ; e.g. for (t1) this means that the equivalence holds *if* $(\psi_0 \wedge \psi_1) \in \Phi$.

Let S denote the set of types. E.g., given $t \in A(\bar{X})^\omega$ and $i > 0$, we have the type

$$\Phi_{t,i} := \{\psi \in \Phi \mid t, i \models \psi\}.$$

Claim: There is a generalized Büchi automaton $\mathbb{A} = (S, S, \mathcal{F}, A(\bar{X}), \Delta)$ with $|\mathcal{F}| \leqslant |S|$ that on $t \in A(\bar{X})^\omega$ has exactly one accepting computation, namely $\Phi_{t,1}, \Phi_{t,2}, \cdots$.

This suffices: by Exercise 3.4.7, for $\mathbb{A}_\varphi$ we can take a Büchi automaton equivalent to the generalized Büchi automaton $(S, I, \mathcal{F}, A(\bar{X}), \Delta)$ where $I := \{s \in S \mid \varphi \in s\}$. Its number of states is polynomial in $|S| \leqslant 2^{|\varphi|}$. We are left to prove the claim.

We define Δ to contain (s, β, s') if and only if the following hold for formulas in Φ:

(Δ1) a variable X from $\bar{X}$ is in s if and only if $\beta(X) = 1$;

(Δ2) $\mathsf{X}\psi \in s \iff \psi \in s'$;

(Δ3) $(\psi_0 \mathsf{U} \psi_1) \in s \iff \psi_1 \in s$ or, both $\psi_0 \in s$ and $(\psi_0 \mathsf{U} \psi_1) \in s'$.

The idea is as follows: computing a state s containing $(\psi_0 \mathsf{U} \psi_1)$ is a commitment to make ψ_1 true eventually. This means to either make ψ_1 true in the next step s' or to delay this; when delaying, ψ_0 has to be true in s' and the commitment maintained. Delaying forever is not allowed – we demand the automaton to eventually reaching a state where $\psi_1$is true.

More precisely, we define $\mathcal{F}$ to contain

$$F_\psi := \left\{s \in S \mid (\psi_0 \mathsf{U} \psi_1) \notin s \text{ or } \psi_1 \in s\right\}.$$

for every $\psi := (\psi_0 \mathsf{U} \psi_1) \in \Phi$. It is clear that $\Phi_{t,1}, \Phi_{t,2}, \ldots$ is an accepting computation. Conversely, let $s_1, s_2, \ldots$ be an accepting computation. We show for all $\psi \in \Phi$ and $i \in \mathbb{N}_{>0}$:

$$\psi \in s_i \iff t, i \models \psi.$$

We proceed by induction on ψ and treat the case $\psi = (\psi_0 \mathsf{U} \psi_1)$. Assume $i = 1$, for simplicity.

Assume $t, 1 \models \psi$. Choose $i \geqslant 1$ such that $t, i \models \psi_1$ and $t, j \models \psi_0$ for all $1 \leqslant j < i$. By induction, $\psi_1 \in s_i$ and $\psi_0 \in s_j$. By (t3), $\psi \in s_i$. By $\psi_0 \in s_{i-1}$ and (Δ3), $\psi \in s_{i-1}$. By $\psi_0 \in s_{i-2}$ and (Δ3), $\psi \in s_{i-2}$. Continue and get $\psi \in s_1$.

Assume $\psi \in s_1$ and, for contradiction, $t, 1 \not\models \psi$. By (t1), $\psi_0 \in s_1$ or $\psi_1 \in s_1$. If $\psi_1 \in s_1$, then $t, 1 \models \psi_1$ by induction, so $t, 1 \models \psi$ contradicting our assumption. Hence, $\psi_0 \in s_1$ and $\psi_1 \notin s_1$. By (Δ3), $\psi \in s_2$. Also $t, 2 \not\models \psi$ since $t, 1 \not\models \psi$ by assumption and $t, 1 \models \psi_0$ by induction. Continuing like this gives $\psi \in s_1, s_2, s_3, \ldots$ and $\psi_0 \in s_1, s_2, s_3, \ldots$ and $\psi_1 \notin s_1, s_2, s_3 \ldots$. But then no s_i is in $F_\psi \in \mathcal{F}$, a contradiction. □

Corollary 3.6.11. *There is an algorithm that decides*

MC(LTL,TS)	
Input:	a transition system T and an LTL formula φ.
Problem:	does T satisfy $L_\omega(\varphi)$?

in time polynomial in $2^{|\varphi|} \cdot |T|$.

