Universität Innsbruck Academic Year 2013/14

Lecture Notes

Module Automated Reasoning

Notes for the Lectures in 2013/2014

Georg Moser

Winter 2013

2nd edition

This document has been produced with the help of KOMA-Script and LaTeX.

Contents

Preface

These course notes are for the module *Automated Reasoning* are are aimed at students in the Master of Science program of Computer Science at the University of Innsbruck. The module consists of two lectures (i) *Computational Logic* and (ii) *Automated Reasoning*, as well as a (practical) seminar. The lecture on *Computational Logic* is held in the winter term, while the lecture on *Automated Reasoning* is held in the successive summer term.

The lecture *Automated Reasoning* builds upon the material presented in the lecture *Computational Logic*, so in principle it is suggested to first hear the logic lecture. However, the lecture notes explain all topics in sufficient detail, so that it is possible to understand (and pass) the lecture on *Automated Reasoning*, if a mild amount of self-study on the logic part is performed. In the first part of these lecture notes the following topics are mainly discussed:

- Syntax, Semantics and Formal Systems of Propositional and First-Order Logic (including equality)
- Curry-Howard Isomorphism
- Herbrand's Theorem
- Extensions of First-order Logic like Second-Order Logic

Furthermore in the second part (among others) the following topics are addressed:

- Resolution and Tableau provers for First-order Logic
- Paramodulation, Ordered Completion and Proof Orders, Superposition
- Applications of of Automated Reasoning

Note that these notes are not meant to *replace* the lecture, but to *accompany* it. In particular in the following almost no examples will be given. This is on purpose, as plenty of examples will be discussed in the lecture. Beware that these lecture notes assume the reader to be familiar with general logical concepts as for example provided by the lecture on *Logic in Computer Science* (*LICS* for short) held by Prof. Aart Middeldorp[1] or by text books covering this topic (see for example [39, 10, 29]).

Similar material as is covered in these lecture notes can be found in the following text books (in the order of importance): [11, 20, 36, 41]. For additional references see [19, 23, 7, 27].

[1] See `http://cl-informatik.uibk.ac.at/teaching/ws11/lics` for the online information on the course "Logic in Computer Science" as offered in winter 2011.

Part I.

Computational Logic

1. Why Logic is Good For You

Logic is defined as the study of the *principle of reasoning* and mathematical logic is defined as the study of principles of mathematical reasoning. In order to explain what is meant with "study of reasoning" we consider the two (correct) arguments given below.

> *A mother or father of a person is an ancestor of that person. An ancestor of an ancestor of a person is an ancestor of a person. Sarah is the mother of Isaac, Isaac is the father of Jacob. Thus, Sarah is an ancestor of Jacob.*

and

> *A square or cube of a number is a power of that number. A power of a power of a number is a power of that number.* 64 *is the cube of* 4, 4 *is the square of* 2. *Thus,* 64 *is a power of* 2.

On the surface these two argument are different: the first argument is concerned with parenthood and ancestors, while the second one refers to mathematics and number theory in particular. However, employing the language of first-order logic (see Chapter 3) we can express both arguments as follows:

$$\begin{array}{ll}
\text{assume} & \forall x \forall y((\mathsf{R}_1(x,y) \lor \mathsf{R}_2(x,y)) \to \mathsf{R}_3(x,y)) \\
\text{assume} & \forall x \forall y \forall z((\mathsf{R}_3(x,y) \land \mathsf{R}_3(y,z)) \to \mathsf{R}_3(x,z)) \\
\text{assume} & \mathsf{R}_1(\mathsf{c}_1,\mathsf{c}_2) \land \mathsf{R}_2(\mathsf{c}_2,\mathsf{c}_3) \\
\text{thus} & \mathsf{R}_3(\mathsf{c}_1,\mathsf{c}_3)\ .
\end{array}$$

Here R_1, R_2, R_3 denote binary predicate constants (aka[1] predicate symbols), while c_1, c_2, c_3 denote individual constants (aka constant symbols). Depending on the way we *interpret* these symbols in a given structure we obtain either the first argument or the second argument. Using a bit more formalism (again see Chapter 3 for details) we can write the generalised argument as follows.

$$\left.\begin{array}{l}
\forall x \forall y((\mathsf{R}_1(x,y) \lor \mathsf{R}_2(x,y)) \to \mathsf{R}_3(x,y)) \\
\forall x \forall y \forall z((\mathsf{R}_3(x,y) \land \mathsf{R}_3(y,z)) \to \mathsf{R}_3(x,z)) \\
\mathsf{R}_1(\mathsf{c}_1,\mathsf{c}_2) \land \mathsf{R}_2(\mathsf{c}_2,\mathsf{c}_3)
\end{array}\right\} \models \mathsf{R}_3(\mathsf{c}_1,\mathsf{c}_3)\ . \tag{1.1}$$

[1] also known as

Using the technology discussed in Chapter 3 we can easily verify that the *consequence* depicted in (1.1) is valid. Hence the argument used to deduce that either Sarah is an ancestor of Jacob or that 64 is a power of 2 is not only correct, but general in the sense that the correctness of this argument does not depend on the interpretation of the symbols used in (1.1). Using standard methods in automated resoning, the validity of (1.1) can be verified automatically (in an instant).

Nowadays computer science is more prominent in the use of (mathematical) logic than mathematics itself and logic has grown to be more relevant to computer science than any other branch of mathematics (compare [52]). Below we give some application areas of logic in computer science.

1.1. Minesweeper

Consider Figure 1.1 which shows a typical configuration that may appear during a play of Minesweeper:[2]

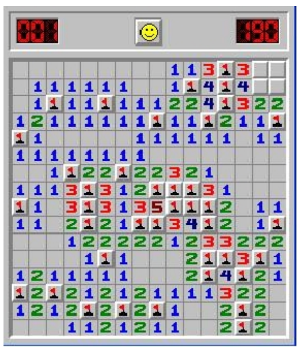

Figure 1.1.: A Minesweeper Configuration

Richard Kaye has shown in [32] that the problem whether an arbitrary configuration on a Minesweeper is indeed a *possible* configuration that can be reached through a sequence of moves is NP-complete. On the other hand, we can employ standard SAT solvers like for example MiniSat[3] to play Minesweeper fully automatically (although the first move has to be guessed). Such a Minesweeper solver has been implemented by Christoph Rungg (see [47]).

The central idea of such an implementation is the encoding of the rules of the game as a (large) set S of propositional formulas. As soon as this encoding is established any satisfying assignment for S can be re-translated into a solution to the original question in the context of the game. Due to the efficiency of modern SAT solvers it is typically the case that this approach outperforms any ad-hoc search method that tries to find the correct next move directly. Similar ideas can be used to easily implement very efficient solvers for logic puzzles.[4]

[2] `http://en.wikipedia.org/wiki/Minesweeper_(computer_game)`.
[3] `http://minisat.se/`
[4] See `http://cl-informatik.uibk.ac.at/software/puzzles/` for a collection of logic puzzle solvers.

These (toy) examples serve as a reminder of the huge importance of SAT technology in providing efficient and powerful techniques to implement search methods (compare [33]).

1.2. Program Analysis

Interesting properties of programs (like termination) are typically undecidable. Despite this limitation such properties are studied and automatic procedures have been designed to (partially) verify whether certain properties hold.

In the analysis of programs one doesn't study the concretely given program, but abstracts it in a suitable way, abstract interpretations [14] formalise this idea. Here the level of abstraction is crucial if one wants to prevent *false negatives*: properties that hold true for the program become false for the abstraction. In order to design expressive abstractions one combines simple abstractions into more complicated and thus more expressive ones.

Sumit Gulwani and Ashish Tiwari have presented a methodology to automatically combine abstract interpretations based on specific theories to construct an abstract interpreter based on the combination of the studied theories. This is encapsulated into the notion of *logical product* (compare [26]) and based on the Nelson-Oppen method for combining decision procedures of different theories (compare [38]). Here a *theory* is simply a set of sentences (over a given language) that is closed under logical consequence. Examples of theories would be for example the theory of linear arithmetic (making use of the symbols 0, 1, $+$, $\times$, $\leqslant$, and $=$) or the theory of lists (making use of the symbols car, cdr, cons, and $=$). If two theories T_1, T_2 fulfil certain conditions[5] and it is known that satisfiability of quantifier-free formulas with respect to the theories T_1 and T_2 is decidable, then satisfiability of quantifier-free formulas with respect to the union $T_1 \cup T_2$ is decidable. In Chapter 5 we study a related result, Robinson's joint consistency theorem.

The methodology invented in [26] allows the modularisation of the analysis of programs via abstract interpretations. Modularisation is possible for both stages of the analysis: One one hand the technique can be employed to define suitable interpretations for complex theories. On the other hand it can be employed to simplify the implementation of such an abstract interpreter.

1.3. Databases

Datalog is a database query language based on the logic programming paradigm. Syntactically it is a subset of Prolog (compare [12]). It is widely used in knowledge representation systems, see for example [22]. Logically a datalog query is a formula in Horn logic. Hence any such query has a unique model, its minimal model. This allows to assign a simple and unique semantics to datalog programs.

[5] To be precise the theories T_1, T_2 are supposed to be *convex*, *disjoint*, and *stably infinite*, see [38].

Datalog rules can be translated into inclusions in relational databases. Datalog extends positive relational algebras as recursive queries can be formed, which is not possible in positive relational algebras. The success of datalog can for example be witnessed in changes to the database query language SQL that has been extended by the possibility of recursive queries.

Contrary to full first-order logic, datalog queries are decidable. One can distinguish two notions of complexity in this context. On one hand we have *expression complexity*, where the complexity of fulfilling a given query is expressed in relation to the size of the query. On the other hand we have *data complexity*, where the complexity is measured in the size of the database and the query. The former notion is closely related to the notion of complexity of formal theories. Hence we focus on this notion. The expression complexity of datalog is EXPTIME-complete, that is, far beyond the complexity of typical intractable problems like for example SAT.

Thomas Eiter et al. extended datalog to *disjunctive datalog*. Disjunctive datalog allows disjunctions in heads of rules (compare [21]). It is a strict extension of SQL and forms the basis of *semantic web* applications and has connections to *description logics* and *ontologies*. Disjunctive datalog queries can be extended with negation, so that the typical closed-world semantics of negation can be overcome. To indicate the expressivity of disjunctive datalog observe that the travelling salesperson problem can be directly formulated in this database query language. Disjunctive datalog remains decidable, but the expression complexity becomes $\mathsf{NEXPTIME}^{\mathsf{NP}}$-complete. This implies that such queries can be only solved on a nondeterministic Turing machine that runs in exponential time and employs an NP-oracle.

2. Propositional Logic

This chapter recalls the language of propositional logic, that is, its *syntax* and its meaning, that is, its *semantics* (see Section 2.1). Furthermore, we recall the rules of *natural deduction* and the rules of *resolution* for propositional logic (see Section 2.2 and Section 2.3). Finally, in Section 2.4 we report on the use of many-valued propositional logics in medical expert systems.

2.1. Syntax and Semantics of Propositional Logic

Let $p_1, p_2, \ldots, p_j, \ldots$ denote an infinite set of *propositional atoms*, denoted by p, q, r. The set of all propositional atoms is denoted by AT. In addition we make use of the truth constants $\top$ and $\bot$.

Definition 2.1. The (propositional) connectives of propositional logic are

$$\neg \qquad \wedge \qquad \vee \qquad \rightarrow \ ,$$

and the *(propositional) formulas* are defined inductively as follows:

(i) A propositional atom p or a truth constant is a formula, and

(ii) if A, B are formulas, then

$$\neg A \qquad (A \wedge B) \qquad (A \vee B) \qquad (A \rightarrow B) \ ,$$

are also formulas.

Convention. In order to drop brackets, we use the following precedence: $\neg$ binds stronger than $\vee$ and $\wedge$, which in turn bind stronger than $\rightarrow$. Furthermore, we tacitly assume right-associativity of $\rightarrow$.

This completes the definition of the *syntax* of propositional logic. In the remainder of this section we define its *semantics*. We write T, F for the two truth values, representing “true” and “false” respectively.

Definition 2.2. An assignment $\mathsf{v} \colon \mathsf{AT} \rightarrow \{\mathsf{T}, \mathsf{F}\}$ is a mapping that associates atoms with truth values.

We write $\mathsf{v}(F)$ for the *valuation* of the formula F. The valuation $\mathsf{v}(F)$ is defined as the extension of the assignment v to formulas, using the following truth tables:

$\neg$	
T	F
F	T

$\wedge$	T	F
T	T	F
F	F	F

$\vee$	T	F
T	T	T
F	T	F

$\to$	T	F
T	T	F
F	T	T

Definition 2.3. The *consequence relation*, denoted as $A_1, \ldots, A_n \models B$, asserts that $\mathsf{v}(B) = \mathsf{T}$, whenever $\mathsf{v}(A_1)$, ..., $\mathsf{v}(A_n)$ is true for any assignment v. We write $\models A$, instead of $\varnothing \models A$ and call A a *tautology* or *valid* in this case.

We call two formulas *(logically) equivalent* (denoted as $A \equiv B$) if $A \models B$ and $B \models A$ hold.

2.2. Natural Deduction

We recall the rules of natural deduction. We assume the reader is acquainted with some notion of formal proof system and will only briefly motivate the rules. See [29] for additional information.

Georg Gentzen introduced the calculus of *natural deduction*, whose rules for propositional logic are given in Figure 2.1. The calculus aims to mimic the "natural" way in which mathematical proofs are performed, for example the disjunction elimination rule is best understood as an inference rule that represents a proof by case analysis.

Let $\mathcal{G}$ be a finite set of formulas and let F be a formula. A *natural deduction proof* is a sequence of applications of rules depicted in Figure 2.1. If there exists a natural deduction proof of F with assumptions $\mathcal{G}$, then we say F is *provable* (or *derived*) from $\mathcal{G}$.

Definition 2.4. The *provability relation*, denoted as $A_1, \ldots, A_n \vdash B$, asserts that B is derived from the assumptions A_1, ..., A_n. This notion extends to infinite sets of formulas $\mathcal{G}$: We write $\mathcal{G} \vdash F$ if there exists a finite subset $\mathcal{G}' \subseteq \mathcal{G}$ such that $\mathcal{G}' \vdash F$. We write $\vdash A$ instead of $\varnothing \vdash A$ and call the formula A *provable* in this case.

We say that a set of formulas $\mathcal{G}$ is *consistent* if we cannot find a proof of $\bot$ from $\mathcal{G}$. A set of formulas $\mathcal{G}$ is called *inconsistent* if there exists a proof of $\bot$ from $\mathcal{G}$. A proof is sometimes also called a *derivation*.

The proof of the following theorem can for example be found in [29].

Theorem 2.1. *Natural deduction is* sound *and* complete *for propositional logic, that is, the following holds:*

$$A_1, \ldots, A_n \models B \iff A_1, \ldots, A_n \vdash B \ .$$

Note that natural deduction is not the only formal system that is sound and complete for propositional logic, but only one among many. This motivates the next definition.

Definition 2.5. If there exists a *finite* system of axioms and inference rules that is sound and complete for a logic, we say this logic is *finitely axiomatised* by such a system.

In the next section we briefly introduce propositional resolution which forms another sound and complete proof system for propositional logic.

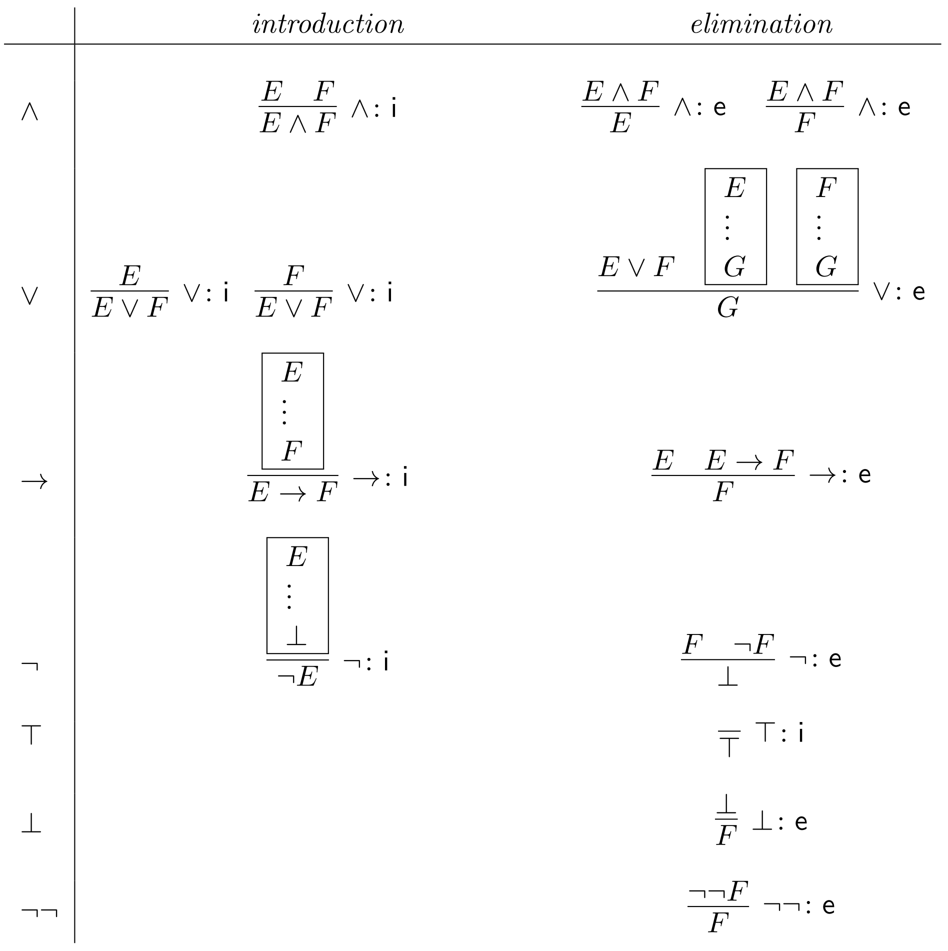

Figure 2.1.: Natural Deduction for Propositional Logic

2.3. Propositional Resolution

A *literal* is a propositional atom p or its negation $\neg p$. A formula F is said to be in *conjunctive normal form* (*CNF* for short) if F is a conjunction of disjunctions of literals. For brevity we often speak of "a CNF F" instead of "a formula F in CNF".

The next lemma is easy, a complete proof can for example be found in [29].

Lemma 2.1. *For all formulas A, there exists a formula B in CNF, such that $A \equiv B$.*

Definition 2.6. A *clause* is a disjunction of literals, defined inductively as follows:

(i) The empty clause (denoted as $\Box$) is a clause,

(ii) literals are clauses, and

(iii) if C, D are clauses, then $C \vee D$ is a clause.

As usual disjunction $\lor$ is associative and commutative. In addition we define the following identities:

$$p = \neg\neg p \qquad \Box \lor \Box = \Box \qquad C \lor \Box = \Box \lor C = C \ ,$$

where p denotes a propositional atom and C an arbitrary clause.

Note that the language to define clauses is a variation of our language for propositional logic: instead of the truth constant $\bot$ we use $\Box$ and we make only use of the connectives $\neg$ and $\lor$. It is easy to see that a formula F in CNF directly gives rise to a set of clauses $\mathcal{C}$, where $\mathcal{C}$ is defined as the collection of disjunctions in F. On the other hand Lemma 2.1 implies that for every formula F there exists a CNF F', which can then be directly represented as a clause set $\mathcal{C}$. We call $\mathcal{C}$ the *clause form* of F.

John Alan Robinson invented the *resolution calculus* (for first-order logic), whose propositional rules are given in Figure 2.2. The resolution calculus was invented in the 1960s and various different presentations are known. See [36] for a complete treatment of the differences between existing calculi. In particular note that the resolution and factoring rule can also be combined into a single rule [29].

$$\textit{resolution} \quad \frac{C \lor p \quad D \lor \neg p}{C \lor D} \qquad\qquad \textit{factoring} \quad \frac{C \lor l \lor l}{C \lor l} \quad l \text{ a literal}$$

Figure 2.2.: Resolution for Propositional Logic

Definition 2.7. Let $\mathcal{C}$ be a set of clauses. Then we define the *resolution operator* $\mathsf{Res}(\mathcal{C})$ as follows:

$$\mathsf{Res}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 2.2 with premises in } \mathcal{C}\} \ .$$

Based on this we define the n^{th} and the unlimited iteration of the resolution operator as follows:

$$\mathsf{Res}^0(\mathcal{C}) := \mathcal{C} \qquad\qquad \mathsf{Res}^{n+1}(\mathcal{C}) := \mathsf{Res}^n(\mathcal{C}) \cup \mathsf{Res}(\mathsf{Res}^n(\mathcal{C}))$$
$$\mathsf{Res}^*(\mathcal{C}) := \bigcup_{n \geqslant 0} \mathsf{Res}^n(\mathcal{C}) \ .$$

We say the empty clause is derivable from $\mathcal{C}$ if $\Box \in \mathsf{Res}^*(\mathcal{C})$.

Let $\mathcal{C}$ be a set of clauses. If $\mathsf{Res}(\mathcal{C}) \subseteq \mathcal{C}$, then the clause set $\mathcal{C}$ is called *saturated*. Obviously, we have that $\mathsf{Res}^*(\mathcal{C})$ is saturated. If for a clause D, $D \in \mathsf{Res}^*(\mathcal{C})$, then we say that D is *derived* from $\mathcal{C}$ by resolution. If for a saturated set $\mathcal{C}$, $\Box \notin \mathcal{C}$, then $\mathcal{C}$ is called *consistent*, otherwise $\mathcal{C}$ is said to be *inconsistent*.

Suppose $\mathcal{C}$ is inconsistent. Then it is easy to see that $\mathcal{C}$ (and the formula F represented by $\mathcal{C}$) are *unsatisfiable*. In other words (propositional) resolution is a sound proof method. Furthermore, we have the following theorem, whose proof can be found for example in [36, 29].