Proof. Argue as for Theorem 3.5.5. From the proof of Theorem 3.6.10 it is clear that $\mathbb{A}_{\neg\varphi}$ can be computed from φ in time polynomial in $2^{|\varphi|}$. □

We see, the factor $2^{|\varphi|}$ accounts for the size of $\mathbb{A}_{\neg\varphi}$. There is, however, not much room for improvement of Theorem 3.6.10:

Proposition 3.6.12. *For every* $k > 0$ *there is an LTL formula* φ_k *of length* $O(k^2)$ *such that every Büchi automaton* $\mathbb{A}$ *with* $L_\omega(\mathbb{A}) = L_\omega(\varphi)$ *has at least* 2^k *states.*

Proof. Set $\varphi_k(Y) := \bigwedge_{i=0}^{k-1}(\mathsf{X}^i Y \leftrightarrow \mathsf{X}^{k+i} Y)$. Then $L_\omega(\varphi_k)$ contains the ω-words with a prefix of the form $\beta_1 \cdots \beta_k \beta_1 \cdots \beta_k$ (with $\beta_i : \{Y\} \to \{0,1\}$). If $\mathbb{A}$ accepts these words, choose one and an accepting computation and let $s(\beta_1 \cdots \beta_k)$ be the state after k steps, i.e., after reading $\beta_1 \cdots \beta_k$. These states are pairwise distinct: if $s(\beta_1 \cdots \beta_k) = s(\beta'_1 \cdots \beta'_k)$, then $\mathbb{A}$ accepts a word with prefix $\beta_1 \cdots \beta_k \beta'_1 \cdots \beta'_k$, so $\beta_1 \cdots \beta_k = \beta'_1 \cdots \beta'_k$. □

3.7 Computation tree logic

Imagine a transition system T in which some states report a problem and some states give a response. Formally, T has variables including P and R. The liveness property "Whenever P is true, R is eventually true", is formalized by the LTL formula $\mathsf{G}(P \to \mathsf{F}R)$. But the property "whenever P is true, then R can be eventually true", i.e., "whenever P is true, then the execution can be continued in a way such that finally R is true" existentially quantifies over executions. This cannot be done in LTL. In *computation tree logic (CTL)* it is formalized by $\forall\mathsf{G}(P \to \exists\mathsf{F}R)$.

Definition 3.7.1 (Syntax and semantics of CTL)**.** Add $\exists$ to the alphabet of LTL and define the set of *CTL formulas* as the smallest set F of words that contains all variables $X_0, X_1, \ldots$ and such that for all $\varphi, \psi \in F$

$$\neg\varphi \in F,\ (\varphi \wedge \psi) \in F,\ \exists\mathsf{X}\varphi \in F,\ \exists\mathsf{G}\varphi \in F,\ \exists(\varphi\mathsf{U}\psi) \in F.$$

Writing $\varphi = \varphi(\bar{X})$ means all variables occurring in φ are among $\bar{X}$. Let $T = (\mathfrak{G}, \ell)$ be a transition system over $\bar{X}$. Let $\varphi(\bar{X})$ be a CTL formula. We define φ *is true in* T *at state* $g \in G$, symbolically $T, g \vDash \varphi$, by recursion on φ, stipulating for all LTL formulas ψ, χ and variables X from $\bar{X}$:

(T1) $T, g \vDash X \iff \ell(g)(X) = 1$;

(T2) $T, g \vDash \neg\psi \iff T, g \nvDash \psi$;

(T3) $T, g \vDash (\psi \wedge \chi) \iff T, g \vDash \psi$ and $T, g \vDash \chi$;

(T4) $T, g \vDash \exists\mathsf{X}\psi \iff$ there is an execution $g = g_1, g_2, \ldots$ (of T) with $T, g_2 \vDash \psi$;

(T5) $T, g \vDash \exists\mathsf{G}\psi \iff$ there is an execution $g = g_1, g_2, \ldots$ such that for all $i > 0$: $T, g_i \vDash \psi$;

(T6) $T, g \models \exists(\psi \mathsf{U} \chi) \iff$ there are an execution $g = g_1, g_2, \ldots$ and $i > 0$ such that for all $1 \leqslant j < i$: $T, g_i \models \chi$ and $T, g_j \models \psi$.