Theorem 2.2. *Let F be a formula and let $\mathcal{C}$ denote its clause form. Propositional resolution is sound and complete, that is, the following holds:*

$$F \text{ is unsatisfiable} \iff \Box \in \mathsf{Res}^*(\mathcal{C}) \; .$$

Observe that the resolution calculus is a refutation based technique, whose aim is to derive the empty clause, that is, a contradiction. On the contrary the calculus of natural deduction aims to prove the validity of a given formula. Hence to prove the validity of a given formula F by resolution, we have to consider the clause form $\mathcal{C}$ of its negation $\neg F$. This entails that an application of resolution may require the translation of an arbitrary formula into CNF. If the latter is done naively, this transformation may be quite costly.

Before we turn to an application of propositional logic in the next section, we mention a general theorem on propositional logic, whose easy proof is left to the reader.

Theorem 2.3. *Let $A \to C$ be a valid formula. Then there exists a formula B such that $A \to B$ and $B \to C$ are valid. Furthermore, the* interpolant *B contains only propositional atoms that occur both in A and C.*

2.4. Many-Valued Propositional Logics

We briefly remark on the possibility to replace the two truth values T and F used in *classical* propositional logic with infinitely many truth values.

Let $V \subseteq [0,1]$ denote a set of finitely or infinitely many truth values containing at least the truth values 0, 1, representing "false" and "true", respectively.

Definition 2.8. A *Lukasiewicz assignment* (based on V) is a mapping $\mathsf{v}\colon \mathsf{AT} \to V$ and the assignment v is extended to a *(Lukasiewicz) valuation* of formulas as follows:

$$\begin{aligned}
\mathsf{v}(\neg A) &= 1 - \mathsf{v}(A)\\
\mathsf{v}(A \wedge B) &= \min\{\mathsf{v}(A), \mathsf{v}(B)\}\\
\mathsf{v}(A \vee B) &= \max\{\mathsf{v}(A), \mathsf{v}(B)\}\\
\mathsf{v}(A \to B) &= \min\{1, 1 - \mathsf{v}(A) + \mathsf{v}(B)\}
\end{aligned}$$

A formula F is *valid* if $\mathsf{v}(A) = 1$ for all assignments v based on V.

Logics with more than two truth values are called *many-valued logics* or *fuzzy logics*. Many-valued logics, based on Lukasiewicz valuations are called *Lukasiewicz logics*.

Theorem 2.4. *Finite- or infinite-valued Lukasiewicz logics are finitely axiomatisable. Furthermore validity is decidable for propositional Lukasiewicz logics. More precisely the validity problem for these logics is* coNP*-complete.*

Although many-valued logics have been introduced for purely theoretical reasons they find a number of applications in modelling uncertainty. Note for example that the database language SQL uses a third truth value (called *unknown*) to model unknown data.

If we consider infinitely many truth values V from the real interval $[0, 1]$, we can conceive these values as assigning *probabilities* to propositions. In this interpretation infinite-valued logics can be used to model the behaviour of data bases or medical expert systems.

CADIAG (*Computer Assisted DIAGnosis*) is a series of medical expert systems developed at the Vienna Medical University (since the 1980s). The latest system is called CADIAG-2, see [53] for more details. This expert system is rule based and for example contains rules of the following form:

```
IF suspicion of liver metastases by liver palpation
THEN pancreatic cancer
with degree of confirmation 0.3
```

CADIAG-2 is characterised by its ability to process not only definite truth or falsity, but also indeterminate (vague or uncertain) information. The inference system of CADIAG-2 can be expressed as an infinite valued fuzzy logics and this formalisation revealed inconsistencies in the rule based knowledge representation.

Problems

Problem 2.1. Verify whether the following propositional formulas are (i) satisfiable, (ii) valid, or (iii) unsatisfiable:

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

(ii) $(p \to (q \to p))$

(iii) $((p \to (q \to r)) \to ((p \to q) \to (p \to r)))$

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

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

Problem 2.2. Show that the following claims about the consequence relation are correct:

(i) $(p \to q) \land p \models q$

(ii) $(p \to q) \land \neg q \models \neg p$

(iii) $p \to q \not\models q \to p$

(iv) $(p \lor q) \land \neg p \models q$

(v) $\neg(p \land q) \not\models (\neg p \land \neg q)$

Problem 2.3. Show that the following inference rules are derivable in (propositional) natural deduction:

(i) $\dfrac{A \to B \quad \neg B}{\neg A}$

(ii) $\dfrac{A}{\neg\neg A}$

(iii) $\dfrac{}{A \vee \neg A}$

Problem 2.4. Show that the following (propositional) clause sets are unsatisfiable:

(i) $\mathcal{C} = \{p, q, \neg r, \neg p \vee \neg q \vee r\}$

(ii) $\mathcal{C} = \{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$

(iii) $\mathcal{C} = \{p, \neg p \vee q \vee r, \neg p \vee \neg q \vee \neg r, \neg p \vee s \vee t, \neg p \vee \neg s \vee \neg t, \neg s \vee q, r \vee t, s \vee \neg t\}$

Problem 2.5. Show that the formula $(p \wedge q \to r) \to p \to q \to r$ is valid, using resolution.

Problem 2.6. Let F be a propositional formula, where zero, one or more subformulas G are replaced by a logically equivalent formula G'. Then we obtain a formula F' that is logically equivalent to the formula F.

(i) Give a precise definition of this process of substitution.

(ii) Show the correctness of the claim.

Problem 2.7. Show Theorem 2.3.
Hint: Proceed by induction on the number of atoms that occur in A but not in B.

Problem 2.8. Consider the following alternative to Definition 2.7:

$$\mathsf{Res}_1^0(\mathcal{C}) := \mathcal{C} \qquad \mathsf{Res}_1^{n+1}(\mathcal{C}) := \mathsf{Res}(\mathsf{Res}_1^n(\mathcal{C}))$$
$$\mathsf{Res}_1^*(\mathcal{C}) := \bigcup_{n \geqslant 0} \mathsf{Res}_1^n(\mathcal{C}) \ .$$

Is this definition equivalent to the original one? In particular try to establish whether we have $\Box \in \mathsf{Res}^*(\mathcal{C})$ iff $\Box \in \mathsf{Res}_1^*(\mathcal{C})$ for any clause set $\mathcal{C}$.

3.

Syntax and Semantics of First-Order Logic

This chapter recalls the language of first-order logic, that is, its *syntax* and its *semantics*. This will be established in the first two sections. Finally, in Section 3.3 we state and prove the isomorphism theorem.

3.1. Syntax of First-Order Logic

A first-order *language* is determined by specifying its *constants*, *variables*, *logical symbols*, and other auxiliary symbols like brackets or comma. In particular *constants* are:

(i) Individual constants: $k_0, k_1, \ldots, k_j, \ldots$

(ii) Function constants with i arguments: $f_0^i, f_1^i, \ldots, f_j^i, \ldots$

(iii) Predicate constants with i arguments: $R_0^i, R_1^i, \ldots, R_j^i, \ldots$

Here $i = 1, 2, \ldots$ and $j = 0, 1, 2, \ldots$ While *variables* are

(i) $x_0, x_1, \ldots, x_j, \ldots$

Here $j = 0, 1, 2, \ldots$ As *logical symbols* we have *truth constants*, *propositional connectives*, and *quantifiers*:

(i) Truth contants: $\bot$, $\top$

(ii) Propositional connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$.

(iii) Quantifiers: $\forall$, $\exists$.

As soon as the constants of a language $\mathcal{L}$ are fixed, the language $\mathcal{L}$ is fixed. Any finite sequence of symbols (from a language $\mathcal{L}$) is called an *expression.* We often include one more “logical symbol”, the equality sign $=$. To be precise the expression $=$ is a predicate constant, but for convenience we count it as a logical symbol. This is done as we often want to assume that equality is part of our language without explicitly remarking on its presence as one of the constants. This is the only exception, all other (predicate) constants are referred to as *non-logical symbols.*

Convention. If $\mathcal{L}$ is clear from context the phrase "of $\mathcal{L}$" will be dropped. The meta-symbols c, d, f, g, h, ... are used to denote individual constants and function symbols, while the meta-symbols P, Q, R, ... vary through predicate symbols. Variables are denoted by a, b, ... or we use x, y, z, and so forth.

As defined above the cardinality of the constants and variables in any language is countable. In this case we call the language *countable* or *enumerable.* To assume a countable language is a restriction, but this restriction is standard.

Definition 3.1. *Terms* are defined as follows:

(i) Any individual constant c is a term.

(ii) Any variable x is a term.

(iii) If $t_1, \dots, t_n$ are terms, f an n-ary function symbol, then $f(t_1, \dots, t_n)$ is a term.

Note that (as explained above) the phrase "of $\mathcal{L}$" for a pre-assumed language $\mathcal{L}$ has been dropped.

Definition 3.2. If P is a predicate constant with arity n and $t_1, \dots t_n$ are terms, then $P(t_1, \dots, t_n)$ is called an *atomic formula.* If the equality sign is present then $t_1 = t_2$ is also an atomic formula.

Definition 3.3. *(First-order) formulas* are defined as follows:

(i) Atomic formulas are formulas.

(ii) If A and B are formulas, then $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, and $(A \to B)$ are formulas.

(iii) If A is a formula, x is a variable, then $\forall x A$ and $\exists x A$ are formulas.

Convention. Terms are often denoted as s, t, ..., and formulas are often denoted by A, B, C, ..., F, G,...

Let F be a formula. A variable x that occurs in F inside the scope of a quantifier $\mathsf{Q} \in \{\forall, \exists\}$ is called *bound.* If a variable x does not occur inside the scope of any quantifier, this variable is called *free.* This definition is somewhat imprecise. The precise definition is delegated to the problem section. A formula that does not contain free variables is called *closed.* Sometimes we refer to a closed formula as a *sentence.*

It is often convenient to indicate occurrences of variables in a formula. Suppose F is a formula and let x denote a free variable occurring in F. We write $F(x)$ instead of F to indicate all occurrences of x in F. Let t be a term. We write $F(t)$ to denote the formula obtained from $F(x)$, where all occurrences of x are replaced by t.

Example 3.1. Let F be a formula over the language $\mathcal{L} = \{\mathsf{P}, \mathsf{Q}\}$, where P is unary and Q binary. Suppose $F = \forall x(\mathsf{P}(x) \wedge \mathsf{Q}(x, y))$. Using the above convention, we set $F = \forall x G(x)$, where $G(x) := (\mathsf{P}(x) \wedge \mathsf{Q}(x, y))$.

This notation is particular convenient, when one refers to instances of quantified formulas. Let t be an arbitrary term. Then $G(t) = (\mathsf{P}(t) \wedge \mathsf{Q}(t, y)$ is an *instance* of the formula $\forall x G(x)$.

If this does not affect the readability of formulas we will omit parentheses. In particular parentheses are omitted in the case of double negation. We write $\neg\neg A$ instead of $\neg(\neg A)$. Moreover we use the following convention on the priority of the logical symbols.

Convention. Extending the convention introduced in Chapter 2 we assert that quantifiers $\forall$, $\exists$ bind stronger than $\neg$. Furthermore we often write $s \neq t$ as abbreviation for $\neg(s = t)$.

3.2. Semantics of First-Order Logic

In the following $\mathcal{L}$ always denotes an arbitrary, but fixed language. Recall that we drop reference to $\mathcal{L}$ if no confusion can arise.

Definition 3.4. A *structure* is a pair $\mathcal{A} = (A, a)$ such that:

(i) A is a non-empty set, called *domain* or *universe* of the structure.

(ii) The mapping a associates constants with the domain:
   - Every individual constant c is associated with an element $a(c) \in A$.
   - Every n-ary function constant f is associated with an n-ary function $a(f)\colon A^n \to A$.
   - Every n-ary predicate constant P is associated with a subset $a(P) \subseteq A^n$.

(iii) The equality sign $=$ is associated with the identity relation $a(=)$.

Instead of $a(c)$, $a(f)$, $a(P)$, $a(=)$ we usually write $c^{\mathcal{A}}$, $f^{\mathcal{A}}$, $P^{\mathcal{A}}$, $=^{\mathcal{A}}$, respectively. Further, for brevity we write $=$ for the equality sign *and* the identity relation.

Remark. The definition of a structure is equivalent to the definition of *model* in [29]. The latter name is sometimes problematic, as the word "model" is often used in a more restrictive way, see below.

Definition 3.5. An *environment* (or a *look-up table*) for a structure $\mathcal{A}$ is a mapping $\ell\colon \{x_n \mid n \in \mathbb{N}\} \to A$ from the set of variables into the universe of A. By $\ell\{x \mapsto t\}$ we denote the environment mapping x to t and all other variables $y \neq x$ to $\ell(y)$.

Definition 3.6. An *interpretation* $\mathcal{I}$ is a pair $(\mathcal{A}, \ell)$ consisting of a structure $\mathcal{A}$ and an environment ℓ. The *value* of a term t (with respect to $\mathcal{I}$) is defined as follows:

$$t^{\mathcal{I}} := \begin{cases} \ell(t) & \text{if } t \text{ a variable} \\ f^{\mathcal{A}}(t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}) & \text{if } t = f(t_1, \dots, t_n) \ . \end{cases}$$

Let $\mathcal{I} = (\mathcal{A}, \ell)$ be an interpretation, we write $\mathcal{I}\{x \mapsto t\}$ for the interpretation $(\mathcal{A}, \ell\{x \mapsto t\})$.

Given an interpretation $\mathcal{I}$ and a formula F, we are going to define when $\mathcal{I}$ is a *model* of F. We also say that $\mathcal{I}$ *satisfies* F or that F *holds in* $\mathcal{I}$. In the following the word "model" is exclusively used in this sense.

Definition 3.7. Let $\mathcal{I} = (\mathcal{A}, \ell)$ be an interpretation and let F be a formula, we define the *satisfaction relation* $\mathcal{I} \models F$ inductively.

$$\begin{array}{lcl}
\mathcal{I} \models t_1 = t_2 & :\Longleftrightarrow & t_1^{\mathcal{I}} = t_2^{\mathcal{I}} \\
\mathcal{I} \models P(t_1, \dots, t_n) & :\Longleftrightarrow & (t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}) \in P^{\mathcal{A}} \\
\mathcal{I} \models \neg F & :\Longleftrightarrow & \mathcal{I} \not\models F \\
\mathcal{I} \models F \wedge G & :\Longleftrightarrow & \mathcal{I} \models F \text{ and } \mathcal{I} \models G \\
\mathcal{I} \models F \vee G & :\Longleftrightarrow & \mathcal{I} \models F \text{ or } \mathcal{I} \models G \\
\mathcal{I} \models F \rightarrow G & :\Longleftrightarrow & \text{if } \mathcal{I} \models F\text{, then } \mathcal{I} \models G \\
\mathcal{I} \models \forall x F & :\Longleftrightarrow & \text{if } \mathcal{I}\{x \mapsto a\} \models F \text{ holds for all } a \in A \\
\mathcal{I} \models \exists x F & :\Longleftrightarrow & \text{if } \mathcal{I}\{x \mapsto a\} \models F \text{ holds for some } a \in A\ .
\end{array}$$

If $\mathcal{G}$ is a set of formulas, we write $\mathcal{I} \models \mathcal{G}$ to indicate that $\mathcal{I} \models F$ for all $F \in \mathcal{G}$. We say $\mathcal{I}$ *models* $\mathcal{G}$ whenever $\mathcal{I} \models \mathcal{G}$ holds.

Definition 3.7 follows the presentation in [29, 20]. Note that this is not the only possibility to define that a given interpretation $\mathcal{I}$ models a formula F. Indeed in [27, 11] different approaches are taken that are essentially equivalent. The above approach has the advantage that the satisfaction relation is defined directly for *formulas*, whereas [27, 11] define the satisfaction relation first for *sentences*, which is later lifted to formulas. The here followed approach is slightly more technical, but conceptionally easier. The interested reader is kindly referred to [27, 11].

The next definitions lifts the satisfaction relation to a *consequence relation* (aka *semantic entailment relation*).

Definition 3.8. Let F, G be formulas and let $\mathcal{G}$ be a set of formulas. Then $\mathcal{G} \models F$ iff each interpretation of $\mathcal{G}$ that is a model, is also a model of F. Instead of $\{G\} \models F$ we write $G \models F$.

A formula F is called *satisfiable* if there exists an interpretation that is a model of F (denoted as $\mathsf{Sat}(F)$); F is called *unsatisfiable* if *no* interpretation is a model (denoted as $\neg\,\mathsf{Sat}(F)$). If F is satisfied by *any* interpretation, then we call F *valid* (denoted as $\models F$).

Lemma 3.1. *For all formulas F and all sets of formulas $\mathcal{G}$ we have that $\mathcal{G} \models F$ iff $\neg\,\mathsf{Sat}(\mathcal{G} \cup \{\neg F\})$.*

Proof. We have $\mathcal{G} \models F$ iff any interpretation that is a model of $\mathcal{G}$ is a model of F. This holds iff no interpretation is model of a $\mathcal{G}$ but not a model of F. This again holds iff $\mathcal{G} \cup \{\neg F\}$ is not satisfiable. $\square$

We call two formulas F and G *logically equivalent* if $F \models G$ and $G \models F$. As above this is denoted as $F \equiv G$. Clearly this is equivalent to $\models F \leftrightarrow G$, where the latter abbreviates $\models (F \rightarrow G) \wedge (G \rightarrow F)$. It is easy to see that for any formula F there exists a logically equivalent formula F' such that F' contains only $\neg$, $\wedge$ as connectives and the quantifier $\exists$. This fact comes in handy to simplify proofs by induction on F.

The proof of the following lemma is delegated to the problem section.

Lemma 3.2. *Let $\mathcal{I}_1 = (\mathcal{A}_1, \ell_1)$ and $\mathcal{I}_2 = (\mathcal{A}_2, \ell_2)$ be interpretations such that the respective universes coincide. Suppose F is a formula such that $\mathcal{I}_1$ and $\mathcal{I}_2$ coincide on the constants and variables occurring in F. Then $\mathcal{I}_1 \models F$ iff $\mathcal{I}_2 \models F$.*

Observe that the lemma states that for a given interpretation $\mathcal{I} = (\mathcal{A}, \ell)$ only a finite part of the look-up table ℓ is used as only finitely many variables may occur in a given formula F. In particular if F is a sentence, we may simplify the notation introduced in Definition 3.7. Instead of $\mathcal{I} \models F$, we simply write $\mathcal{A} \models F$ and say the structure $\mathcal{A}$ *models* F.

3.3. Models

In this section we state and prove the isomorphism theorem.

Definition 3.9. Let $\mathcal{A}$, $\mathcal{B}$ be two structures (with respect to the same language $\mathcal{L}$) and let A, B denote the respective domains. Suppose there exists a bijection $m \colon A \to B$ such that

(i) for any individual constant c, $m(c^{\mathcal{A}}) = c^{\mathcal{B}}$,

(ii) for any n-ary function constant f and all $a_1, \dots, a_n \in A$ we have
$$m(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(m(a_1), \dots, m(a_n)) \text{ , and}$$

(iii) for any n-ary predicate constant P and all elements $a_1, \dots, a_n \in A$ we have:
$$(a_1, \dots, a_n) \in P^{\mathcal{A}} \iff (m(a_1), \dots, m(a_n)) \in P^{\mathcal{B}} \text{ .}$$

Then m is called an *isomorphism*. We write $m \colon \mathcal{A} \cong \mathcal{B}$ to denote m and write $\mathcal{A} \cong \mathcal{B}$ if there exists an isomorphism $m \colon \mathcal{A} \to \mathcal{B}$.

The proof of the next lemma is not difficult and left to the reader.

Lemma 3.3. *Let A, B be sets such that there exists a bijection m between them. Then if $\mathcal{A}$ is a structure with domain A, there exists a structure $\mathcal{B}$ with domain B such that $\mathcal{A} \cong \mathcal{B}$.*

Theorem 3.1. *Let $\mathcal{A}$, $\mathcal{B}$ be structures such that $\mathcal{A} \cong \mathcal{B}$. Then for every sentence F we have $\mathcal{A} \models F$ iff $\mathcal{B} \models F$.*

Proof. Assume $m \colon \mathcal{A} \cong \mathcal{B}$; in proof we show that the same formulas hold if one uses corresponding environments together with the structures $\mathcal{A}$, $\mathcal{B}$. Let $\mathcal{I}$ be an interpretation. With an environment $\ell \in \mathcal{I}$ we associate the environment $\ell^m := m \circ \ell$. Let $\mathcal{I} = (\mathcal{A}, \ell)$ and $\mathcal{J} = (\mathcal{B}, \ell^m)$. Then we show by induction:

(i) For every term t: $m(t^{\mathcal{I}}) = t^{\mathcal{J}}$.

(ii) For every formula F: $\mathcal{I} \models F$ iff $\mathcal{J} \models F$.

The proof of the assertion (i) is left to the reader. We concentrate on the proof of assertion (ii).

Suppose F is an atomic formula, that is, either $F = (t_1 = t_2)$ or $F = P(t_1, \ldots, t_n)$ for terms $t_1, t_2, \ldots, t_n$. In the first sub-case we have:

$$\begin{aligned}
\mathcal{I} \models t_1 = t_2 &\iff t_1^{\mathcal{I}} = t_2^{\mathcal{I}} && \\
&\iff m(t_1^{\mathcal{I}}) = m(t_2^{\mathcal{I}}) && m \text{ is injective} \\
&\iff t_1^{\mathcal{J}} = t_2^{\mathcal{J}} && \text{property (i)} \\
&\iff \mathcal{J} \models t_1 = t_2 \ ,
\end{aligned}$$

and in the second

$$\begin{aligned}
\mathcal{I} \models P(t_1, \ldots, t_n) &\iff (t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \in P^{\mathcal{A}} && \\
&\iff (m(t_1^{\mathcal{I}}), \ldots, m(t_n^{\mathcal{I}})) \in P^{\mathcal{B}} && \text{as } m\colon \mathcal{A} \cong \mathcal{B} \\
&\iff (t_1^{\mathcal{J}}, \ldots, t_n^{\mathcal{J}}) \in P^{\mathcal{B}} && \text{property (i)} \\
&\iff \mathcal{J} \models P(t_1, \ldots, t_n) \ .
\end{aligned}$$

Suppose F is a complex formula. The sub-cases where $F = \neg G$, $F = (G \wedge H)$ follow directly by the definition of the satisfaction relation $\models$ and the induction hypothesis. Hence, we assume $F(x) = \exists x G$.

$$\begin{aligned}
\mathcal{I} \models \exists x G &\iff \text{there exists } a \in A,\ \mathcal{I}\{x \mapsto a\} \models G && \\
&\iff \text{there exists } a \in A,\ \mathcal{J}\{x \mapsto m(a)\} \models G && \text{ind. hypothesis} \\
&\iff \text{there exists } b \in B,\ \mathcal{J}\{x \mapsto b\} \models G && m \text{ is surjective} \\
&\iff \mathcal{J} \models \exists x G
\end{aligned}$$

This concludes the proof. □

Corollary 3.1. *(*i*) Any set of formulas that has a finite model has a model in the domain* $\{0, 1, 2, \ldots, n\}$ *for some* n.

*(*ii*) Any set of formulas that has a countable infinite model has a model whose domain is the set of all natural numbers.*

Proof. Follows directly from Lemma 3.3 and Theorem 3.1. □

Problems

Problem 3.1. Indicate the form of the following argument, which is traditionally called 'syllogism in Felapton'.

(i) No centaurs are allowed to vote.

(ii) All centaurs are intelligent beings.

(iii) Therefore, some intelligent beings are not allowed to vote.

Problem 3.2. Let $\mathcal{L} = \{\mathsf{F}, \mathsf{P}, =\}$, where F is unary, P is binary and let $\mathcal{A}$ be a structure whose domain are sets of persons, such that $\mathsf{P}(a, b)$ denotes "a is parent of b" and F "female". Give informal explanations of the following formulas:

(i) $\exists z \exists u \exists v (u \neq v \land \mathsf{P}(u, b) \land \mathsf{P}(v, b) \land \mathsf{P}(u, z) \land \mathsf{P}(v, z) \land \mathsf{P}(z, a) \land \neg\mathsf{F}(b))$

(ii) $\exists z \exists u \exists v (u \neq v \land \mathsf{P}(u, a) \land \mathsf{P}(v, a) \land \mathsf{P}(u, z) \land \mathsf{P}(v, z) \land \mathsf{P}(z, b) \land \mathsf{F}(b))$

Problem 3.3. Consider the following sentences:

① Each smurf is happy if all its children are happy.

② Smurfs are green if at least two of their ancestors are green.

③ A smurf is really small if one of its parents is large.

④ Large smurfs are not really small.

⑤ There are red smurfs that are large.

For each of the sentences above, give a first-order formula that formalises it. Use the following constants, functions and predicates:

– constants: green, red.

– functions: colour(a).

– predicates: Smurf(a), Large(a), ReallySmall(a), Happy(a), Child(a, b) ("a is child of b"), Ancestor(a, b) ("a is ancestor of b"), $=$.

Problem 3.4. Show that the formalisation in the previous problem is satisfiable.

Problem 3.5. Let t be a term and let F be a formula.

– Give a formal definition of $\mathcal{V}\mathsf{ar}(t)$, the set of variables in t.

– Give a formal definition of $\mathcal{F}\mathcal{V}\mathsf{ar}(F)$, the set of free variables in F.

Problem 3.6. Show the following statements, either by reduction to definitions or by providing a counter-example:

(i) $\exists y \forall x \mathsf{P}(x, y) \models \forall x \exists y \mathsf{P}(x, y)$.

(ii) $\forall x \exists y \mathsf{R}(x, y) \not\models \exists y \forall x \mathsf{R}(x, y)$.

Problem 3.7. Define two formulas F and G, such that $F \not\models G$ holds and $F \not\models \neg G$ holds.

Problem 3.8. Give a formal proof of Lemma 3.2.
Hint: First prove (by induction) that the value of a term is the same with respect to $\mathcal{I}_1$ and $\mathcal{I}_2$. Based on this prove the lemma by structural induction.

Problem 3.9. Complete the proof of Theorem 3.1.

Problem 3.10. Let S be the set of satisfiable sets $\mathcal{G}$ of formulas and show the following properties, where $\mathcal{G} \in S$ is assumed.

(i) If $\mathcal{G}_0 \subseteq \mathcal{G}$, then $\mathcal{G}_0 \in S$.

(ii) If $\neg\neg F \in \mathcal{G}$, then $\mathcal{G} \cup \{F\} \in S$

(iii) If $(E \vee F) \in \mathcal{G}$, then either $\mathcal{G} \cup \{E\} \in S$ or $\mathcal{G} \cup \{F\} \in S$

(iv) If $\exists x F(x) \in \mathcal{G}$ and the individual constant c doesn't occur in $\mathcal{G}$ or $\exists x F(x)$, then $\mathcal{G} \cup \{F(c)\} \in S$

(v) If $\{F(s), s = t\} \subseteq \mathcal{G}$, then $\mathcal{G} \cup \{F(t)\} \in S$

4. Soundness and Completeness of First-Order Logic

In this chapter we show soundness and completeness of first-order logic. Furthermore, we prove the compactness theorem and the Löwenheim-Skolem theorem. These theorems are often proved as corollaries to the completeness theorem, cf. [27, 29, 20]. However, this has the disadvantage that the *proof* of these theorems depends on a formal system, while their *statement* does not. This is not elegant and comparable to the bad programming practice of first defining a clean interface of a data type and then ignoring the interface and altering private functions on the data type. Thus we give a direct proof of compactness and Löwenheim-Skolem. Based on this proof we conclude completeness essentially as a corollary.

In Section 4.1 we state the compactness theorem and Löwenheim-Skolem theorem together with direct corollaries. In Section 4.2 we prove the model existence theorem, which forms the core of the proof of compactness and Löwenheim-Skolem. Further, in Section 4.3 we recall the rules of natural deduction for first-order logic and proof completeness of first-order logic.

4.1. Compactness and Löwenheim-Skolem Theorem

We state the compactness theorem and Löwenheim-Skolem theorem together with direct corollaries. The proof of these theorems is given in Section 4.2 below.

Theorem 4.1 (Compactness Theorem)**.** *If every finite subset of a set of formulas $\mathcal{G}$ has a model, then $\mathcal{G}$ has a model.*

Theorem 4.2 (Löwenheim-Skolem Theorem)**.** *If a set of formulas $\mathcal{G}$ has a model, then $\mathcal{G}$ has a countable model.*

Corollary 4.1. *If a set of formulas $\mathcal{G}$ has arbitrarily large finite models, then it has a countable infinite model.*

Proof. Define an infinite set of sentences $(I_n)_{n\geqslant 1}$ as follows. (Note that the prefix of universal quantifiers is empty if $n = 1$).

$$I_n := \forall x_1 \dots \forall x_{n-1} \exists y \; (x_1 \neq y \wedge \cdots \wedge x_{n-1} \neq y)$$

Note that if $\mathcal{I} \models I_n$, then $\mathcal{I}$ has at least n elements. Consider

$$\mathcal{G}' := \mathcal{G} \cup \{I_1, I_2, \dots\} .$$

Any finite subset of $\mathcal{G}'$ is a subset of $\mathcal{G} \cup \bigcup_{1 \leqslant i \leqslant n} I_i$ for some n. By assumption that $\mathcal{G}$ has arbitrarily large finite models, this finite subset has a model. Due to compactness $\mathcal{G}'$ has a model, which is also an infinite model of $\mathcal{G}$. Finally, we employ Löwenheim-Skolem to conclude that this model is countable. ☐

Further we obtain the following strengthening of Corollary 3.1.

Corollary 4.2. (i) *Any set of formulas $\mathcal{G}$ that has a model, has a model whose domain is either the set of natural numbers $< n$ for some positive number n, or else the set of all numbers.*

(ii) *Suppose a set of formulas $\mathcal{G}$, whose language $\mathcal{L}$ is based on individual and predicate constants only and such that $\mathcal{L}$ doesn't contain $=$. If $\mathcal{G}$ has a model, then $\mathcal{G}$ has a model whose domain is the set of all natural numbers.*

Proof. It suffices to prove the second assertion, the first follows from Corollary 3.1 and Löwenheim-Skolem. Consider $\mathcal{G}$: due to the first part $\mathcal{G}$ has either a model $\mathcal{I}$ whose domain is the set of all numbers, or a model $\mathcal{I}$ whose domain is $\{0, 1, \dots, n-1\}$ for $n \in \mathbb{N}$. We assume the latter and we assume that the environment of $\mathcal{I}$ is denoted as ℓ. Let $f \colon \mathbb{N} \to \{0, 1, \dots, n-1\}$ be defined as follows:

$$f(m) := \min\{m, n-1\} .$$

Then clearly f is surjective. We define an interpretation $\mathcal{J}$ with environment ℓ^f induced by f. (Compare the proof of Theorem 3.1.) For any individual constant c, we set $c^{\mathcal{J}}$ such that $f(c^{\mathcal{J}}) = c^{\mathcal{I}}$ and for any numbers $n_1, \dots, n_k$ and k-ary predicate constant P we set $(n_1, \dots, n_k) \in P^{\mathcal{J}}$ iff $(f(n_1), \dots, f(n_k)) \in P^{\mathcal{I}}$. Note that this definition would not be well-defined if extended in the same way to function symbols.

Now f is almost an isomorphism, but it is not injective. Inspection of the proof of Theorem 3.1 shows that injectivity is only necessary when equality is present. Hence we obtain for all formulas F: $\mathcal{I} \models F$ iff $\mathcal{J} \models F$. In sum we obtain $\mathcal{J} \models \mathcal{G}$, as $\mathcal{I} \models \mathcal{G}$. Further, the domain of $\mathcal{J}$ are the set of all natural numbers, which concludes the proof. ☐

4.2. Model Existence Theorem

Recall Theorem 4.1.

Theorem (Compactness Theorem)**.** *If every finite subset of a set of formulas $\mathcal{G}$ has a model, then $\mathcal{G}$ has a model.*

In proof we assume that the only propositional connectives used are $\neg$ and $\vee$. The only quantifier occurring in a formula is $\exists$. This simplifies the number of cases we need to consider. Let S be the set of satisfiable formulas sets. The next lemma consolidates certain properties of S to be exploited later on.

Lemma 4.1. *Let S be the set of satisfiable sets of formulas $\mathcal{G}$ and let $\mathcal{G} \in S$. Then we have:*

(i) *If $\mathcal{G}_0 \subseteq \mathcal{G}$, then $\mathcal{G}_0 \in S$.*

(ii) *For no formula F, both F and $\neg F$ are in $\mathcal{G}$.*

(iii) *If $\neg\neg F \in \mathcal{G}$, then $\mathcal{G} \cup \{F\} \in S$.*

(iv) *If $(E \vee F) \in \mathcal{G}$, then $\mathcal{G} \cup \{E\} \in S$ or $\mathcal{G} \cup \{F\} \in S$.*

(v) *If $\neg(E \vee F) \in \mathcal{G}$, then $\mathcal{G} \cup \{\neg E\} \in S$ and $\mathcal{G} \cup \{\neg F\} \in S$.*

(vi) *If $\exists x F(x) \in \mathcal{G}$, the individual constant c doesn't occur in $\mathcal{G}$ or $\exists x F(x)$, then $\mathcal{G} \cup \{F(c)\} \in S$.*

(vii) *If $\neg\exists x F(x) \in \mathcal{G}$, then for all terms t, $\mathcal{G} \cup \{\neg F(t)\} \in S$.*

(viii) *For any term t, $\mathcal{G} \cup \{t = t\} \in S$.*

(ix) *If $\{F(s), s = t\} \subseteq \mathcal{G}$, then $\mathcal{G} \cup \{F(t)\} \in S$.*

Proof. The proof of all 9 properties follows directly form the definition of satisfiability. Compare also Problem 3.10. ☐

Definition 4.1. The 9 properties in Lemma 4.1 are called *satisfaction properties.*

The proof of Theorem 4.1 is concluded if we can argue that the considered formula set $\mathcal{G}$ belongs to S as defined above. In order to do so, we express the assumption of this theorem as another set: Let S^* denote the set of all formula sets whose *finite* subsets belong to S.

Lemma 4.2. *If S is a set of sets of formulas having the satisfaction properties, then the set S^* of all sets of formulas whose every finite subset is in S has the satisfaction properties.*

Proof. In proof one performs case distinction over all 9 properties to verify that S^* indeed admits the satisfaction properties. We consider the only interesting case: disjunction.

Suppose $\mathcal{G} \cup \{E \vee F\} \in S^*$. By definition, for every finite subset $\mathcal{G}'$ of $\mathcal{G} \cup \{E \vee F\}$, $\mathcal{G}' \in S$. We have to prove that either every finite subset of $\mathcal{G} \cup \{E\}$ is in S or every finite subset of $\mathcal{G} \cup \{F\}$ is in S, as this would imply that either $\mathcal{G} \cup \{E\} \in S^*$ or $\mathcal{G} \cup \{F\} \in S^*$.

Assume there exists a finite subset $\mathcal{G}_0 \subseteq \mathcal{G} \cup \{E\}$ such that $\mathcal{G}_0 \notin S$. Clearly $\mathcal{G}_0 \not\subseteq \mathcal{G}$ as otherwise $\mathcal{G}_0 \in S$ follows from $\mathcal{G}_0 \subseteq \mathcal{G} \cup \{E \vee F\}$, $\mathcal{G} \cup \{E \vee F\} \in S^*$, and the definition of S^*. Thus we assume there exists a finite set $\mathcal{G}_1 \subseteq \mathcal{G}$ such that $\mathcal{G}_0 = \mathcal{G}_1 \cup \{E\}$.

We claim that for any finite subset $\mathcal{G}_2 \subseteq \mathcal{G} \cup \{F\}$, $\mathcal{G}_2 \in S$. In proof of this claim observe that the assumption $\mathcal{G}_2 \subseteq \mathcal{G}$ immediately implies that $\mathcal{G}_2 \in S$. Thus we assume without loss of generality that there exists a finite set $\mathcal{G}_3 \subseteq \mathcal{G}$ and $\mathcal{G}_2 = \mathcal{G}_3 \cup \{F\}$. Consider $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{E \vee F\}$. By assumption this is a finite subset of $\mathcal{G} \cup \{E \vee F\}$. Hence $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{E \vee F\} \in S$ and thus either $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{E\} \in S$ or $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{F\} \in S$. If $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{E\} \in S$, then observe:

$$\mathcal{G}_0 = \mathcal{G}_1 \cup \{E\} \subseteq \mathcal{G}_1 \cup \mathcal{G}_3 \cup \{E\} \in S \ ,$$

which would imply $\mathcal{G}_0 \in S$, contrary to our assumption. Thus $\mathcal{G}_1 \cup \mathcal{G}_3 \cup \{F\} \in S$. And also $\mathcal{G}_2 = \mathcal{G}_3 \cup \{F\} \in S$. This completes the proof of this case. □

Let $\mathcal{L}$ be a language, let $\mathcal{L}^+$ be an extension of $\mathcal{L}$ with infinitely many individual constants. In the sequel we will prove the following theorem.

Theorem 4.3 (Model Existence Theorem). *(i) If S^* is a set of sets of formulas of $\mathcal{L}^+$ having the satisfaction properties, then every set of formulas of $\mathcal{L}$ in S^* has a model $\mathcal{M}$.*

(ii) Every element of the domain of $\mathcal{M}$ is the denotation of some term in $\mathcal{L}^+$.

We momentarily assume Theorem 4.3. Based on this theorem, we conclude compactness and Löwenheim-Skolem.

Theorem (Compactness Theorem). *If every finite subset of a set of formulas $\mathcal{G}$ has a model, then $\mathcal{G}$ has a model.*

Proof. Let S denote the set of satisfiable sets of formulas (over $\mathcal{L}$) and let S^* denote the set of all sets of formulas (over $\mathcal{L}^+$) whose every finite subset belongs to S. By Lemma 4.1 S admits the satisfaction properties. This together with Lemma 4.2 yields that S^* admits the satisfaction properties. Hence (due to Theorem 4.3) every formula set in S^* has a model. Consider the set $\mathcal{G}$ assumed in the theorem. Then every finite subset of $\mathcal{G}$ is satisfiable, hence $\mathcal{G} \in S^*$ and thus $\mathcal{G}$ is satisfiable. □

Theorem (Löwenheim-Skolem Theorem). *If a set of formulas $\mathcal{G}$ has a model, then $\mathcal{G}$ has a countable model.*

Proof. Let S denote the set of satisfiable sets of formulas (over $\mathcal{L}$). By Theorem 4.3 every set of formulas in S has a model $\mathcal{M}$ in which each element of the domain is the denotation of some term in $\mathcal{L}^+$. The language $\mathcal{L}$ is countable, thus the extended language $\mathcal{L}^+$ is countable and there are at most countably many terms in $\mathcal{L}^+$. As every element of (the domain of) $\mathcal{M}$ is the denotation of a term and a term can be the denotation of at most one element of $\mathcal{M}$, $\mathcal{M}$ is countable. □

In proof of Theorem 4.3 we consider properties of formulas which are modelled by some interpretation $\mathcal{M}$ (of $\mathcal{L}^+$).

Lemma 4.3. *Let $\mathcal{G}$ denote the set of formulas true in $\mathcal{M}$. Then we have:*

(i) for no formula F and $\neg F$ in $\mathcal{G}$,

(ii) if $\neg\neg F \in \mathcal{G}$, then $F \in \mathcal{G}$,

(iii) if $(E \vee F) \in \mathcal{G}$, then either $E \in \mathcal{G}$ or $F \in \mathcal{G}$,

(iv) *if $\neg(E \lor F) \in \mathcal{G}$, then $\neg E \in \mathcal{G}$ and $\neg F \in \mathcal{G}$,*

(v) *if $\exists x F(x) \in \mathcal{G}$, then there exists a term t (of $\mathcal{L}^+$), such that $F(t) \in \mathcal{G}$,*

(vi) *if $\neg\exists x F(x) \in \mathcal{G}$, then for any term t (of $\mathcal{L}^+$), $\neg F(t) \in \mathcal{G}$,*

(vii) *for any term t (of $\mathcal{L}^+$), $t = t \in \mathcal{G}$, and*

(viii) *if $F(s) \in \mathcal{G}$, and $s = t \in \mathcal{G}$, then $F(t) \in \mathcal{G}$.*

Proof. The 8 properties follow as $\mathcal{M} \models \mathcal{G}$. □

Definition 4.2. The 8 properties in Lemma 4.3 are called *closure properties.*

In addition to Lemma 4.3 we have its converse. For the moment we restrict our attention to the case where the base language $\mathcal{L}$ is restricted. It is not difficult to see that this restriction can be easily lifted.

Lemma 4.4. *Let $\mathcal{G}$ be a set of formulas (of $\mathcal{L}$) admitting the closure properties. Suppose that $\mathcal{L}$ is free of the equality symbol and free of function constants. Then there exists a model $\mathcal{M}$ such that every element of the domain of $\mathcal{M}$ is the denotation of a term (of $\mathcal{L}^+$) and $\mathcal{M} \models \mathcal{G}$.*

Proof. Let $\mathcal{G}$ be set of formulas. We construct a model of $\mathcal{G}$. We define the domain of $\mathcal{M}$ as the set of all terms in $\mathcal{L}^+$. Thus, due to our restriction on $\mathcal{L}^+$, the domain of $\mathcal{M}$ consists of infinitely many individual constants and variables. In order to define the structure underlying $\mathcal{M}$, we set:

$$c^{\mathcal{M}} := c \text{ for any individual constant } c\ .$$

In order to guarantee that $\mathcal{M} \models \mathcal{G}$ it suffices to make all atomic formulas occurring in $\mathcal{G}$ true. For that we set for any predicate constant P and for any sequence of terms $t_1, \ldots, t_n$:

$$P^{\mathcal{M}}(t_1, \ldots, t_n) \Longleftrightarrow P(t_1, \ldots, t_n) \in \mathcal{G}\ .$$

Finally, we lift this structure to an interpretation $\mathcal{M}$ by defining the look-up table as follows:

$$\ell(x) := x \qquad \text{for any variable } x\ .$$

This completes the definition of the interpretation $\mathcal{M}$. Note that each term of $\mathcal{L}^+$ is interpreted by itself, that is, we have:

$$t^{\mathcal{M}} = t \qquad \text{for any term } t\ .$$

The definition of $\mathcal{M}$ takes care of the demand that every element of its domain is the denotation of a term. Any term t in $\mathcal{L}^+$ is denoted by an element of $\mathcal{M}$, namely the domain element t.

It remains to prove that for any formula F: $F \in \mathcal{G}$ implies $\mathcal{M} \models F$. This we proof by induction on F.

– For the base case $F = P(t_1, \ldots, t_n)$, we obtain: if $F \in \mathcal{G}$, then by definition $P^{\mathcal{M}}(t_1, \ldots, t_n)$, hence $\mathcal{M} \models F$.

- For the step case assume $F = \exists x G(x)$ and $F \in \mathcal{G}$. By induction hypothesis for any term t such that $G(t) \in \mathcal{G}$, we have $\mathcal{M} \models G(t)$. Now, by assumption $\mathcal{G}$ fulfils the closure properties, hence there exists a term t such that $G(t) \in \mathcal{G}$. Thus $\mathcal{M} \models \exists x G(x)$ holds by definition of the satisfaction relation $\models$. The other cases follow similarly.

□

Remark 4.1. In Chapter 6 we will study models $\mathcal{M}$ that feature the equation $t^{\mathcal{M}} = t$ (for any term t) in more detail. Such models are often called *term* or *Herbrand* models.