Remark 3.7.2. As usual, the definition by recursion on syntax is justified by a straightforward lemma on unique readability. Also the set $sub(\varphi)$ of *subformulas* of a formula φ is defined by recursion adding, in Example 1.1.8, the conditions: $sub(\exists \mathsf{X}\varphi) := \{\exists \mathsf{X}\varphi\} \cup sub(\varphi)$ and $sub(\exists \mathsf{G}\varphi) := \{\exists \mathsf{G}\varphi\} \cup sub(\varphi)$ and $sub(\exists(\varphi \mathsf{U}\psi)) := \{\exists(\varphi \mathsf{U} \psi)\} \cup sub(\varphi) \cup sub(\psi)$.

Lemma 3.7.3. *Let $\varphi(\bar{X}), \psi(\bar{X})$ be CTL formulas and set:*

$$\begin{array}{rcl} \forall \mathsf{X}\varphi & := & \neg\exists\mathsf{X}\neg\varphi, \\ \forall(\varphi \mathsf{U} \psi) & := & \big(\neg\exists(\neg\psi \mathsf{U}(\neg\varphi \wedge \neg\psi)) \wedge \neg\exists\mathsf{G}\neg\psi\big). \end{array}$$

Then for every transition system $T = (\mathfrak{G}, \ell)$ over $\bar{X}$:

1. $T, g \models \forall\mathsf{X}\varphi \iff$ *for every execution $g_1, g_2, \ldots$ with $g = g_1$: $T, g_2 \models \varphi$.*
2. $T, g \models \forall(\varphi\mathsf{U}\psi) \iff$ *for every execution $g_1, g_2, \ldots$ with $g = g_1$ there is $i > 0$ such that for all $1 \leqslant j < i$: $T, g_i \models \psi$ and $T, g_j \models \varphi$.*

Proof. 1 is trivial. 2 $\Rightarrow$: assume $T, g \models \forall(\psi\mathsf{U}\chi)$ and let $g = g_1, g_2, \ldots$ be an execution. By $T, g \not\models \exists\mathsf{G}\neg\psi$, there is $i > 0$ such that $T, g_i \models \psi$. Choose a minimal such i, so $T, g_j \models \neg\psi$ for all $1 \leqslant j < i$. Assume for contradiction that $T, g_j \models \neg\varphi$ for some $1 \leqslant j < i$. Then $T, g_j \models (\neg\varphi \wedge \neg\psi)$ and $g = g_1, g_2, \ldots$ witnesses $T, g \models \exists(\neg\psi\mathsf{U}(\neg\varphi \wedge \neg\psi))$, a contradiction.

2 $\Leftarrow$: assume the r.h.s.. Clearly, $T, g \models \neg\exists\mathsf{G}\neg\psi$. Assume for contradiction that $T, g \models \exists(\neg\psi\mathsf{U}(\neg\varphi \wedge \neg\psi))$, that is, there is an execution $g = g_1, g_2, \ldots$ and $i > 0$ such that $T, g_i \models (\neg\varphi \wedge \neg\psi)$ and $T, g_j \models \neg\psi$ for all $1 \leqslant j < i$. Then $T, g_j \not\models \psi$ for all $j \leqslant i$. Choose k with $T, g_k \models \psi$ and $T, g_j \models \varphi$ for all $1 \leqslant j < k$. Then $k > i$ and $T, g_i \models \varphi$, contradiction. □

Remark 3.7.4. Often one restricts attention to transition systems $T = (\mathfrak{G}, \ell)$ *without sinks*: every $g \in G$ has out-degree at least 1. Then $T, g \models \exists\mathsf{X}\varphi$ (resp. $\forall\mathsf{X}\varphi$) if and only if $T, g' \models \varphi$ for some (resp., all) $g' \in G$ with $(g, g') \in E^{\mathfrak{G}}$.

Notation: Define $\top := (X \vee \neg X)$ for a variable X and, for CTL formulas φ, ψ,

"Potentially φ":	$\exists\mathsf{F}\varphi := \exists(\top\mathsf{U}\varphi)$	"Inevitably φ":	$\forall\mathsf{F}\varphi := \forall(\top\mathsf{U}\varphi)$
"Potentially always φ":	$\exists\mathsf{G}\varphi$	"Invariantly φ":	$\forall\mathsf{G}\varphi := \neg\exists\mathsf{F}\neg\varphi$

In transition systems $T = (\mathfrak{G}, \ell)$ without sinks, $T, g \models \exists\mathsf{F}\varphi$ if and only if there is a path in $\mathfrak{G}$ from g to some g' with $T, g' \models \varphi$.

Exercise 3.7.5. Let $\varphi(\bar{X}), \psi(\bar{X})$ be CTL formulas and set $(\varphi \rightsquigarrow \psi) := \forall\mathsf{G}(\varphi \to \forall\mathsf{F}\psi)$, pronounced "$\varphi$ leads to ψ". Let T be a transition system over $\bar{X}$. Show:

$T, g \models (\varphi \rightsquigarrow \psi) \iff$ for every execution $g_1, g_2, \ldots$ with $g = g_1$ and all $i > 0$ there is $j \geqslant i$: if $T, g_i \models \varphi$, then $T, g_j \models \psi$.

$T, g \models \forall\mathsf{G}\forall\mathsf{F}\varphi \iff$ for every execution $g_1, g_2, \ldots$ with $g = g_1$ there are infinitely many $i > 0$ with $T, g_i \models \varphi$.

3.7.1 Model-checking CTL

Theorem 3.7.6. *There is an efficient algorithm that decides*

MC(CTL,TS)
Input: a CTL formula $\varphi(\bar{X})$, a transition system $T = (\mathfrak{G}, \ell)$ over $\bar{X}$ and $g \in G$.
Problem: $T, g \vDash \varphi$?

Proof. For a CTL formula $\psi(\bar{X})$ let $[\psi] := \{g \in G \mid T, g \vDash \psi\}$. It suffices to compute $[\psi]$ for every subformula ψ of φ. Since there are $\leqslant |\varphi|$ subformulas, it suffices to show how to efficiently compute these sets. This is done recursively using: $[X] = \{g \in G \mid \ell(g)(X) = 1\}$ for a variable X from $\bar{X}$, $[\neg\psi] = G \setminus [\psi]$, $[\psi \wedge \chi] = [\psi] \cap [\chi]$. Further:

1. $[\exists \mathsf{X}\psi]$ contains g if and only if there is a *relevant* $g_0 \in [\psi]$ with $(g, g_0) \in E^{\mathfrak{G}}$. *Relevant* means there exists an execution starting at g_0; equivalently, there are $g_1 \in G$, a path from g_0 to g_1 and a cycle on g_1; a *cycle on* g_1 is a path from g_1 to g_1 with at least one edge (i.e., $(g_1, g_1) \in E^{\mathfrak{G}}$ or there are paths from g_1 to some $g_2 \neq g_1$ and back).
2. $[\exists \mathsf{G}\psi]$ contains g if and only if there are a path from g to some g' and a cycle on g' – in $\langle[\psi]\rangle^{\mathfrak{G}}$, the subgraph induced on $[\psi]$.
3. $[\exists(\psi \mathsf{U} \chi)]$ contains g if and only if there are $g_0, g_1 \in G$ and a path from g to g_0 in $\langle[\psi]\rangle^{\mathfrak{G}}$ such that g_1 is relevant, $(g_0, g_1) \in E^{\mathfrak{G}}$ and $g_1 \in [\varphi]$.

In each case the r.h.s. of the equivalence is efficiently checked using an efficient algorithm for REACHABILITY. □

Remark 3.7.7. The above is often formulated only for transitions systems without sinks and the model-checker computes fixed-points according to the following exercise. As seen, this restriction is superfluous.

Note that eliminating subformulas $\forall(\varphi \mathsf{U} \psi)$ by their definition can lead to an exponential blow-up (since ψ is repeated three times) in length. However, the number of subformulas is not increased, so the theorem holds true for formulas with $\forall(\varphi \mathsf{U} \psi)$.

Exercise 3.7.8. Let $\varphi(\bar{X}), \psi(\bar{X})$ be CTL formuals and $T = (\mathfrak{G}, \ell)$ a transition system over $\bar{X}$ without sinks.

1. $[\exists \mathsf{G}\varphi]$ is the largest set $S \subseteq G$ such that $S \subseteq [\varphi]$ and for every $g \in T$ there is $g' \in T$ with $(g, g') \in E^{\mathfrak{G}}$.
2. $[\exists(\varphi \mathsf{U} \psi)]$ is the smallest set $S \subseteq G$ such that $[\psi] \subseteq S$ and S contains every $g \in [\varphi]$ such that $(g, g') \in E^{\mathfrak{G}}$ for some $g' \in S$.