Lemma 4.5. *Let $\mathcal{L}$ be a language and let $\mathcal{L}^+$ denote an extension of $\mathcal{L}$ with infinitely many individual constants. Suppose S^* is a set of set of formulas (of $\mathcal{L}^+$) with the satisfaction properties. Then every set $\mathcal{G}$ of formulas (over $\mathcal{L}$) in S^* is extensible to a set $\mathcal{G}^*$ of formulas (over $\mathcal{L}^+$) having the closure property.*

Proof. We construct a sequence of sets, starting with $\mathcal{G}$ such that only elements are added, never removed:

$$\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2 \ldots \qquad \mathcal{G}_n \subseteq \mathcal{G}_{n+1} ,$$

Moreover, we demand that any $\mathcal{G}_i$ belongs to S^*. Finally we define $\mathcal{G}^* := \bigcup_{n \geqslant 0} \mathcal{G}_n$.

We have to verify that $\mathcal{G}^*$ admits all 8 closure properties. The first one is trivial. Assume that for a given formula F, F and $\neg F$ occurs in $\mathcal{G}^*$. As $\mathcal{G}^*$ is the union of the above sequence there exists an index k such that $\{F, \neg F\} \subseteq \mathcal{G}_k$, but if we can guarantee that $\mathcal{G}_k \in S^*$, then this contradicts the fact that S^* admits the satisfaction properties. Thus we only need to consider the other 7 properties and make sure that for each n, $\mathcal{G}_n \in S^*$.

At each stage n we aim to add only one formula to $\mathcal{G}_n$. The remaining 7 closure properties define certain demands on $\mathcal{G}^*$ that can be formulated as follows.

(i) if $\neg\neg F \in \mathcal{G}_n$, then there exists $k \geqslant n$ such that $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{F\}$,

(ii) if $(E \vee F) \in \mathcal{G}_n$, then there exists $k \geqslant n$ such that $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{E\}$ or $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{F\}$,

(iii) if $\neg(E \vee F) \in \mathcal{G}_n$, then there exists $k \geqslant n$ such that $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{\neg E\}$ and $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{\neg F\}$,

(iv) if $\exists x F(x) \in \mathcal{G}_n$, then there exists $k \geqslant n$ such that there is a term t and $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{F(t)\}$, and

(v) if $\neg\exists x F(x) \in \mathcal{G}_n$, then for any term t there exists $k \geqslant n$, such that $\mathcal{G}_{k+1} = \mathcal{G}_k \cup \{\neg F(t)\}$,

(vi) for any term t, then there exists $k \geqslant n$ such that $t = t \in \mathcal{G}_k$, and

(vii) if $F(s) \in \mathcal{G}_n$, and $s = t \in \mathcal{G}_n$, then there exists $k \geqslant n$ such that $F(t) \in \mathcal{G}_k$.

In meeting these demands we use the fact that all previously constructed $\mathcal{G}_n$ are contained in S^* and that S^* admits the satisfaction properties. Hence, we can use the following facts. Note that we employ the fact that the sequence $(\mathcal{G}_n)_{n\geqslant 0}$ is growing: $\mathcal{G}_n \subseteq \mathcal{G}_{n+1}$.

(i) If $\neg\neg F \in \mathcal{G}_n$, then for any $k \geqslant n$, $\mathcal{G}_k \cup \{F\} \in S^*$.

(ii) If $(E \vee F) \in \mathcal{G}_n$, then for any $k \geqslant n$, either $\mathcal{G}_k \cup \{E\} \in S^*$ or $\mathcal{G}_k \cup \{F\} \in S^*$.

(iii) If $\neg(E \vee F) \in \mathcal{G}_n$, then for any $k \geqslant n$, $\mathcal{G}_k \cup \{\neg E\} \in S^*$ and $\mathcal{G}_k \cup \{\neg F\} \in S^*$.

(iv) If $\exists x F(x) \in \mathcal{G}$, then for any $k \geqslant n$ and any unused individual constant c, $\mathcal{G}_k \cup \{F(c)\} \in S^*$.

(v) If $\neg\exists x F(x) \in \mathcal{G}$, then for any $k \geqslant n$ and for any term t, $\mathcal{G}_k \cup \{\neg F(t)\} \in S^*$.

(vi) For any term t, for any $k \geqslant n$, $\mathcal{G}_k \cup \{t = t\} \in S^*$.

(vii) If $\{F(s), s = t\} \subseteq \mathcal{G}_n$, then for any $k \geqslant n$, $\mathcal{G}_k \cup \{F(t)\} \in S^*$.

The correspondence between demand and properties induced by the fact that S^* fulfils the satisfaction properties shows that we can in principle fulfil any demand. It only remains to define a fair strategy such that eventually any of the infinite demands is fulfilled.

However this is easy if we recall that any pair (i, n) can be encoded as a single natural number. Associate to each demand a pair (i, n) such that i is the number of the demand raised at stage n. Hence it remains to enumerate all pairs (i, n) so that at a given stage k we decode the pair k represents and grant the i^{th} demand that was raised at stage $n < k$. In this way it is guaranteed that all demands above (except Demand (vi)) can be eventually satisfied such that all constructed sets $\mathcal{G}_n$ belong to S^*. In order to meet the 6th demand, we encode the triple $(i, n, \ulcorner t \urcorner)$, where $\ulcorner t \urcorner$ denotes the Gödel number of the term t. Then we proceed as before. □

Based on Lemmas 4.4 and 4.5 we can prove the model existence theorem. We recall the theorem:

Theorem. *(i) If S^* is a set of sets of formulas of $\mathcal{L}^+$ having the satisfaction properties, then every set of formulas of $\mathcal{L}$ in S^* has a model $\mathcal{M}$.*

(ii) Every element of the domain of $\mathcal{M}$ is the denotation of some term in $\mathcal{L}^+$.

Proof. In proof of the theorem we restrict our base language $\mathcal{L}$ to the case where $\mathcal{L}$ is free of function constants and equality, cf. Lemma 4.4.

By assumption S^* admits the satisfaction properties. Due to Lemma 4.5 we have that for any formula set $\mathcal{G}$ (over $\mathcal{L}$) in S^* is extensible to a set $\mathcal{G}^*$ of formulas (of $\mathcal{L}^+$) such that $\mathcal{G}^*$ fulfils the closure properties. But then Lemma 4.4 is applicable to $\mathcal{G}^*$ and we obtain a $\mathcal{M}$ such that $\mathcal{M} \models \mathcal{G}^*$. This takes care of the first statement of the lemma.

Moreover $\mathcal{M}$ has the property that any element in the universe of $\mathcal{M}$ is the denotation of a term (of $\mathcal{L}^+$). This takes care of the second statement of the lemma. □

4.3. Soundness and Completeness

In this section we prove soundness and completeness of predicate logic. The propositional rules (for the connectives $\neg$, $\vee$, $\wedge$, and $\rightarrow$) are given in Figure 2.1 in Chapter 2. We only need to lift (or conceive) these rules in the present context, the context of first-order logic. The rules for equality are given in Figure 4.1 and quantifier rules for $\exists$ and $\forall$ are given in Figure 4.2.

	introduction	*elimination*
$=$	$\dfrac{}{t = t}\ =\colon \mathsf{i}$	$\dfrac{s = t \quad F(s)}{F(t)}\ =\colon \mathsf{e}$

Figure 4.1.: Natural Deduction: Equality Rules

	introduction	*elimination*
$\exists$	$\dfrac{F(t)}{\exists x F(x)}\ \exists\colon \mathsf{i}$	$\dfrac{\exists x F(x) \quad \boxed{\begin{array}{cc} a & F(a) \\ & \vdots \\ & G \end{array}}}{G}\ \exists\colon \mathsf{e}$
$\forall$	$\dfrac{\boxed{\begin{array}{cr} & a \\ \vdots & \\ F(a) & \end{array}}}{\forall x F(x)}\ \forall\colon \mathsf{i}$	$\dfrac{\forall x F(x)}{F(t)}\ \forall\colon \mathsf{e}$

Here the variable a in $\exists\colon \mathsf{e}$ and in $\forall\colon \mathsf{i}$ is local to the box it occurs in.

Figure 4.2.: Natural Deduction: Quantifier Rules

Let $\mathcal{G}$ be a finite set of formulas and let F be a formula. A *natural deduction proof* is a sequence of applications of rules depicted in Figure 2.1, 4.1, and 4.2. We adapt the definition of provability given above with respect to propositional logic.

Definition 4.3. The *provability relation*, denoted as $A_1, \dots, A_n \vdash B$, asserts that B is derived from the assumptions $A_1, \dots, A_n$. This notion extends to infinite set of formulas $\mathcal{G}$: We write $\mathcal{G} \vdash F$ if there exists a finite subset $\mathcal{G}' \subseteq \mathcal{G}$ such that $\mathcal{G}' \vdash F$. We write $\vdash A$ instead of $\varnothing \vdash A$ and call the formula A *provable* in this case.

Theorem 4.4 (Soundness Theorem)**.** *Let $\mathcal{G}$ be a set of formulas and let F be a formula such that $\mathcal{G} \vdash F$. Then $\mathcal{G} \models F$.*

Sketch of Proof. We only sketch the proof. For a slightly different formal system a completely worked out proof can be found in [11].

In proof of soundness one verifies that every single inference rule is correct. For this one shows that if the assumptions of an inference rule are modelled by a model $\mathcal{M}$, then the consequence (of the rule) holds in $\mathcal{M}$ as well. $\square$

In order to prepare for the completeness theorem, we state two lemmas, whose proof is left to the reader (compare also [11]). Recall that a set of formulas $\mathcal{G}$ is called inconsistent if $\bot$ is derivable from $\mathcal{G}$.

Lemma 4.6. *We have $\mathcal{G} \vdash F$ iff $\mathcal{G} \cup \{\neg F\}$ is inconsistent.*

Lemma 4.7. *The set S of all consistent sets of formulas has the satisfaction properties.*

Theorem 4.5 (Completeness Theorem)**.** *Let $\mathcal{G}$ be set of formulas and let F be a formula such that $\mathcal{G} \models F$. Then $\mathcal{G} \vdash F$.*

Proof. By compactness we know that there exists a finite subset $\mathcal{G}'$ of $\mathcal{G}$, such that $\mathcal{G}' \models F$. Hence we can assume without loss of generality that the formula set $\mathcal{G}$ is finite.

Thus in order to show completeness, we have to show that $\mathcal{G} \vdash F$ holds. We show the contra-positive. Suppose F is *not* derivable form $\mathcal{G}$, then F is *not* a consequence of $\mathcal{G}$. Due to Lemmas 4.6, $\mathcal{G} \nvdash F$ is equivalent to the assertion that $\mathcal{G} \cup \{\neg F\}$ is consistent. On the other hand, due to Lemma 3.1 $\mathcal{G} \not\models F$ is equivalent to the assertion that the set $\mathcal{G} \cup \{\neg F\}$ is satisfiable.

Hence, we have to prove that the consistency of $\mathcal{G} \cup \{\neg F\}$ implies that the set $\mathcal{G} \cup \{\neg F\}$ is satisfiable. Thus it suffices to show that any consistent set is satisfiable.

By the model existence theorem (Theorem 4.3) it suffices to verify that the set S of consistent sets of formulas has the satisfaction properties. As the latter follows by Lemma 4.7 we conclude completeness. $\square$

4.4. Normalisation

We conclude this chapter by briefly looking into the very important topic of proof normalisation. While of restricted importance in establishing completeness of natural deduction, normalisation of deductions becomes a major topic, if we study derivations from the viewpoint of proof theory. In particular normalisation yields the *consistency* of natural deduction by purely syntactic arguments [24, 54, 45, 49]. Another application that we will come across in the context of the Curry-Howard isomorphism, is the correspondence between normalisation steps in natural deduction for intuitionistic logic (or more precisely minimal logic) and β-reduction in the simple-typed λ-calculus, see Chapter 7.

In a natural deduction it may happen that an introduction rule is immediately followed by an elimination rule, like for example in the following scenario:

$$\dfrac{\dfrac{\begin{matrix}\Pi_1 \\ E\end{matrix} \quad \begin{matrix}\Pi_2 \\ F\end{matrix}}{E \wedge F}\,\wedge\colon \mathsf{i}}{E}\,\wedge\colon \mathsf{e} \tag{4.1}$$

Clearly this derivation can be simplified and replaced by the following derivation, where the proof of Π_2 has been deleted.

$$\begin{matrix} \Pi_1 \\ E \end{matrix} \tag{4.2}$$

In the literature a situation as in (4.1) has been called *detour*. Intuitively such a detour should be prevented, thus we want to rewrite a derivation by replacing the fragment in (4.1) by the fragment in (4.2).

The process of eliminating all detours from a given proof is called *normalisation*. One can prove that normalisation terminates for any reduction sequence terminates (*strong normalisation*). This result is well-known for (propositional) intutitionistic logic, but it also holds for richer logics. In particular Gentzen showed (in a long forgotten draft) normalisation for first-order intuitionistic logic [54]. Essentially the same proof was found by Prawitz, who also extended the normalisation theorem to classical logic (for a restricted set of connectives) [45]. Finally (among others) Stålmarck showed the same result for first-order classical logic (over the usual language) [49].

To simplify the presentation, we restrict our attention to a fragment of propositional logic. More precisely, we study normalisation in the context of *minimal logic*. The language of minimal logic contains $\perp$ as truth constant (for falsity) and $\wedge$, $\vee$, $\rightarrow$ as binary connectives. Negation is defined as follows $\neg A := A \rightarrow \perp$. Minimal logic is a restriction of intuitionistic propositional logic (see Chapter 7) and classical propositional logic. In order to obtain intuitionistic logic it suffices to add the rule

$$\frac{\perp}{F}\ \neg\colon \mathtt{e} \quad .$$

Alternatively, one can include $\neg$ again as logical symbol and add the usual rules for introduction and eliminatin of $\neg$. To obtain classical logic it suffices to add the rule

$$\frac{\neg\neg F}{F}\ \neg\neg\colon \mathtt{e} \quad .$$

to minimal logic. Figure 4.3 makes the notion of reduction precise for the connectives of minimal logic.

Let Π be a proof and Ψ be the result of applying one of the normalisations steps given in Figure 4.3 to Ψ. Then we say that Π is *immediately reduced* to Ψ. A sequence of immediate reduction steps is called a *reduction*. A derivation is said to be *normal*, if it has no immediate reduction. In other words in a normal derivations there are no detours (as given in Figure 4.3). A *reduction sequence* is a sequence of derivations $\Pi_1, \ldots, \Pi_n$, such that for all $i = 1, \ldots, n_1$, Π_{i+1} is an immediate reduct of Π_i and Π_n is normal.

The next theorem states that any proof can be normalised. Furthermore in normalisation we need not take care of order of reduction sequences.

Theorem 4.6 (Normalisation and Strong Normalisation)**.** *Let* Π *be a proof in minimal logic.*

*(*i*)* Π *reduces to a normal proof* Ψ*.*

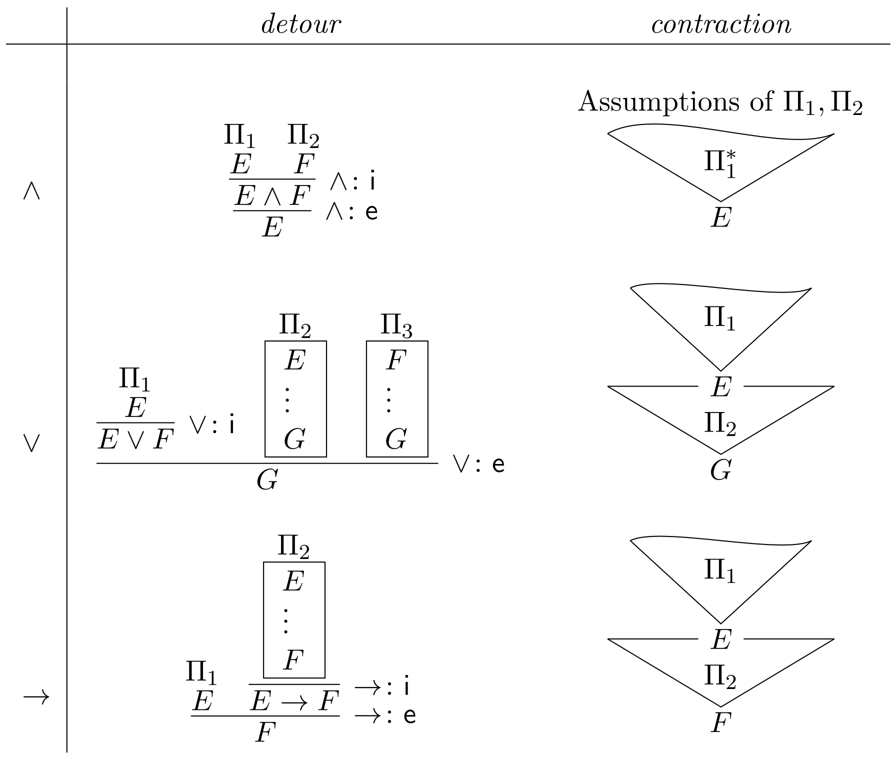

Figure 4.3.: Immediate Reductions

(ii) *Any reduction sequence is finite.*

The first part of the theorem can be expressed as saying that there exists a reduction sequence for Π. This is usally called (weak) normalisation. The second part expresses there is an upper bound n on the maximal length of any reduction sequence.

Problems

Problem 4.1. Let $\mathcal{L}_{\mathsf{arith}}$ contain $=$ and the constants $\mathsf{0}$, s, $+$, $\cdot$, $<$. By *true arithmetic* we mean the set of sentences $\mathcal{G}$ of $\mathcal{L}_{\mathsf{arith}}$ that are true in the usual interpretation in number theory.

By a *non-standard* model of arithmetic we mean a model of $\mathcal{G}$ that is not isomorphic to the standard interpretation. Let $\mathcal{H} = \mathcal{G} \cup \{\mathsf{c} \neq 0, \mathsf{c} \neq 1, \dots\}$, where c denotes a constant not in $\mathcal{L}_{\mathsf{arith}}$. Prove that any model of $\mathcal{H}$ is a non-standard model.

Problem 4.2. Complete the proof of Theorem 4.4.

Problem 4.3. Prove Lemma 4.6.

Problem 4.4. Prove Lemma 4.7.

Problem 4.5. Let $\mathcal{G}$ denote an interpretation that models a directed graph G. Show that reachability is not expressible in first-order logic: there exists no formula $F(x,y)$ whose only free variables are x and y, such that $\mathcal{G} \models F(x,y)$ iff $\ell(y)$ is reachable from $\ell(x)$ in G.

Problem 4.6. Consider classical propositional logic using only the connectives $\bot$, $\wedge$, $\rightarrow$, such that negation and disjunction is defined as usual. Prove that for any derivation there exists a normal derivation.

Hint: Prawitz proved the normalisation theorem for the first-order extension of this logic [45] for a variant of the usual natural deduction rules. One keeps only the introduction and elimination rules for $\wedge$, $\rightarrow$, but adds the following *proof by contradiction (PBC)* rule:

$$\dfrac{\boxed{\begin{array}{c}\neg E\\ \vdots\\ \bot\end{array}}}{E}\;\bot\;.$$

It is easy to see that this calculus is equivalent to the standard natural deduction for classical propositional logic.

Problem 4.7. Prove the following formulas with *normal* derivations:

a) $A \rightarrow B \rightarrow A$

b) $(A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$

c) $(A \wedge B \rightarrow C) \rightarrow (A \rightarrow B \rightarrow C)$

5. Craig's Interpolation Theorem

Given an implication $A \to C$, Craig's interpolation theorem tells us that there exists a sentence B, the *interpolant*, such that B is implied by A and B implies C. Moreover B employs only non-logical constants that occur in both A and C. After presenting the theorem in some detail, we will employ it to prove Robinson's joint consistency theorem, a theorem that allows us to speak about the satisfiability of the union $S \cup T$ of theories S and T, based only on the satisfiability of S and T. The latter theorem is partly related to the Nelson-Oppen method briefly mentioned in Chapter 9.

5.1. Craig's Theorem

Recall Theorem 2.3 that stated the existence of interpolants for valid implications in the context of propositional logic. We extend this result to first-order logic.

We start with the following simple lemma, whose proof is left to the reader.

Lemma 5.1. *If the sentence $A \to C$ holds, there exists a sentence B such that $A \to B$ and $B \to C$ and only those individual constants occur in B that occur in both A and C.*

If we attempt to generalise the lemma such that B contains only individual, function, and predicate constants that occur in both A and C, some care is necessary.

Example 5.1. Let $A :\Longleftrightarrow \exists x F(x) \land \exists x \neg F(x)$ and let $C :\Longleftrightarrow \exists x \exists y (x \neq y)$. Then $A \to C$ holds, but there exists no interpolant B such that only individual, function, or predicate constants occur in B that are shared by A and C.

Theorem 5.1. *If the sentence $A \to C$ holds, there exists a sentence B such that $A \to B$ and $B \to C$ such that only those non-logical constants occur in B that occur in both A and C.*

Note that the example above doesn't contradict the theorem as we consider the equality sign as *logical* symbol, compare Section 3.1. Before proving this theorem we deal with two degenerated cases. Suppose $A \to C$ holds and A is unsatisfiable. Then any unsatisfiable sentence can be used as interpolant that only uses non-logical constants that occur in A and C. Consider for example

$$\exists x(F(x) \land \neg F(x)) \to \exists G(x) \ .$$

Then $\exists x(x \neq x)$ serves as interpolant: Clearly $\exists x(F(x) \land \neg F(x)) \to \exists x(x \neq x)$ and $\exists x(x \neq x) \to \exists G(x)$. The dual case occurs if C is valid. Then any valid sentence serves as interpolant if the condition on non-logical constants is fulfilled. As a side-remark observe that for languages *without* the equality sign $=$ Craig's interpolation theorem for these degenerated cases holds only true if we extend the language by logical constants like $\top$ and $\bot$.

We are ready to give the proof of the theorem.

Proof. From the above discussion it is clear that we can restrict to those implications $A \to C$, where neither A is unsatisfiable nor C is valid. Moreover we will only treat the special case where equality and individual and function constants are absent. The general case follows from the special case by the use of the pattern of the proofs of Lemma 6.1 and 6.2.

In proof we proceed indirectly and assume that no interpolant exists. Then we show that the set of sentences $\{A, \neg C\}$ is satisfiable, which contradicts the assumption that A implies C. The general proof plan is as follows. We make use of the model existence theorem. Thus we consider a language $\mathcal{L}$ that contains all the non-logical symbols occurring in both A and C and its extension $\mathcal{L}^+$ containing infinitely many individual constants. Then we define a collection S of sets of sentences such that $\{A, \neg C\} \in S$ and S fulfils the satisfaction properties, cf. Definition 4.1. Then Theorem 4.3 is applicable to yield that any set of formulas of $\mathcal{L}$ in S has a model and thus $\{A, \neg C\}$ is satisfiable.

First, we define the collection S. We call a set of sentences $\mathcal{G}$ (of $\mathcal{L}^+$) *A-sentences* (*C-sentences*) if all sentences in $\mathcal{G}$ contain only predicate constants that occur in A (C). A pair of set of sentences $(\mathcal{G}_1, \mathcal{G}_2)$ such that $\mathcal{G}_1$ are satisfiable A-sentences and $\mathcal{G}_2$ are satisfiable C-sentences is *barred* by a sentences B, if B is both an A-sentence and a C-sentence and $\mathcal{G}_1 \models B$ and $\mathcal{G}_2 \models \neg B$ holds. Note that the assumption that there exists no interpolant B (of $\mathcal{L}$) is equivalent to say that no sentences bars $(A, \neg C)$. Moreover no sentence of $\mathcal{L}^+$ can bar $(A, \neg C)$ if this assumption holds. (This follows by similar argument as used in the proof of Lemma 5.1.) Let S be the collection of set of sentences $\mathcal{G}$ that admit an *unbarred division*, that is, there exists a pair $(\mathcal{G}_1, \mathcal{G}_2)$ of A-sentences and C-sentences such that $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$, $\mathcal{G}_1$ and $\mathcal{G}_2$ are satisfiable and no sentence bars $\mathcal{G}_1, \mathcal{G}_2$. This concludes the definition of S.

Next, we verify that S admits the satisfaction properties. We consider the only interesting case.

– Let $\mathcal{G} \in S$. If $(E \lor F) \in \mathcal{G}$, then either $\mathcal{G} \cup \{E\} \in S$ or $\mathcal{G} \cup \{F\} \in S$.

As $\mathcal{G} \in S$ there exists a pair $(\mathcal{G}_1, \mathcal{G}_2)$ such that $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$ and $(\mathcal{G}_1, \mathcal{G}_2)$ is unbarred. Without loss of generality assume $(E \lor F) \in \mathcal{G}_1$. Then both E and F are A-sentences. It suffices to show that either $(\mathcal{G}_1 \cup \{E\}, \mathcal{G}_2)$ or $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$ forms an unbarred division of $\mathcal{G} \cup \{E\}$. In proof, first observe that if $\mathcal{G}_1 \cup \{E\}$

is unsatisfiable then $\mathcal{G}_1 \models \neg E$ and $\mathcal{G}_1 \models E \vee F$ holds by assumption. Hence $\mathcal{G}_1 \models F$. Then we see that $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$ forms an unbarred division as follows. Suppose there exists an A- and C-sentence B that bars $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$, that is, $\mathcal{G}_1 \cup \{F\} \models B$ and $\mathcal{G}_2 \models B$. But from $\mathcal{G}_1 \models F$, we conclude that B bars $(\mathcal{G}_1, \mathcal{G}_2)$, which is a contradiction. Similar for the case that $\mathcal{G}_1 \cup \{F\}$ is unsatisfiable. Thus we can assume that $\mathcal{G}_1 \cup \{E\}$ and $\mathcal{G}_1 \cup \{F\}$ are satisfiable.

Suppose both alternatives fail to be unbarred divisions. This means there exist sentences B_i ($i \in \{1, 2\}$) that bar $(\mathcal{G}_1 \cup \{E\}, \mathcal{G}_2)$ and $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$ respectively, that is, $\mathcal{G}_1 \cup \{E\} \models B_1$, $\mathcal{G}_2 \models \neg B_1$ and $\mathcal{G}_1 \cup \{F\} \models B_2$, $\mathcal{G}_2 \models \neg B_2$. Thus $\mathcal{G}_1 \models B_1 \vee B_2$. Moreover $\mathcal{G}_2 \models \neg(B_1 \vee B_2)$ follows from $\mathcal{G}_2 \models \neg B_1$ and $\mathcal{G}_2 \models \neg B_2$. Therefore $(B_1 \vee B_2)$ bars the pair $(\mathcal{G}_1, \mathcal{G}_2)$, which is a contradiction to the assumption that $(\mathcal{G}_1, \mathcal{G}_2)$ is unbarred. Hence either of the pairs $(\mathcal{G}_1 \cup \{E\}, \mathcal{G}_2)$ or $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$ forms an unbarred division. From this we conclude that $\mathcal{G} \cup \{E\} \in S$ or $\mathcal{G} \cup \{F\} \in S$. Thus the satisfaction property of S has been verified for the considered case.

In sum, there exists a collection of sets S admitting the satisfaction properties. Furhtermore from the assumption that there exists no interpolation for the sentence $A \to C$, we conclude that $\{A, \neg C\} \in S$. Thus by model existence $\{A, \neg C\}$ is satisfiable. However then $A \to C$ cannot be valid. This shows the existence of an interpolant for $A \to C$. $\square$

5.2. Robinson's Joint Consistency Theorem

For the next result we need to define precisely what is to be understood by a theory of a language.

Definition 5.1. A *theory* in a language $\mathcal{L}$ is a set of sentences of $\mathcal{L}$ that is closed under the consequence relation. We call an element of a theory a *theorem*. A theory T is called *complete* if for every sentence F of $\mathcal{L}$ either $F \in T$ or $\neg F \in T$.

A theory T' is an *extension* of a theory T if $T \subseteq T'$. An extension T' is *conservative* if any sentence F of the language of T that is a theorem of T' is a theorem of T.

Note that any mathematical theory like for example the natural numbers together with the usual operations can be expressed as an (infinite) theory in the above sense. Moreover any reasoning over data-types like for example arrays can be so represented, compare also Chapter 9.

The (not difficult) proof of the next lemma is omitted, but see Problem 5.3 below.

Lemma 5.2. *The union $S \cup T$ of two theories S and T is satisfiable iff there is no sentence in S whose negation is in T.*

Theorem 5.2. *Let $\mathcal{L}_0$, $\mathcal{L}_1$, and $\mathcal{L}_2$ be languages such that $\mathcal{L}_0 = \mathcal{L}_1 \cap \mathcal{L}_2$. Let T_i be a theory in $\mathcal{L}_i$ ($i \in \{0, 1, 2\}$). Let T_3 be the set of sentences of $\mathcal{L}_1 \cup \mathcal{L}_2$ that are consequences of $T_1 \cup T_2$. If T_1, T_2 are conservative extensions of T_0, then T_3 is a conservative extension of T_0.*

Proof. Suppose A is a sentence of $\mathcal{L}_0$ that is a theorem of T_3. Set $U_2 := \{B \mid T_2 \cup \{\neg A\} \models B\}$. As $A \in T_3$, $T_1 \cup T_2 \cup \{\neg A\}$ is unsatisfiable hence also $T_1 \cup U_2$ is unsatisfiable.

By the lemma there exists a theorem $C \in T_1$ whose negation $\neg C$ is in U_2. It is easy to see that C, $\neg C$ are sentences of $\mathcal{L}_0$. Moreover $\neg A \to \neg C$ is of $\mathcal{L}_0$. By assumption on T_1, C is a theorem of T_0, while $\neg A \to \neg C$ is in T_2 and thus a theorem of T_0. Thus also $C \to A \in T_0$, which together with $C \in T_0$ yields that $A \in T_0$. □

Based on the above theorem we can state and prove Robinson's joint consistency theorem.

Corollary 5.1 (Robinson's Joint Consistency Theorem)**.** *Let $\mathcal{L}_i$ and T_i ($i \in \{0,1,2\}$) be as in the theorem. If T_0 is complete and T_1, T_2 are satisfiable extensions of T_0, then $T_1 \cup T_2$ is satisfiable.*

Proof. Note that a satisfiable extension of a complete theory T is conservative. Assume there exists a theorem A of the extension in the language of the complete theory. Then if $A \in T$ we are done and if $\neg A \in T$, then the extension cannot be satisfiable. On the other hand a conservative extension of a satisfiable theory has to be satisfiable. Otherwise, assume the extension is unsatisfiable, then by the completeness theorem this extension is inconsistent and any formula is contained in it, for example $\forall x(x \neq x)$. The latter is clearly a sentence that must not be a theorem of the original theory.

Based on these observations the corollary is a direct consequence of the theorem. □

Problems

Problem 5.1. Show Lemma 5.1. *Hint*: Those individual constants that occur in A but not in C have to be suitably replaced, for example with fresh variables. And observe that since $A \to C$ is valid so is $\forall x_1 \dots x_n(A' \to C)$, where A' denotes the result of the replacement of constants.

Problem 5.2. Consider the proof of Theorem 5.1.

(i) Show that all applicable satisfaction properties are fulfilled by the set S.

(ii) Extend the theorem to languages containing equality. *Hint*: Study the proof of Lemma 6.1 and observe that the existence of a valid implication $A \to C$ is equivalent to the statement that $A \land \neg C$ is unsatisfiable.

(iii) Extend the theorem to languages containing individual and function constants.

Problem 5.3. Show Lemma 5.2. *Hint*: The direction from right to left is obvious and the other direction follows by the use of compactness and Craig's interpolation theorem.

Problem 5.4. Show that (i) a satisfiable extension of a complete theory is conservative and (ii) that a conservative extension of a satisfiable theory is satisfiable.

6.

Normal Forms and Herbrand's Theorem

The central result in this chapter is Herbrand's theorem, a theorem that is as important in formal logic as in automated reasoning and which we will employ in latter chapters. As forerunner to this theorem two normal form theorems will be presented in the first two sections.

Such a normal form theorem falls into two categories: either the theorem tell us that for a given formula F there exists a formula G of specific syntactic form such that F and G are *logically equivalent*, or it tells us that F and G are *equivalent for satisfaction*. The aim of normal form theorems is to provide us with (simple) procedures to transform arbitrary formulas into a form that can later easily analysed.

In Section 6.3 Herbrand's theorem is proven together with some corollaries that will be used later. In Section 6.4 it is shown that equality, individual and function constants can be eliminated from formulas without affecting the satisfiability.

6.1. Prenex Normal Form

In this section we state and prove a normal form theorem of the first type: a given formula F is shown to be transformable into prenex normal form and this transformation preserves logical equivalence.

Definition 6.1. A formula F is in *prenex normal form* if it has the form

$$\mathsf{Q}_1 x_1 \cdots \mathsf{Q}_n x_n \; G(x_1, \ldots, x_n) \qquad\qquad \mathsf{Q}_i \in \{\forall, \exists\} ,$$

where G is quantifier-free. The subformula G is also called *matrix*. If the matrix G is a conjunction of disjunctions of literals, we say F is in *conjunctive prenex normal form* (*CNF* for short). Recall that a literal is an atomic formula or a negated atomic formula.

Remark 6.1. Observe the overloading of the abbreviation for *conjunctive prenex normal form*. In Chapter 2 we used CNF to denote the conjunctive normal form of a propositional formula. In the following we will sometimes also call a quantifier-free formula that is a conjunction of disjunctions of literals a CNF. No confusion will arise from this.

Note that the conjunctive prenex normal form need not be unique as illustrated by the next example.

Example 6.1. Consider $\forall x F(x) \leftrightarrow G(a)$, which abbreviates:

$$(\forall x F(x) \to G(a)) \land (G(a) \to \forall x F(x)) \ .$$

One logically equivalent CNF would be

$$\forall x \exists y ((\neg F(y) \lor G(a)) \land (\neg G(a) \lor F(x)) \ .$$

Another logically equivalent CNF is obtained if the quantifiers are pulled out in different order. That is

$$\exists y \forall x ((\neg F(y) \lor G(a)) \land (\neg G(a) \lor F(x)) \ ,$$

is also a CNF of F.

Theorem 6.1. *For any formula F there exists a formula G in prenex normal form such that $F \equiv G$.*

Proof. To prove the theorem we give a construction to transform F into a formula G in prenex normal form. Each step performed preserves logical equivalence of formulas.

(i) We replace all occurring implication signs $\to$ in F, employing the equivalence $(E \to F) \equiv (\neg E \lor F)$.

(ii) We rename bound variables such that each quantifier introduces a unique bound variable. The proof that this step preserves equivalence is left to the reader, see Problem 6.2.

(iii) We pull quantifiers out using one of the following equivalences:

$$\begin{array}{ll} \neg \forall x F(x) \equiv \exists x \neg F(x) & \neg \exists x F(x) \equiv \forall x \neg F(x) \\ \mathsf{Q} x E(x) \odot F \equiv \mathsf{Q} x (E(x) \odot F) & \end{array}$$

where $\mathsf{Q} \in \{\forall, \exists\}$, $\odot \in \{\land, \lor\}$, and in the last equivalence the variable x must not occur free in F. It is easy to see that replacement of logically equivalent formulas preserves logical equivalence, see Problem 6.1.

$\square$

By adapting the transformation procedure so that also the matrix of the obtained prenex normal form is normalised, we immediately get the next result.

Corollary 6.1. *For any formula F there exists a formula G in CNF such that $F \equiv G$.*

6.2. Skolem Normal Form

In this section we state and prove a normal form theorem of the second type: a given formula F is shown to be transformable into Skolem normal form and this transformation is satisfiability preserving.

An *existential* formula F is of form

$$\exists x_1 \cdots \exists x_n \ G(x_1, \dots, x_n) \ ,$$

where the matrix G is quantifier free. A *universal* formula is of form

$$\forall x_1 \cdots \forall x_n \ G(x_1, \dots, x_n) \ .$$

For later arguments we note that any quantifier-free formulas is existential and universal: simply set $n = 0$ in the above presentation.

Definition 6.2. A formula F is in *Skolem normal form* (*SNF* for short) if F is universal and in CNF.

Let $\mathcal{L}$ be a language and $\mathcal{L}^+$ an extension of $\mathcal{L}$, that is, the constants in $\mathcal{L}$ form a subset of the constants in the language $\mathcal{L}^+$. Suppose further that $\mathcal{I}$ is an interpretation of $\mathcal{L}$ and $\mathcal{I}^+$ an interpretation of $\mathcal{L}^+$ such that $\mathcal{I}$ and $\mathcal{I}^+$ coincide on $\mathcal{L}$. Then $\mathcal{I}^+$ is called *expansion* of $\mathcal{I}$.

Definition 6.3. Given a sentence F, we define its *Skolemisation* F^S as follows:

(i) Transform F into a CNF F' such that F' can be represented as

$$\mathsf{Q}_1 x_1 \cdots \mathsf{Q}_m x_m \ G(x_1, \dots, , x_m) \ .$$

(ii) Set $F'' = F'$ and repeatedly transform F'' by replacing the sentence

$$\forall x_1 \cdots \forall x_{i-1} \exists x_i \mathsf{Q}_{i+1} x_{i+1} \cdots \mathsf{Q}_m x_m \ G(x_1, \dots, x_i, \dots, x_m)$$

by the sentences $\mathsf{s}(F'')$

$$\forall x_1 \cdots \forall x_{i-1} \mathsf{Q}_{i+1} x_{i+1} \cdots \mathsf{Q}_m x_m \ G(x_1, \dots, f(x_1, \dots, x_{i-1}), \dots, x_m)$$

where f denotes a *fresh* function symbol of arity $i-1$. The transformation ends if no existential quantifier remains.

The fresh function symbols introduced in the process of Skolemisation are often called *Skolem functions*. We say formulas F and G are *equivalent for satisfiability* if F is satisfiable iff G is satisfiable. This is denoted as $F \approx G$.

Theorem 6.2. *For any formula F there exists a computable formula G in SNF such that $F \approx G$.*

Proof. Without loss of generality we assume that F is already in CNF. Otherwise we transform it in CNF using Corollary 6.1. It suffices to prove that

$F \approx \mathsf{s}(F)$, as the theorem then follows by an inductive argument from the special case. We fix some notation:

$$\begin{aligned}
&F := \forall x_1 \cdots \forall x_{i-1} \exists x_i \mathsf{Q}_{i+1} x_{i+1} \cdots \mathsf{Q}_m x_m \; G(x_1, \ldots, x_i, \ldots, x_m) \\
&s(F) := \forall x_1 \cdots \forall x_{i-1} \mathsf{Q}_{i+1} x_{i+1} \cdots \mathsf{Q}_m x_m \; G(x_1, \ldots, f(x_1, \ldots, x_{i-1}), \ldots, x_m) \\
&H(a_1, \ldots, a_i) := \mathsf{Q}_{i+1} x_{i+1} \cdots \mathsf{Q}_m x_m \; G(a_1, \ldots, a_i, x_{i+1}, \ldots, x_m)
\end{aligned}$$

where $a_1, \ldots, a_i$ are fresh variables.

First assume that $s(F)$ is satisfiable, that is, there exists a model $\mathcal{M}$ such that $\mathcal{M} \models \mathsf{s}(F)$. Then clearly $\mathcal{M}$ also models F. Indeed the stronger assertion $s(F) \to F$ is valid.

The other direction is more involved. Suppose F is satisfiable and let $\mathcal{M}$ be a model of F. Then we can expand $\mathcal{M}$ to a model $\mathcal{M}^+$ such that for any assignment of the variables $a_1, \ldots, a_{i-1}$

$$\mathcal{M}^+ \models H(a_1, \ldots, a_{i-1}, f(a_1, \ldots, a_{i-1})) \; . \tag{6.1}$$

To define $f^{\mathcal{M}^+}$ we fix $i-1$ elements $b_1, \ldots, b_{i-1} \in \mathcal{M}$ and consider the set B of all elements $b \in \mathcal{M}$ such that H holds, where the variables $a_1, \ldots, a_{i-1}$ are interpreted as $b_1, \ldots, b_{i-1}$, respectively.

By assumption $B \neq \varnothing$. Thus we can pick (in an arbitrary but fixed way) an element $b \in B$ and set

$$f^{\mathcal{M}^+}(b_1, \ldots, b_{i-1}) := b \; .$$

In this way the interpretation of the function constant f is completely described and the assertion (6.1) follows. Hence $\forall x_1 \cdots \forall x_{i-1} H(x_1, \ldots, f(x_1, \ldots, x_{i-1})) = s(F)$ is satisfiable. □

6.3. Herbrand's Theorem

In this section we state and prove the main result of this chapter. A term t is called *closed* or *ground*, if t does not contain (free) variables.

Definition 6.4. A *Herbrand universe* for a language $\mathcal{L}$ is the set of all closed terms (of $\mathcal{L}$). If $\mathcal{L}$ doesn't contain an individual constant, then we add a fresh constant to $\mathcal{L}$.

An interpretation $\mathcal{I}$ (of $\mathcal{L}$) is a *Herbrand interpretation* if

(i) the universe of $\mathcal{I}$ is the Herbrand universe H for $\mathcal{L}$ and

(ii) the interpretation $\mathcal{I}$ is defined such that

$$t^{\mathcal{I}} := t \qquad \text{for any closed term } t$$

A Herbrand interpretation $\mathcal{I}$ is a *Herbrand model* of a set of formulas $\mathcal{G}$ if $\mathcal{I} \models \mathcal{G}$.

A specific Herbrand model has been constructed in the proof of Lemma 4.4 in Chapter 4.3. Thus (by the proof of) Lemma 4.4 we already know that a satisfiable set of (universal) sentences $\mathcal{G}$ has a Herbrand model. In preparation for Herbrand's theorem, we argue directly. Let $t_1, \dots, t_n$ be terms. Then the formula $F(t_1, \dots, t_n)$ is called an instance of $\forall x_1 \cdots \forall x_n F(x_1, \dots, x_n)$. If all terms t_i ($1 \leqslant i \leqslant n$) are ground, $F(t_1, \dots, t_n)$ is called a *ground instance*.

Suppose that $\mathcal{I}$ models $\mathcal{G}$ and let $\forall x_1 \cdots \forall x_n F(x_1, \dots, x_n) \in \mathcal{G}$. By definition of the satisfaction relation $\mathcal{I}$ also models every ground instance $F(t_1, \dots, t_n)$ of $\forall x_1 \cdots x_n F(x_1, \dots, x_n)$. Consider a Herbrand interpretation $\mathcal{J}$ (of the language of $\mathcal{G}$) that satisfies exactly the same instances $F(t_1, \dots, t_n)$ as $\mathcal{I}$. This amounts to set $\mathcal{J}$ as the collection of all true atoms $F(t_1, \dots, t_n)$ in the interpretation $\mathcal{I}$. Then $\mathcal{J} \models \forall x_1 \cdots \forall x_n F(x_1, \dots, x_n)$ and thus $\mathcal{J}$ is a Herbrand model of $\mathcal{G}$.

This observation motivates a new notation. Let $\mathcal{I} = (\mathcal{A}, \ell)$ be an interpretation and let F be a formula. Recall that Lemma 3.2 states that only a finite part of the look-up table ℓ is necessary to conclude the truth value of F as only finitely many variables may occur in a given formula F.

Let $a_1, \dots, a_n$ denote the set of (free) variables in F. Then only the values $\ell(a_1)$, $\dots \ell(a_n)$ of the environment ℓ are important. Thus instead of $(\mathcal{A}, \ell) \models F$ we sometimes write:

$$\mathcal{A} \models F[\ell(a_1), \dots, \ell(a_n)] \ .$$

Theorem 6.3. *Let $\mathcal{G}$ be a set of universal sentences (of $\mathcal{L}$) without $=$. Then the following assertions are equivalent:*

(i) *$\mathcal{G}$ is satisfiable.*

(ii) *$\mathcal{G}$ has a Herbrand model (over $\mathcal{L}$).*

(iii) *every finite subset of $\mathsf{Gr}(\mathcal{G})$ has a Herbrand model (over $\mathcal{L}$).*

here we set

$$\mathsf{Gr}(\mathcal{G}) := \{F(t_1, \dots, t_n) \mid \forall x_1 \cdots \forall x_n F(x_1, \dots, x_n) \in \mathcal{G}, \textit{where the } t_i \textit{ are closed}\} \ .$$

Proof. The argument for the equivalence of the first two statements has already been given above.

To see that the third item is equivalent, it suffices to show that item (iii) implies item (i). Thus we assume that any finite subset of $\mathsf{Gr}(\mathcal{G})$ has a Herbrand model. Then in particular any finite subset of $\mathsf{Gr}(\mathcal{G})$ has a model and hence $\mathsf{Gr}(\mathcal{G})$ itself has a model by compactness. Thus (using the equivalence of the first two statements) $\mathsf{Gr}(\mathcal{G})$ has a Herbrand model $\mathcal{M}$.

Now let $\forall x_1 \cdots \forall x_n F(x_1, \dots, x_n) \in \mathcal{G}$, then for any sequence of (closed) terms $t_1, \dots, t_n$, we have $F(t_1, \dots, t_n) \in \mathsf{Gr}(\mathcal{G})$. Thus $\mathcal{M} \models F(t_1, \dots, t_n)$ for any sequence $t_1, \dots, t_n$. Hence, $\mathcal{M} \models F[t_1, \dots, t_n]$ for all domain elements $t_1, \dots, t_n \in \mathcal{M}$. This implies $\mathcal{M} \models \forall x_1 \cdots \forall x_n F(x_1, \dots, x_n)$ by definition. This holds for any sentence in $\mathcal{G}$ and thus $\mathcal{M} \models \mathcal{G}$. $\square$

To simplify later developments we represent Herbrand's theorem in a more condensed form below.

Corollary 6.2. *Let $\mathcal{G}$ be a set of universal sentences (of $\mathcal{L}$) without $=$. Either $\mathcal{G}$ has a Herbrand model or $\mathcal{G}$ is unsatisfiable. For the latter case the following assertions hold (and are equivalent):*

(i) *There exists a finite subset of* $\mathsf{Gr}(\mathcal{G})$ *whose conjunction is unsatisfiable.*

(ii) *There exists a finite subset* S *of* $\mathsf{Gr}(\mathcal{G})$ *such that the disjunction of the negation of formulas in* S *is valid.*

Proof. By the theorem $\mathcal{G}$ either has a Herbrand model or is unsatisfiable. Moreover in the latter case there exists a finite subset of $\mathsf{Gr}(\mathcal{G})$ whose conjunction is unsatisfiable by the theorem. Otherwise all finite subset of $\mathsf{Gr}(\mathcal{G})$ would be satisfiable from which we would conclude that $\mathcal{G}$ is satisfiable.

Hence it remains to verify that both items in the corollary are equivalent. For that suppose there exists a finite subset of $\mathsf{Gr}(\mathcal{G})$ whose conjunction C is unsatisfiable. Then clearly the negation of this conjunction C is a disjunction D of negations of formulas in $\mathsf{Gr}(\mathcal{G})$ and D is finite. Moreover D is valid. $\square$

We can paraphrase Herbrand's theorem (and Corollary 6.2) as the statement: A universal sentence $\forall x F(x)$ is unsatisfiable if and only if there exists a finite sets S of ground instances $F(t)$ for terms t in the Herbrand universe such that S is unsatisfiable.

Note that the restriction to *universal sentences* in Theorem 6.3 and Corollary 6.2 is essential: On one hand we cannot generalise the theorem to universal formulas, on the other we cannot generalise it to general sentences, see Problem 6.3. One way to overcome this problem is to assert that any formula $F(a_1, \dots, a_n)$ (with free variables $a_1, \dots, a_n$) is understood to be implicitly universally quantified as follows: $\forall x_1 \cdots x_n F(x_1, \dots, x_n)$, see for example [27].

Corollary 6.3. *If* $F(a_1, \dots, a_n)$ *is a quantifier-free formula in a language* $\mathcal{L}$ *with at least one constant, then* $\exists x_1 \cdots \exists x_n F(x_1, \dots, x_n)$ *is valid iff there are ground terms* $t_1^k, \dots, t_n^k$, $k \in \mathbb{N}$ *such that the* Herbrand disjunction $F(t_1^1, \dots, t_n^1) \vee \cdots \vee F(t_1^k, \dots, t_n^k)$*, is valid.*

Proof. If $\exists x_1 \cdots \exists x_n F(x_1, \dots, x_n)$ is valid, then $\forall x_1 \cdots \forall x_n \neg F(x_1, \dots, x_n)$ is unsatisfiable and vice versa. By Corollary 6.2 there exists a finite disjunction of formulas $F(t_1^k, \dots, t_n^k)$ that is valid. $\square$

Based on Herbrand's theorem a naive form of automating the verification of a given sentence F becomes possible. Let F be an arbitrary sentence in a language $\mathcal{L}$. Then by Theorem 6.2 there exists a formula F' in SNF such that $F \approx F'$. Suppose F' has the following shape:

$$\forall x_1 \cdots \forall x_n \; G(x_1, \dots, x_n) \; .$$

Let H be the Herbrand universe for $\mathcal{L}$. Recall that G is in CNF. Then we consider all possible Herbrand interpretations of $\mathcal{L}$. For that we make use of so called *semantic trees*. Let $\mathcal{A}$ be a set of atomic formulas (of $\mathcal{L}$) over the Herbrand universe H and let $A_0, A_1, \dots$ be some enumeration of $\mathcal{A}$. The *semantic tree* T is inductively defined as follows.

– The tree which contains only the root is a semantic tree.

– The two edges leaving the root are labelled by A_0 or $\neg A_0$, respectively

– Let I be a node in T. Then I is either a
  (i) leaf node or
  (ii) the edges e_1, e_2 leaving node I are labelled by A_{n+1} and $\neg A_{n+1}$ respectively, when the edge that enters node I is labelled by A_n or $\neg A_n$.

Any path in T gives rise to a partial Herbrand interpretation $\mathcal{I}$ of F'. We traverse the path and set all literals used as edge labels true in $\mathcal{I}$. In this way a semantic tree represents all possible Herbrand interpretations of F' (as $\mathcal{L}$ is assumed to be countable).

Let I denote a node in T and let $\mathcal{I}$ denote the partial Herbrand interpretation induced by this node. We call I *closed* if there exists a ground instance D of a disjunction in G such that $\mathcal{I} \not\models D$ and thus $\mathcal{I} \not\models F'$. Clearly when all leaves in T are closed, then there exists a finite sets S of ground instances:

$$G(t_1^k, \dots, t_n^k) ,$$

for closed terms $t_1^k, \dots, t_n^k$, $k \geqslant 1$ in the Herbrand universe H such that S is unsatisfiable.

By Herbrand's theorem this implies that F' is unsatisfiable and thus F is unsatisfiable.

Hence in order to prove that a given existential formula is valid or that a given universal formula is unsatisfiable, we construct the semantic tree T as above iteratively. Note that we can stop the construction of T as soon as all leaf nodes in T are closed.

This procedure can be automated and provides us with a sound and complete algorithm A. Here soundness means that A will never refute a formula F that is satisfiable and completeness means that for any unsatisfiable formula F we will find a finite semantic tree witnessing that F is unsatisfiable. Of course the algorithm A need not terminate and is hopelessly inefficient. Still, this idea forms the basis of modern tools in automated reasoning.

6.4. Eliminating Function Symbols and Identity

Above we restricted Herbrand's theorem to languages without equality. In this section we show how to overcome this restriction. In addition we show how to eliminate individual and function constants from the language.

We start with the transformation rules to eliminate individual and function constants. For that observe that any formula F is logically equivalent to a formula G such that individual and function constants only occur immediately to the right of an equality sign. So the only occurrence of an n-place function symbol or a constant is in atomic formulas of the following shape: $a = f(b_1, \dots, b_n)$, where the indicated terms a, b_1, …, b_n are variables. To obtain the formula G from F we iteratively apply the following transformation. Suppose the n-place function symbol occurs somewhere else in F than immediately to the right of $=$. Suppose f is the first symbol (also known as *root symbol*) of a term t occurring in a subformula A of F. Let x be a fresh bound variable and denote as F' the result of replacing $A(t)$ by $\exists x(x = t \wedge A(x))$. It is not difficult to argue that F' is logically equivalent of F.

Hence, we assume that in the given formula F individual and function constants only occur immediately to the right hand of $=$. Based on this information we show how to replace any of the occurring individual and function constants. Let F be a formula, f an n-place function symbol or a constant occurring in $a = f(b_1, \dots, b_n)$. Then we replace all occurrences of this equality by a $P(b_1, \dots, b_n, a)$, where the predicate constant P is fresh. The result of this transformation is denoted as F''. Let $C(f)$ denote the following sentence, denoted as *functionality axiom*:

$$\forall x_1 \cdots \forall x_n \exists y \forall z (P(x_1, \dots, x_n, z) \leftrightarrow z = y) \ .$$

Then we obtain the following lemma, whose not difficult proof is left to the reader.

Lemma 6.1. *F is satisfiable if and only if $F'' \wedge C(f)$ is satisfiable.*

We turn our attention to the elimination of the symbol $=$. For that we assume without loss of generaltiy that the formula F admits only predicate constants as non-logical symbols. (Otherwise we first employ Lemma 6.1.) We make use of an additional binary predicate symbol $\risingdotseq$ together with the following *equivalence axioms E*.

$$\forall x\ x \risingdotseq x \wedge \forall x \forall y\ (x \risingdotseq y \rightarrow y \risingdotseq x) \wedge \forall x \forall y \forall z\ ((x \risingdotseq y \wedge y \risingdotseq z) \rightarrow x \risingdotseq z) \ .$$

In addition for each n-ary predicate constant P we consider the following sentence $C(P)$

$$\forall x_1 \cdots \forall x_n \forall y_1 \cdots \forall y_n\ ((x_1 \risingdotseq y_1 \wedge \cdots \wedge x_n \risingdotseq y_n) \rightarrow \\ \rightarrow (P(x_1, \dots, x_n) \leftrightarrow P(y_1, \dots, y_n)) \ .$$

For any formula F let F''' denote the result of replacing the equality sign $=$ everywhere by $\risingdotseq$ and let $C(F)$ denote the conjunction of all *congruence axioms* $C(P)$ for any constant P. Then we obtain the following lemma, whose proof follows similarly to Lemma 6.1.

Lemma 6.2. *F is satisfiable if and only if $F''' \wedge E \wedge C(F)$ is satisfiable.*

Lemma 6.1 and 6.2 allow us to eliminate individual and function constants and the equality symbol from considered formulas, while preserving satisfaction. In particular this means that Herbrand's theorem (in all variants discussed above) remains valid. We conclude this chapter with the following theorem.

Theorem 6.4. *For any formula F there exists a formula G such that G does neither contain individual or function constants nor equality and $F \approx G$.*

Problems

Problem 6.1. Two formulas are *equivalent over an interpretation $\mathcal{I}$* if they have the same truth value with respect to $\mathcal{I}$. Show that the following hold for equivalence over any interpretation $\mathcal{I}$ (and hence for logical equivalence):

(i) If sentence G is obtained from sentence F by replacing each occurrence of an atomic sentence A by an equivalent sentence B, then F and G are equivalent.

(ii) Show the same holds for an atomic *formula* A and an equivalent formula B.

(iii) Show that this holds for *arbitrary* subformulas A.

Problem 6.2. Show that

(i) If F is a formula and x a bound variable in F, then F is logically equivalent to a formula in which x doesn't occur at all.

(ii) Generalise this to any number of variables $x_1, \ldots, x_n$.

Problem 6.3. Let $\mathcal{L} = \{\mathsf{c}, \mathsf{P}\}$.

(i) Give the Herbrand universe for $\mathcal{L}$.

(ii) Give two examples of Herbrand interpretations of $\mathcal{L}$.

(iii) Let $\mathcal{G}_1 = \{\mathsf{P}(\mathsf{c}), \exists x \neg \mathsf{P}(x)\}$. Show that $\mathcal{G}_1$ is satisfiable, but doesn't have a Herbrand model.

(iv) Let $\mathcal{G}_2 = \{\mathsf{P}(\mathsf{c}), \neg \mathsf{P}(x)\}$. Show that $\mathcal{G}_2$ is satisfiable, but doesn't have a Herbrand model.

Problem 6.4. Prove Lemma 6.1.

Hint: It simplifies the argument if the following *auxiliary axiom* D is employed:

$$\forall x_1 \cdots \forall x_n \forall z (P(x_1, \ldots, x_n, z) \leftrightarrow z = f(x_1, \ldots, x_n))$$

Note that $D \models C$ and $D \models F \leftrightarrow F''$.

Problem 6.5. Prove Lemma 6.2.

Hint: Only the direction from right to left is of interest. Start with a model $\mathcal{M}$ for $F''' \wedge E \wedge C(F)$ and define an interpretation whose universe consists of all equivalence classes (with respect to $\leftrightharpoons$) and whose denotation of $\leftrightharpoons$ is the identity.

7.

The Curry-Howard Isomorphism

In this chapter we consider the connection between proofs and programs in more detail. For that we describe the *Curry-Howard isomorphism* between intuitionistic natural deduction and the typed λ-calculus. This correspondence allows us to speak of programs and proofs interchangingly and transform or develop formalisms and methods in one area to apply it to the other. We restrict ourselvs to the bare essentials in presenting the Curry-Howard correspondence. For a complete account the reader is kindly referred to Goubault-Larrecq and Makie, see [35].

7.1. A Problem with the Excluded Middle

In order to set the table for the presentation of the Curry-Howard isomorphism it is necessary to describe intuitionistic logic and the λ-calculus. In this section we give the usual motivating example of intuitionistic logic and present a calculus for this logic.

Theorem 7.1. *There are solutions of the equation $x^y = z$ with x and y irrational and z rational.*

Proof. We give a non-constructive proof. Clearly $\sqrt{2}$ is an irrational number. Consider $\sqrt{2}^{\sqrt{2}}$: One of the following two cases has to occur:

(i) $\sqrt{2}^{\sqrt{2}}$ is rational. In this case put

$$x = \sqrt{2} \qquad y = \sqrt{2} \qquad z = \sqrt{2}^{\sqrt{2}}$$

Clearly these settings solve the equation $x^y = z$. Thus the theorem is proven.

(ii) $\sqrt{2}^{\sqrt{2}}$ is irrational. In this case put

$$x = \sqrt{2}^{\sqrt{2}} \qquad y = \sqrt{2} \qquad z = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = 2$$

Again the equation $x^y = z$ is solved and the theorem is proven.

	introduction	*elimination*
$\wedge$	$\dfrac{E \quad F}{E \wedge F}\ \wedge\colon \mathsf{i}$	$\dfrac{E \wedge F}{E}\ \wedge\colon \mathsf{e} \qquad \dfrac{E \wedge F}{F}\ \wedge\colon \mathsf{e}$
$\vee$	$\dfrac{E}{E \vee F}\ \vee\colon \mathsf{i} \qquad \dfrac{F}{F \vee F}\ \vee\colon \mathsf{i}$	$\dfrac{E \vee F \quad \boxed{\begin{array}{c}E\\ \vdots\\ G\end{array}} \quad \boxed{\begin{array}{c}F\\ \vdots\\ G\end{array}}}{G}\ \vee\colon \mathsf{e}$
$\rightarrow$	$\dfrac{\boxed{\begin{array}{c}E\\ \vdots\\ F\end{array}}}{E \rightarrow F}\ \rightarrow\colon \mathsf{i}$	$\dfrac{E \quad E \rightarrow F}{F}\ \rightarrow\colon \mathsf{e}$

Figure 7.1.: Intuitionistic Propositional Rules (Part I)

□

The problem with the above proof is that it is *non-constructive*: The statement of the theorem is existential: something should exist namely the numbers x, y, and z. Despite this the proof does not provide a method to actually *construct* these numbers. This is a serious problem if we want to extract a program out of the given proof which is exactly the point of the Curry-Howard correspondence. To overcome this problem we consider proofs of a specific form: *intuitionistic proofs*.

7.2. Natural Deduction for Intuitionistic Logic

In this section we introduce a formal system for intuitionistic logic and claim its soundness and completeness with respect to the standard Kripke-semantics. As the focus of this section is on the correspondence between proofs and programs we are not concerned with the semantics of intuitionistic logic here. (For more information see [18].) Note that the semantics of intuitionistic logic (even for the propositional case) is more complex than the one considered for classical (propositional) logic.

It suffices if we consider propositional logic only. Below we give the natural deduction rules of intuitionistic logic, denoted as NJ, see Figure 11.2 and Figure 11.3. In the following the natural deduction rules as defined in Figure 2.1 are denoted as NK.

The only difference between the classical rules and those given here is the absence of the double-negation rule:

$$\frac{\neg\neg F}{F}\ \neg\neg\colon \mathsf{e}$$

	introduction	*elimination*
$\neg$	$\dfrac{\boxed{\begin{array}{c}E\\ \vdots\\ \bot\end{array}}}{\neg E}\ \neg\!:\mathsf{i}$	$\dfrac{F \quad \neg F}{\bot}\ \neg\!:\mathsf{e}$
$\bot$		$\dfrac{\bot}{F}\ \neg\!:\mathsf{e}$

Figure 7.2.: Intuitionistic Proposition Rules (Part II)

This seemingly small change has the effect that in NJ the *tertium non-datur* $F \vee \neg F$ is no longer derivable: $\mathsf{NJ} \not\vdash F \vee \neg F$.

7.3. Typed λ-Calculus

In this section we (very) briefly introduce the typed λ-calculus. See [8] for extensive information on the untyped λ-calculus and see [9, 28] for background information on the typed system.

Definition 7.1. We define the set of *types* T as follows:

- a variable type: α, β, γ, ...
- if σ, τ are types, then $(\sigma \times \tau)$ is a (*product*) type
- if σ, τ are types, then $(\sigma \to \tau)$ is a (*function*) type

Definition 7.2. The *typed λ-terms* are defined as follows:

- any (typed) variable $x : \sigma$ is a (typed) term
- if $M : \sigma$, $N : \tau$ are terms, then $\langle M, N\rangle : \sigma \times \tau$ is a term
- if $M : \sigma \times \tau$ is a term, then $\mathsf{fst}(M) : \sigma$ and $\mathsf{snd}(M) : \tau$ are terms
- if $M : \tau$ is a term, $x : \sigma$ a variable,
  then the *abstraction* $(\lambda x^\sigma.M) : \sigma \to \tau$ is a term
- if $M : \sigma \to \tau$, $N : \sigma$ are terms, then the *application* $(MN) : \tau$ is a term.

Example 7.1. The following are (well-formed, typed) terms

$$\lambda fx.fx : (\sigma \to \tau) \to \sigma \to \tau \qquad \langle \lambda x.x, \lambda y.y\rangle : (\sigma \to \sigma) \times (\tau \to \tau)\ ,$$

but $\lambda x.xx$ cannot be typed!

Definition 7.3. The set of *free variables* of a term is defined as follows

- $\mathsf{FV}(x) = \{x\}$.

- $\mathsf{FV}(\lambda x.M) = \mathsf{FV}(M) - \{x\}$
- $\mathsf{FV}(MN) = \mathsf{FV}(\langle M, N\rangle) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$.
- $\mathsf{FV}(\mathsf{fst}(M)) = \mathsf{FV}(\mathsf{snd}(M)) = \mathsf{FV}(M)$.

Occurrences of x in the scope of λ are called *bound*: $\lambda x.xy(\lambda y.xy(\lambda x.z))y$. This notion is made precise in the next definition.

Definition 7.4. The set of *bound variables* of a term is defined as follows

- $\mathsf{BV}(x) = \varnothing$.
- $\mathsf{BV}(\lambda x.M) = \mathsf{BV}(M) \cup \{x\}$.
- $\mathsf{BV}(MN) = \mathsf{BV}(\langle M, N\rangle) = \mathsf{BV}(M) \cup \mathsf{BV}(N)$.
- $\mathsf{BV}(\mathsf{fst}(M)) = \mathsf{BV}(\mathsf{snd}(M)) = \mathsf{BV}(M)$.

In the definition of β-reduction below we make use of substitution.

Definition 7.5. $M[x := N]$ denotes the result of substituting N for x in M

- $x[x := N] = N$ and if $x \neq y$, then $y[x := N] = y$
- $(\lambda x.M)[x := N] = \lambda x.M$
- $(\lambda y.M)[x := N] = \lambda y.(M[x := N])$, if $x \neq y$ and $y \notin \mathsf{FV}(N)$
- $(M_1M_2)[x := N] = (M_1[x := N])(M_2[x := N])$
- $\langle M_1, M_2\rangle[x := N] = \langle M_1[x := N], M_2[x := N]\rangle$
- $\mathsf{fst}(M)[x := N] = \mathsf{fst}(M[x := N])$
- $\mathsf{snd}(M)[x := N] = \mathsf{snd}(M[x := N])$

Now we introduce the notion of computation in the (typed) λ-calculus. This reduction rules are called *β-reduction.*

Definition 7.6.

$$\begin{aligned}(\lambda x.M)N &\xrightarrow{\beta} M[x := N]\\ \mathsf{fst}(\langle M, N\rangle) &\xrightarrow{\beta} M\\ \mathsf{snd}(\langle M, N\rangle) &\xrightarrow{\beta} N\end{aligned}$$

Note that β-reduction is closed under context:

$$M \xrightarrow{\beta} N \Longrightarrow \begin{cases} LM \xrightarrow{\beta} LN\\ ML \xrightarrow{\beta} NL\\ \lambda x.M \xrightarrow{\beta} \lambda x.N\\ \langle M, L\rangle \xrightarrow{\beta} \langle N, L\rangle\\ \langle L, M\rangle \xrightarrow{\beta} \langle L, N\rangle\\ \mathsf{fst}(M) \xrightarrow{\beta} \mathsf{fst}(N)\\ \mathsf{snd}(M) \xrightarrow{\beta} \mathsf{snd}(N)\end{cases}$$

$$\frac{}{x:\sigma\vdash x:\sigma}\ \text{ref}$$

$$\times \quad \frac{\Gamma\vdash M:\sigma \quad \Gamma\vdash N:\tau}{\Gamma\vdash\langle M,N\rangle:\sigma\times\tau}\ \text{pair} \qquad \frac{\Gamma\vdash M:\sigma\times\tau}{\Gamma\vdash \mathsf{fst}(M):\sigma}\ \text{fst} \quad \frac{\Gamma\vdash M:\sigma\times\tau}{\Gamma\vdash \mathsf{snd}(M):\tau}\ \text{snd}$$

$$\to \quad \frac{\Gamma, x:\sigma\vdash M:\tau}{\Gamma\vdash\lambda x.M:\sigma\to\tau}\ \text{abs} \qquad \frac{\Gamma\vdash M:\sigma\to\tau \quad \Gamma\vdash N:\sigma}{\Gamma\vdash MN:\tau}\ \text{app}$$

Figure 7.3.: Type Checking System

	introduction	*elimination*
	$\frac{}{F\vdash F}\ \mathsf{Ax}$	
$\wedge$	$\frac{\Gamma\vdash E \quad \Gamma\vdash F}{\Gamma\vdash E\wedge F}\ \wedge:\mathsf{i}$	$\frac{\Gamma\vdash E\wedge F}{\Gamma\vdash E}\ \wedge:\mathsf{e} \quad \frac{\Gamma\vdash E\wedge F}{\Gamma\vdash F}\ \wedge:\mathsf{e}$
$\vee$	$\frac{\Gamma\vdash E}{\Gamma\vdash E\vee F}\ \vee:\mathsf{i} \quad \frac{\Gamma\vdash F}{\Gamma\vdash E\vee F}\ \vee:\mathsf{i}$	$\frac{\Gamma\vdash E\vee F \quad \Gamma,E\vdash G \quad \Gamma,F\vdash G}{\Gamma\vdash G}\ \vee:\mathsf{e}$
$\to$	$\frac{\Gamma,E\vdash F}{\Gamma\vdash E\to F}\ \to:\mathsf{i}$	$\frac{\Gamma\vdash E \quad \Gamma\vdash E\to F}{\Gamma\vdash F}\ \to:\mathsf{e}$

Figure 7.4.: Minimal Logic Propositional Rules (Sequent Style)

7.4. The Curry-Howard Isomorphism

In this section we introduce the *Curry-Howard isomorphism* also know as the *Curry-Howard correspondence.* We start by presenting a basic type checking system for the typed λ-calculus in Figure 7.3

Note that the system presented in Figure 7.3 is closely related to the type checking system introduced in the lecture on functional programming, see [50, Chapter 9]. The only difference is that we have extended the rules (ref), (abs), and (app) as in [50, Chapter 9] by rules governing the product types. This is due to a different design choice in crafting our type system which is inessential, but simplifies the description of the Curry-Howard isomorphism.

We recall minimal logic, briefly introduced in Chapter 4.3. It is useful to change the style of presentation of these rules. For the sequel of this chapter we use the *sequent style form* to present these rules, see Figure 7.4. It is easy to see that these rules are equivalent to those natural deduction rules presented in Figure 11.2 and 11.3.

The crucial advantage of the presentation of natural deduction rules as in Figure 4.4 is the *direct* correspondence to the type checking system given in Figure 4.3. For instance the rule defining (app) in Figure 7.3 and implication elimination ($\to$: e) are essentially the same rule. More precisely, the type of a term in the type checking system corresponds to a formula in the natural

$$\vee \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{inl}(M) : \sigma + \tau} \qquad \frac{\Gamma \vdash N : \tau}{\Gamma \vdash \mathsf{inr}(N) : \sigma + \tau}$$

$$\frac{\Gamma \vdash M : \sigma + \tau \quad \Gamma, x : \sigma \vdash N_1 : \gamma \quad \Gamma, y : \tau \vdash N_2 : \gamma}{\Gamma \vdash \mathsf{case}\ M\ \mathsf{of}\ \ \mathsf{inl}(x) \longrightarrow N_1 \mid \mathsf{inr}(y) \longrightarrow N_2 : \gamma}$$

Figure 7.5.: Type Checking System (Part II)

deduction rules and vice versa. Observe the following correspondence:

(ref)	$\sim$	(Ax)
(abs)	$\sim$	$(\rightarrow: \mathsf{i})$
(app)	$\sim$	$(\rightarrow: \mathsf{e})$
(pair)	$\sim$	$(\wedge : \mathsf{i})$
(fst)	$\sim$	$(\wedge : \mathsf{e})$
(snd)	$\sim$	$(\wedge : \mathsf{e})$

In order to make this correspondence complete it suffices to give a simple extension of our type system by adding sum types. These correspond precisely to the natural deduction rules $(\vee : \mathsf{i})$ and $(\vee : \mathsf{e})$. This is the purpose of the rules given in Figure 7.5. The case of destructor encodes pattern matching. Based on these type checking rules we can complete the table above:

(inl)	$\sim$	$(\vee : \mathsf{i})$
(inr)	$\sim$	$(\vee : \mathsf{i})$
(case)	$\sim$	$(\vee : \mathsf{e})$

The here described correspondence between *types* and *formulas* is often referred to as "types as formulas" paradigm. We summarise the isomorphism in the following table:

formulas	$\sim$	types
proofs	$\sim$	programs
normalisation	$\sim$	computation

Above we have already seen the precise connection between formulas and types. What is missing is some intuition about the last correspondence: normalisation of proofs and computations in a (typed) λ-calculus. A detailed presentation of the relation is outside the scope of this lecture. But it is easy to sketch the idea. Consider the following proof Ψ in the type checking system:

$$\cfrac{\cfrac{\begin{matrix}\Pi_1 \\ \vdots \\ \Gamma \vdash M : \sigma\end{matrix} \quad \begin{matrix}\Pi_2 \\ \vdots \\ \Gamma \vdash N : \tau\end{matrix}}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}}{\Gamma \vdash \mathsf{fst}(\langle M, N \rangle) : \sigma}$$

If we conceive this proof as a natural deduction proof and focus on the formulas proven we observe that there exists some redundancy in this proof. Essentially we derive the "formula" σ by first introducing the "formula" $\sigma \times \tau$ and then eliminating $\times$ again. A shorter proof Ψ' is given below:

$$\begin{array}{c} \Pi_1 \\ \vdots \\ \Gamma \vdash M : \sigma \end{array}$$

We already know that this proof transformation is called *normalisation*, cf. Chapter 4.3.

If we now consider the terms occurring in these proofs we see that in the first proof Ψ the term $\mathsf{fst}(\langle M, N\rangle)$ is shown to be well-typed with type σ, while in the second proof Ψ' the term M is shown to have type σ. Thus the *normalisation* from Ψ to Ψ' directly corresponds the the β-reduction step $\mathsf{fst}(\langle M, N\rangle) \xrightarrow{\beta} M$ and thus to a *computation*. This correspondence between normalisation of proofs and computation in the typed λ-calculus holds in general. As another example we consider the β-reduction $(\lambda x.M)N \xrightarrow{\beta} M[x := N]$ together with the following proof normalisation:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1 \\ \vdots \\ \Gamma, x : \sigma \vdash M : \tau\end{array}}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \quad \begin{array}{c}\Pi_2 \\ \vdots \\ \Gamma \vdash N : \tau\end{array}}{\Gamma \vdash (\lambda x.M)N : \tau} \quad \Longrightarrow \quad \begin{array}{c}\Pi_1[x\backslash\Pi_2] \\ \vdots \\ \Gamma \vdash M[x := N]\end{array}$$

Here the right proof is to be understood as the extension of proof Π_1 by the inferences in Π_2 such that all occurrences of x in Π_1 are replaced by the term N.

Problem 7.1. Give a type system for the restriction of the λ-calculus, where all variables occur exactly once in the body of a term. More precisely we have the following terms: (i) variables x, (ii) abstractions $\lambda x.M$, and (iii) applications MN. For (ii) we have the constraint that $x \in \mathsf{FV}(M)$, and for (iii) that $\mathsf{FV}(M) \cap \mathsf{FV}(N) = \varnothing$. Furthermore, we define $\mathsf{FV}(x) := \{x\}$, $\mathsf{FV}(\lambda x.M) := \mathsf{FV}(M) - \{x\}$, and $\mathsf{FV}(MN) := \mathsf{FV}(M) \cup \mathsf{FV}(N)$.

Hint: Look-up linear logic

Problem 7.2. Consider the restriction of the λ-calculus defined in Problem 11.7. Prove strong normalisation of the calculus. (Hence also for the corresponding logic.)

8. Extensions of First-Order Logic

In this chapter we consider the limits of expressivity of first-order logic (see Section 8.1) and consider a specific extension of first-order logic: second-order logic (see Section 8.2). Finally, we conclude by mentioning a specific application of (second-order) logic to complexity theory. The complexity class P is captured by existential second-order logic on finite structures.

8.1. Limits of First-Order Logic

Let $\mathcal{G}$ be a directed graph with distinct nodes u, v. Recall that reachability in $\mathcal{G}$ is not expressible in first-order logic, that is, there is no formula $F(x, y)$ such that F holds in an interpretation with environment $\ell(x) = u$, $\ell(y) = v$ iff there exists a path in $\mathcal{G}$ from u to v. This formulation does not (yet) clarify, whether an (infinite) set of formulas $\mathcal{F}$ is sufficient to express reachability. In order to solve this issue, we introduce the notion of *elementary* and Δ-*elementary* collections of structures. Let $\mathcal{F}$ be a set of sentences (over some language $\mathcal{L}$), we define:

$$\mathsf{Mod}(\mathcal{F}) = \{\mathcal{A} \mid \mathcal{A} \text{ is a structure (of } \mathcal{L}\text{) and } \mathcal{A} \models \mathcal{F}\} \ .$$

We call $\mathsf{Mod}(\mathcal{F})$ the *class of models of* $\mathcal{F}$. Instead of $\mathsf{Mod}(\{F\})$ we simply write $\mathsf{Mod}(F)$.

Definition 8.1. Let $\mathcal{K}$ be a collection of structures.

– $\mathcal{K}$ is called *elementary* if there exists a sentence F such that $\mathcal{K} = \mathsf{Mod}(F)$.

– $\mathcal{K}$ is called Δ-*elementary* if there exists a set of sentences $\mathcal{F}$ such that $\mathcal{K} = \mathsf{Mod}(\mathcal{F})$.

Each elementary class is Δ-elementary. Moreover, every Δ-elementary class is the intersection of elementary classes:

$$\mathsf{Mod}(\mathcal{F}) = \bigcap_{F \in \mathcal{F}} \mathsf{Mod}(F) \ .$$

Reachability is not expressible in first-order logic, even with an infinite set of formuals. More precisely the class $\mathcal{K}_1$ of strongly connected graphs is not

Δ-elementary. Let $\mathcal{G}$ be a structure defined over the language $\mathcal{L} = \{R\}$ with the domain G. Here R is a binary relation symbol that represents the (directed) edge relation of the *graph* $\mathcal{G}$.

$\mathcal{G}$ is called *strongly connected* if for arbitrary, but distinct $u, v \in G$ there exists a path in $\mathcal{G}$ from u to v. For each number n, the regular polygon with $n+1$ nodes is denoted as $\mathcal{G}_n$. More precisely, we set $\mathcal{G}_n = (G_n, R^{\mathcal{G}_n})$, where $G_n = \{0, \dots, n\}$ and

$$R^{\mathcal{G}_n} := \{(i, i+1) \mid i < n\} \cup \{(n, 0)\} ,$$

while we define the following sentences ($n \in \mathbb{N}$):

$$F_n(a,b) := a = b \vee \exists x_1 \cdots \exists x_n \bigl(a = x_1 \wedge x_n = b \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_{n-1}, x_n)\bigr) .$$

Suppose, in order to derive a contradiction, that $\mathcal{K}_1 = \mathsf{Mod}(\mathcal{F})$ for set of sentences $\mathcal{F}$.

We set $\mathcal{H} := \mathcal{F} \cup \{\neg F_n \mid 2 \leqslant n\}$. Then it is easy to see that $\mathcal{H}$ is unsatisfiable as by assumption any model of $\mathcal{F}$ is a strongly connected graph, while the family of formulas $(\neg F_n)_{n \geqslant 2}$ can only be modelled if there exists at least two nodes which are not connected. However, each finite subset $\mathcal{F}'$ of $\mathcal{H}$ has a model. Namely there exists a number m such that $\mathcal{F}' \subseteq \mathcal{F} \cup \{\neg F_n \mid 2 \leqslant n \leqslant m\}$ and $\mathcal{G}_{2m} \models \mathcal{F}'$. For the latter observe that we can interpret the free variables a and b by 0 and m, respectively. This contradicts compactness.

8.2. Second-Order Logic

A *second-order* language extends a first-order language by a collection of *variables* for relations and functions. I.e., *variables* are:

(i) First-order variables, which are also called *individual variables.*

(ii) Relation variables with i arguments: $V^i_0, V^i_1, \dots, V^i_j, \dots$

(iii) Function variables with i arguments: $u^i_0, u^i_1, \dots, u^i_j, \dots$

Here $i = 1, 2, \dots$ and $j = 0, 1, 2, \dots$

Definition 8.2. *Second-order terms* are defined like first-order terms together with the following clause:

(iv) If $t_1, \dots, t_n$ are second-order terms, u an n-ary function variable, then $u(t_1, \dots, t_n)$ is a *second-order term.*

A second-order term *without* function variables is a first-order term.

Convention. The meta-symbols c, f, g, h, $\dots$, are used to denote constants and function symbols, while the meta-symbols u, v, w are used to denote function variables. P, Q, R, $\dots$, vary through predicate symbols or predicate variables. Individual variables are denoted as x, y, z, $\dots$, and predicate variables are denoted by V,X,Y,Z, etc.

Definition 8.3. *Second-order formulas* are defined like first-order formulas together with the following clauses:

(iv) If $t_1, \ldots, t_n$ are (second-order) terms, X an n-ary predicate variable, then $X(t_1, \ldots, t_n)$ is a *second-order formula.*

(v) If $A(f)$ is a second-order formula, f a function constant, u a function variable, such that $A(u)$ denotes the replacement of all occurrences of f by u, then

$$\forall u\ A(u) \qquad \exists u\ A(u)\ ,$$

are *second-order formulas.*

(vi) If $A(P)$ a second-order formula, P a predicate constant, X a predicate variable, then

$$\forall X\ A(X) \qquad \exists X\ A(X)\ ,$$

are *second-order formulas.*

A second-order formula *without* predicate and function variables is a first-order formula.

Definition 8.4. Let $\mathcal{A}$ denote a structure and A its domain. A *second-order environment* for $\mathcal{A}$ associates with any individual variable a an element in A, moreover with any n-ary function variable u a function $f\colon A^n \to A$ is associated and finally any n-ary relation variable X is assigned to a subset of A^n.

Let ℓ be a second-order environment and let $A' \subseteq A^n$ be an n-ary relation over A. Then we write $\ell\{X \mapsto A'\}$ for the environment mapping predicate variable X to the relation A' and all other variables $Y \neq X$ to $\ell(Y)$. A similar notion is used for function variables.

Based on the above extension of the notion of environment it is easy to define interpretations in the context of a second-order language. A *second-order interpretation* $\mathcal{I}$ is a pair $(\mathcal{A}, \ell)$ such that $\mathcal{A}$ is a structure and ℓ is a second-order environment. Thus the *value* of a second-order term t is defined as follows:

$$t^{\mathcal{I}} = \begin{cases} \ell(t) & \text{if } t \text{ an individual variable} \\ f^{\mathcal{A}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) & \text{if } t = f(t_1, \ldots, t_n),\ f \text{ a constant} \\ \ell(u)(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) & \text{if } t = u(t_1, \ldots, t_n),\ u \text{ a variable} \end{cases}$$

Definition 8.5. Let $\mathcal{I} = (\mathcal{A}, \ell)$ be a second-order interpretation, let A be the domain of $\mathcal{A}$, let F be a formula, and let A' be a relation. We write $\mathcal{I}\{X \mapsto A'\}$ as abbreviation for $(\mathcal{A}, \ell\{X \mapsto A'\})$.

We define the *satisfaction relation* $\mathcal{I} \models F$ as before, but add the following clauses:

$$\begin{array}{lll}
\mathcal{I} \models X(t_1,\dots,t_n) & :\Longleftrightarrow & \text{if } \ell(X) = P \subseteq A^n \text{ and } (t_1^{\mathcal{I}},\dots,t_n^{\mathcal{I}}) \in P \\
\mathcal{I} \models \forall X F(X) & :\Longleftrightarrow & \text{if } \mathcal{I}\{X \mapsto A'\} \models F(X) \text{ holds for all } A' \subseteq A^n \\
\mathcal{I} \models \exists X F(X) & :\Longleftrightarrow & \text{if } \mathcal{I}\{X \mapsto A'\} \models F(X) \text{ holds for some } A' \subseteq A^n \\
\mathcal{I} \models \forall u F(u) & :\Longleftrightarrow & \text{if } \mathcal{I}\{u \mapsto f\} \models F(u) \text{ holds for all } f \in A^n \to A \\
\mathcal{I} \models \exists u F(u) & :\Longleftrightarrow & \text{if } \mathcal{I}\{u \mapsto f\} \models F(u) \text{ holds for some } f \in A^n \to A
\end{array}$$

The next example shows that reachability (in a directed graph) becomes definable in second-order logic.

Example 8.1. Consider the following second order formula $F(x, y)$:

$$\begin{gathered}\exists P\big(\forall z_1\forall z_2\forall z_3\,(\neg P(z_1,z_1) \land (P(z_1,z_2) \land P(z_2,z_3) \to P(z_1,z_3))) \land \\ \land \forall z_1\forall z_2(P(z_1,z_2) \land \forall z_3(\neg(P(z_1,z_3) \land P(z_3,z_2))) \to R(z_1,z_2)) \land P(x,y)\big)\ .\end{gathered}$$

The idea of the formula is to assert the existence of a predicate P whose interpretation is that of a path in the graph. For that we assert with the first subformula that a path is transitive, but not reflexive. The second formula says that every direct successor in a path is connected by an edge in the graph. Finally, the last subformula expresses that the interpretations of x and y are connected.

It is not difficult to see that for any finite second-order model $\mathcal{G}$ of F with environment ℓ, there exists a path in $\mathcal{G}$ from $\ell(x)$ to $\ell(y)$.

While first-order logic features compactness, Löwenheim-Skolem, and completeness, none of these properties hold for second-order logic. This is summarised in the next theorem, whose proof we omit. The interested reader is kindly referred to [11] or [20].

Theorem 8.1. *(*i*) Compactness fails for second-order logic.*

*(*ii*) Löwenheim-Skolem fails for second-order logic.*

*(*iii*) Completeness fails for second-order logic, i.e., there does not exists a calculus that is sound and complete for second-order logic. In particular the set of valid second-order sentences is not recursively enumerable.*

8.3. Complexity Theory via Logic

In the remainder of this chapter we consider a specific application of the expressivity of second-order logic, namely the characterisation of the class NP of non-deterministic programs that run in polynomial time. For this purpose we suit the definition of problems to finite structures and state that a *complexity problem* denotes a (subset of a) set of finite structures. This re-formulation is standard, compare [44].

Definition 8.6. Let $\mathcal{K}$ be a set of *finite* structures (of a finite language $\mathcal{L}$) and let F be a sentence (of $\mathcal{L}$). Suppose $\mathcal{M}$ is a (second-order) structure in $\mathcal{K}$. Then the *F-$\mathcal{K}$ problem* asks, whether $\mathcal{M} \models F$ holds.

We call a second-order formula F *existential* ($\exists$SO for short) if F has the following form:

$$\exists X_1 \exists X_2 \cdots \exists X_n \; G ,$$

where G is essentially a first-order formula that may contain the free second-order variables $X_1, \dots, X_n$.

Let $\mathcal{K}$ be a set of finite structures and let $\mathcal{L}$ denote a finite language. Suppose F is a second-order sentence (of $\mathcal{L}$), i.e., no variable occurs free in F. The proof of the following lemmas can be found in [27].

Lemma 8.1. *If F is $\exists$SO, then the F-$\mathcal{K}$ problem is in* NP.

Lemma 8.2. *If F-$\mathcal{K}$ is decidable by a NTM M that runs in polynomial time then F is equivalent to an existential second-order sentence.*

Based on Lemma 8.1 and Lemma 8.2 we obtain the following characterisation theorem due to Fagin.

Theorem 8.2. *A sentence F (of $\mathcal{L}$) is equivalent to a sentence in $\exists$SO iff F-$\mathcal{K} \in \mathsf{NP}$. Moreover if F-$\mathcal{K} \in \mathsf{NP}$, then it can be assumed that the first-order part of F is a universal formula.*

Proof. Suppose F is an existential second-order sentence. Then by Lemma 8.1 the corresponding problem F-$\mathcal{K}$ is in NP. Conversely assume there exists a sentence F together with a set of structures $\mathcal{K}$ such that F-$\mathcal{K} \in \mathsf{NP}$. Then by definition of the complexity class NP there exists a TM (not necessarily deterministic) that runs in polynomial time and decides the F-$\mathcal{K}$ problem. Due to Lemma 8.2, F is equivalent to an $\exists$SO sentence G. Moreover it follows from the proof of Lemma 8.2 (see [27]) that the first-order part of G is universal. $\square$

As an easy corollary to this theorem we obtain an easy proof that the satisfiability problem of proposition logic (SAT for short) is complete for NP with respect to the polytime reducibility relation. (The interested reader is encouraged to compare the below given proof sketch to the standard argument, see for example [44].)

Corollary 8.1. SAT *is complete for* NP *(with respect to polytime reducibility).*

Proof. It is easy to see that $\mathsf{SAT} \in \mathsf{NP}$, as this is a consequence of Lemma 8.1. On the other hand consider any problem $A \in \mathsf{NP}$. Then we can reformulate the problem A as an F-$\mathcal{K}$ problem for some set of finite structures $\mathcal{K}$ and some sentence F. Due to Theorem 8.2 the sentence F is $\exists$SO and the first-order part of F is universal.

Let $\mathcal{M} \in \mathcal{K}$ be a finite model. In order to reduce the F-$\mathcal{K}$ problem (with respect to $\mathcal{M}$) to a SAT-problem, consider the finite (!) conjunction of all instances of the the first-order part of F, where we instantiate the bound variables by constants representing all elements in $\mathcal{M}$. We obtain a quantifier-free formula effectively forming a propositional logic formula, when we conceive the atomic formulas as propositional atoms.

It is not difficult to argue that any interpretation of F is conceivable as an assignment of this propositional formula, while on the other hand any assignment that satisfies the propositional formula can be re-interpreted as model of F.

In sum $\mathsf{SAT} \in \mathsf{NP}$ and any problem A in NP is reducible (with an algorithm that runs in polynomial time) to a SAT problem. Hence SAT is complete for NP. $\square$

The next corollary to Theorem 8.2 we state without proof.

Corollary 8.2. *The following is equivalent:*

- $\mathsf{NP} = \mathsf{co} - \mathsf{NP}$ *and*
- $\exists SO$ *is equivalent to (full) second-order logic.*

Problems

Problem 8.1. Let $\mathcal{K}$ be a Δ-elementary class of structures. Show that the subclass $\mathcal{K}^\infty \subseteq \mathcal{K}$ of structures in $\mathcal{K}$ with infinite domain is Δ-elementary, too.

Hint: Observe the difference between elementary and Δ-elementary class of structures.

Problem 8.2. Show Lemma 8.1.

Hint: Use non-determinism to simulate the effect of the existential quantifier.

Problem 8.3. Show Lemma 8.2.

Hint: Represent the Turing machine computation as a directed graph and use the construction in Example 8.1 to encode reachability in this graph.

Problem 8.4. Show that $\mathsf{SAT} \in \mathsf{NP}$, using the results of this chapter.

Hint: It suffices to formulate SAT as an F-$\mathcal{K}$ problem for a suitable class of structures $\mathcal{K}$ and an $\exists$SO sentence F.

Part II.

Automated Theorem Proving

9. Why Automated Resoning is Good For You

9.1. Program Analysis

Interesting properties of programs (like termination) are typically undecidable. Despite this limitation such properties are studied and automatic procedures have been designed to (partially) verify whether certain properties hold.

In the analysis of programs one doesn't study the concretely given program, but abstracts it in a suitable way, abstract interpretations [14] formalise this idea. Here the level of abstraction is crucial if one wants to prevent *false negatives*: properties that hold true for the program become false for the abstraction. In order to design expressive abstractions one combines simple abstractions into more complicated and thus more expressive ones.

Sumit Gulwani and Ashish Tiwari have presented a methodology to automatically combine abstract interpretations based on specific theories to construct an abstract interpreter based on the combination of the studied theories. This is encapsulated into the notion of *logical product* (compare [26]) and based on the Nelson-Oppen method for combining decision procedures of different theories (compare [38]). Here a *theory* is simply a set of sentences (over a given language) that is closed under logical consequence. Examples of theories would be for example the theory of linear arithmetic (making use of the symbols 0, 1, $+$, $\times$, $\leqslant$, and $=$) or the theory of lists (making use of the symbols car, cdr, cons, and $=$). If two theories T_1, T_2 fulfil certain conditions[1] and it is known that satisfiability of quantifier-free formulas with respect to the theories T_1 and T_2 is decidable, then satisfiability of quantifier-free formulas with respect to the union $T_1 \cup T_2$ is decidable. In Chapter 5 we study a related result, Robinson's joint consistency theorem.

The methodology invented in [26] allows the modularisation of the analysis of programs via abstract interpretations. Modularisation is possible for both stages of the analysis: One one hand the technique can be employed to define suitable interpretations for complex theories. On the other hand it can be employed to simplify the implementation of such an abstract interpreter.

[1] To be precise the theories T_1, T_2 are supposed to be *convex*, *disjoint*, and *stably infinite*, see [38].

9.2. Databases

Datalog is a database query language based on the logic programming paradigm. Syntactically it is a subset of Prolog (compare [12]). It is widely used in knowledge representation systems, see for example [22]. Logically a datalog query is a formula in Horn logic. Hence any such query has a unique model, its minimal model. This allows to assign a simple and unique semantics to datalog programs.

Datalog rules can be translated into inclusions in relational databases. Datalog extends positive relational algebras as recursive queries can be formed, which is not possible in positive relational algebras. The success of datalog can for example be witnessed in changes to the database query language SQL that has been extended by the possibility of recursive queries.

Contrary to full first-order logic, datalog queries are decidable. One can distinguish two notions of complexity in this context. On one hand we have *expression complexity*, where the complexity of fulfilling a given query is expressed in relation to the size of the query. On the other hand we have *data complexity*, where the complexity is measured in the size of the database and the query. The former notion is closely related to the notion of complexity of formal theories. Hence we focus on this notion. The expression complexity of datalog is $\mathsf{EXPTIME}$-complete, that is, far beyond the complexity of typical intractable problems like for example SAT.

Thomas Eiter et al. extended datalog to *disjunctive datalog*. Disjunctive datalog allows disjunctions in heads of rules (compare [21]). It is a strict extension of SQL and forms the basis of *semantic web* applications and has connections to *description logics* and *ontologies*. Disjunctive datalog queries can be extended with negation, so that the typical closed-world semantics of negation can be overcome. To indicate the expressivity of disjunctive datalog observe that the travelling salesperson problem can be directly formulated in this database query language. Disjunctive datalog remains decidable, but the expression complexity becomes $\mathsf{NEXPTIME}^{\mathsf{NP}}$-complete. This implies that such queries can be only solved on a nondeterministic Turing machine that runs in exponential time and employs an NP-oracle.

9.3. Issues of Security

Security protocols are small programs that aim at securing communications over a public network. The design of such protocols is difficult and error-prone.

In [40] Clifford Neuman and Stuart Stubblebine invented a key exchange protocol.[2] The goal of this protocol is to establish a secure key between two principals Alice and Bob that already share secure keys with a trusted third party. As shown by Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen in 1995 this protocol is not safe, but there exists a potential attack for a fourth person, such that the attacker can impersonate Alice and learn the shared key, while Bob believes this is the key of Alice

[2] `http://en.wikipedia.org/wiki/Neuman-Stubblebine_protocol`

(compare Chapter 12). It is relative simple to repair the protocol by putting type checks on the messages.

The potential attacks found by Hwang et al., where found manually, but they can also be detected automatically by formalising the protocol in first-order logic and employing an automated theorem prover. This observation is due to Christoph Weidenbach, see [55]. Not only is it possible to find the bug automatically, it is also possible to verify that the repaired protocol is now safe. Or to be more precise: safe against an intruder with the assumed capabilities.

9.4. Software Verification

As already mentioned above termination of programs is an undecidable property. Despite this negative result termination is a very active area in program analysis and in the last decade a number of techniques have been developed to analyse termination of a given program automatically. This is true for abstract program like *term rewrite system* (see [51]), but also for concrete programming languages like C or Java.

Here we focus on a short description of the program Terminator, developed by Byron Cook and others at the Microsoft Research laboratory at Cambridge University.[3] Terminator employs abstract interpretations and model-checking techniques to prove the termination of (concurrent) C-programs fully automatically.

In the early years of model checking mainly hardware was verified. During that time the research was driven by the need to prevent another design error like the one that lead to the costly Intel Pentium FDIV bug.[4] In the last decade the approach was extended to the verification of software, where initially only *safety properties* could be analysed. Such studies aim at verifying that a given program is safe with respect to a given specification, that is, nothing bad should happen in the program. Recently also *liveness properties* became of interest, that is, the specification represents a positive property and the program is checked against this positive specification.

Terminator makes use of model-checking to verify liveness properties of a given (concurrent) C-program P. As termination is a liveness property, termination of P can be established in the same way. The central idea is the automatic generation of *disjunctive well-founded transition invariants*. A binary relation R is called a transition invariant if the transitive closure of the transition relation $\to_{\mathsf{P}}$ (with respect to P) is contained in R. A relation R that is covered by finitely many well-founded relations $U_1, \dots, U_n$ is called disjunctive well-founded. The existence of a disjunctive well-founded transitive invariant for P is equivalent to termination of P. Transition invariants can be found automatically by exploiting abstract interpretations and other techniques in program analysis (compare [13]).

[3] http://research.microsoft.com/en-us/um/cambridge/projects/terminator/

[4] http://en.wikipedia.org/wiki/Pentium_FDIV_bug

10.

Towards Automated Reasoning for First-Order Logic

In this chapter we revisit early approaches in automated reasoning (see Section 10.1) and introduce a modern theory of automated reasoning. While it is in principle possible to automate proof search in a natural deduction calculus, as introduced in Section 4.3, such provers are rarely used in practise. To simplify the presentation we will first disregard languages containing the equality sign. (We know from earlier results that theoretically this is no restriction in power.)

We will introduce the *resolution calculus* in Section 10.2, a system of inference rules well-suited for automation. Furthermore, we will briefly study *tableaux calculi* (see Section 10.3) and continue this chapter with results on structural skolemisation (Section 10.4) as well as redundancy criteria and deletion (Section 10.5). The here introduced automated techniques are easily powerful enough to show the validity of the semantic entailment (1.1) mentioned in Chapter 9.

10.1. Early Approaches in Automated Reasoning

As detailed in Chapter 6 Herbrand's Theorem yields a reduction of validity of a first-order existential formula $\exists x F(x)$ to the validity of a finite disjunction $F(t_1) \vee \cdots \vee F(t_n)$ of instances of $\exists x F(x)$, where the t_i are arbitrary (ground) terms over the base language. In the context of automated reasoning, one typically phrases this result in terms of (un)satisfiability as in Theorem 6.3. In particular a universal formula $\forall x F(x)$ is unsatisfiable, if there exists a finite set of ground instances $\{F(t_1), \ldots, F(t_n)\}$ such that this set is unsatisfiable, cf. Corollary 6.2.

Based on this observation, Paul Gilmore wrote the first automated theorem prover for first-order logic [25]. The basic idea is quite simple. Consider an arbitrary first-order sentence F (over language $\mathcal{L}$) and transforms its negation into Skolem normal form, so that $\neg F$ is equivalent for satisfiability to a formula F' of the form

$$\forall x_1 \cdots \forall x_n G(x_1, \ldots, x_n) \ ,$$

```
contr := false;
n := 0;
while (¬ contr) do {
  $D'$ := DNF($\mathcal{C}'_n$);
  contr := all constituents of $D'$
  contain complementary literals;
  n := n + 1;
}
```

Figure 10.1.: Gilmore's Prover in Pseudo-Code

where G is in CNF. Then one considers all possible Herbrand interpretations of $\mathcal{L}$ and F is valid if there exists a finite unsatisfiable subset S of ground instances of F'.

The question is of course how to *find* a suitable set S. In Chapter 6 this was achieved by the method of *semantic trees*. Here we describe more closely what constitutes Gilmore's prover. For that we informally conceive the above subformula $G(x_1, \ldots, x_n)$ as a set of clauses $\mathcal{C}$ and make use of the following definition.[1] The *Herbrand universe* for a language $\mathcal{L}$ can be constructed iteratively as follows:

$$H_0 := \begin{cases} \{c \mid c \text{ is a constant in } \mathcal{L}\} & \exists \text{ constants in } \mathcal{L} \\ \{c\} & \text{otherwise} \end{cases}$$
$$H_{n+1} := \{f(t_1, \ldots, t_k) \mid f^k \in \mathcal{L}, t_1, \ldots, t_k \in H_n\}$$

Finally $H := \bigcup_{n \geqslant 0} H_n$ denotes the *Herbrand universe* for $\mathcal{L}$. This stratification of the Herbrand universe for $\mathcal{L}$ allows the definition of suitable ground instances of the clause set $\mathcal{C}$: define $\mathcal{C}'_n$ as the ground instances of $\mathcal{C}$ using only terms from H_n. The resulting theorem prover is given in Figure 10.1, where "DNF($\mathcal{C}'_n$)" denotes a method to transform the clauses in the clause set $\mathcal{C}'_n$ into disjunctive normal form. The idea being that (un)satisfiablity of a DNF can be checked in $\mathrm{O}(n \log n)$ time, where n is the length of the formula.

It is clear that Gilmore's prover is sound and complete for first-order logic. However, the following disadvantes are striking:

– generation of all $\mathcal{C}'_n$

– transformation to DNF

While the former point is intrinsic to any direct application of Herbrand's theorem, the second point is an unnecessary bottleneck. Clearly for any CNF, transformation into DNF almost always takes exponential time. Thus, it should not come as a surprise that Gilmore's prover did not yield actual proofs of even simple (predicate logic) formulas.

[1] We refer to clauses here, as it simplifies the presentation of Gilmore's prover. Clause logic will be formally introduced in the next section.

To overcome the mentioned bottleneck, Davis and Putnam developed an original method for satisfiability checking of ground clauses [16, 17]. We start with a simple definition.

Definition 10.1. A clause C is called *reduced*, if every literal occurs at most once in C. Furthermore, a clause set $\mathcal{C}$ is called *reduced for tautologies*, if every clause in $\mathcal{C}$ is reduced and does not contain complementary literals.

The rules of the *Davis*, *Putnam*, *Logemann*, and *Loveland* (*DPLL* for short) method are given as follows.[2]

Definition 10.2. Let $\mathcal{C}'$ denote a reduced ground set of clauses.

(i) *Tautology rule*: delete all clauses containing complementary literals. (For the subsequent rules it is assumed that $\mathcal{C}'$ doesn't contain tautologies.)

(ii) *One-literal rule*: let $C \in \mathcal{C}'$ and suppose C consists of just one literal L. Then remove all clauses $D \in \mathcal{C}'$ such that L occurs in D. Furthermore remove $\neg L$ from all remaining clauses in $\mathcal{C}'$.

(iii) *Pure literal rule*: let $\mathcal{D}' \subseteq \mathcal{C}'$ such that there exists a literal L that appears in all clauses in $\mathcal{D}'$ but $\neg L$ doesn't appear in $\mathcal{C}'$. Then replace $\mathcal{C}'$ by $\mathcal{C}' \setminus \mathcal{D}$.

(iv) *Splitting rule*: suppose the clause set $\mathcal{C}$ can be written as follows:

$$\mathcal{C}' = \{A_1, \dots, A_n, B_1, \dots, B_m\} \cup \mathcal{D} \ ,$$

where there exists a literal L, such that neither L nor $\neg L$ occurs in $\mathcal{D}$. Furthermore L occurs in any A_i (but in no B_j) and $\neg L$ occurs in any B_j (but in no A_i). The rule consists in splitting $\mathcal{C}'$ into $\mathcal{C}'_1 := \{A'_1, \dots, A'_n\} \cup \mathcal{D}$ and $\mathcal{C}'_2 := \{B'_1, \dots, B'_m\} \cup \mathcal{D}$, where A'_i is the result of deleting L from A_i and B'_j is the result of deleting $\neg L$ from B_j.

The proof of the next theorem is easy and hence omitted.

Theorem 10.1. *The rules of the DPLL-method are correct. In particular this means that if $\mathcal{D}$ is a set of ground clauses and either $\mathcal{D}'$ or $\mathcal{D}_1$ and $\mathcal{D}_2$ are obtained by the above rules, then $\mathcal{D}$ is satisfiable if $\mathcal{D}'$ ($\mathcal{D}_1$ or $\mathcal{D}_2$) is satisfiable.*

It is not difficult to see that the above rules constitutes an abstraction of the semantic tree method. In particular the splitting rule is reminiscent of the branching in a semantic tree. Furthermore, the efficiency of the method stems from the fact that it is *reductive*: any rule shrinks the newly considered clause set(s). To picture this reduction, one defines *DPLL-trees* T inductively as follows. Let $\mathcal{C}'$ be a set of reduced ground clauses.

– The tree T which consists only of the root, labelled by $\mathcal{C}'$, is a DPLL-tree.

– Let N be a node in T, labelled by $\mathcal{D}$. Then N is either a

[2] A word of warning: we already use the later developed notion for clause logic, in particular we write L and $\neg L$ for complementary pairs of literals.

(i) leaf node,

(ii) N has one successor N', labelled by $\mathcal{D}'$, where $\mathcal{D}'$ is obtained as the application of either a tautology, one-literal, or pure literal rule to $\mathcal{D}$, or

(iii) N has two successors N_1, N_2 labelled by the clause sets obtained by an application of the split rule to $\mathcal{D}$.

A DPLL-tree for $\mathcal{C}'$ is called a *DPLL-decision tree* or simply a *decision tree for* $\mathcal{C}'$ if either all leafs are labelled by the empty clause $\Box$, or there exists a leaf labelled by the empty clause set $\varnothing$. Let T be a decision tree for $\mathcal{C}'$. We say T proves the satisfiability of $\mathcal{C}'$ if there exists a leaf labelled by $\varnothing$. Otherwise, if all leaves are labelled with $\Box$, T proves the unsatisfiability of $\mathcal{C}'$. The next theorem states correctness of the DPLL-method. Again the proof is easy, and thus omitted.

Theorem 10.2 (Correctness)**.** *Let $\mathcal{C}'$ be a reduced set of ground clauses and let T be a decision tree proving satisfiability or unsatisfiability for $\mathcal{C}'$. Then $\mathcal{C}'$ is satisfiable or unsatisfiable, respectively.*

In order to obtain completeness of the method, we prove the following stronger assertion. Note that completeness essentially states that any clause set $\mathcal{C}'$ admits a decision tree. The below theorem also shows that any strategy in applying the above rules yields a decision tree.

Theorem 10.3 (Strong Completeness)**.** *Let $\mathcal{C}'$ be as above and let T be a DPLL-tree for $\mathcal{C}'$. Then T can be extended to a decision tree for $\mathcal{C}'$.*

Proof. We proceed by induction on the number ℓ of atoms in $\mathcal{C}'$. For $\ell = 0$ the assertion is easy as $\mathcal{C}'$ can then either be empty or just contains the empty clause. In both cases T is already a decision tree. Hence we can assume $\ell > 0$. We distinguish the following cases:

(i) T consists only of the root, labelled by $\mathcal{C}'$. Then $\mathcal{C}'$ contains at least one literal and we can either employ a one-literal or pure literal rule, or a splitting rule. In either case we can extend T such that the successors nodes are labelled with smaller clause sets. Thus induction hypothesis applies to construct a decision tree that extends T.

(ii) T contains more than one node. Then let $\mathcal{D}_1, \dots, \mathcal{D}_n$ denote all leaf nodes of T. Arguing exactly as in the first case we find an extension of T that forms a decision tree for $\mathcal{C}'$.

$\Box$

Note that we can apply the tautology rule as a prepprocessing steps as tautologies cannot be introduced by the other rules. Hence the DPLL-method gives rise to the following decision procedure for sets of ground clauses $\mathcal{C}'$:

DPLL(a) Remove multiple occurrences of literals in $\mathcal{C}'$ to obtain a reduced clause set $\mathcal{D}_1$.

```
if $\mathcal{C}$ does not contain function symbols
then apply DPLL(a)–DPLL(c) on $\mathcal{C}'_0$
else {
  n := 0;
  contr := false;
  while (¬ contr) do {
    apply DPLL(a)–DPLL(c) on $\mathcal{C}'_n$;
    if the decision tree proves unsatisfiability ,
    then contr := true
    else contr := false;
    n := n + 1;
  }
}
```

Figure 10.2.: Davis-Putnam-Logemann-Loveland Method in Pseudo-Code

DPLL(b) Apply the tautology rule exhaustively to $\mathcal{D}_1$ to obtain a reduced clause set $\mathcal{D}_2$ that is reduced for tautologies.

DPLL(c) Construct a decision tree for $\mathcal{D}_2$.

It is easy to see that the construction of a decision tree in the last step is in the worst-case exponential in the number of atoms in $\mathcal{D}_2$. If we pluck this satisfiability check into Gilmore's prover, we obtain the refined theorem prover for first-order logic presented in Figure 10.2. The prover excepts a clause set $\mathcal{C}$ as input.

The DPLL-method is not competitive with more advanced methods in automated reasoning for first-order logic. However the method still forms a very efficient method for SAT-solving.

10.2. Resolution for First-Order Logic

In Chapter 2 we introduced resolution for propositional logic. In this section we extend this calculus to first-order logic. For that we restrict the syntax of first-order logic. This restricted language is sometimes called *(first-order) clause logic*. As in Section 3.1 our language consists of *constants*, *variables*, *logical symbols*, and other auxiliary symbols. In particular we have individual constants $k_0, k_1, \dots, k_j, \dots$, function constants (with i arguments) $f^i_0, f^i_1, \dots, f^i_j, \dots$ and predicate constants (with i arguments) $R^i_0, R^i_1, \dots, R^i_j, \dots$ In addition to these constants we make use of variables: $x_0, x_1, \dots, x_j, \dots$ We collect the (infinite) set of variables as $\mathcal{V}$.

Convention. The meta-symbols c, f, g, h, ..., are used to denote constants and function symbols, while the meta-symbols P, Q, R, ..., vary through predicate symbols. Variables are denoted by a, b, ... or we use x, y, z, and so forth.

The most noticeable restriction to our earlier used languages is the restriction of the logical symbols to $\neg$ and $\vee$. Note that in such a restricted language the notion of a *term* or *atomic formula* is still meaningful. However, we can not really speak of first-order *formula* of this language, simply because elementary logical symbols, in particular quantifiers, are missing. We will see shortly how to overcome this restriction.

Definition 10.3. If $t_1, \ldots, t_n$ denote terms, and P denotes an n-placed predicate constant, then $P(t_1, \ldots, t_n)$ is called an *atomic formula*. A *literal* is an atomic formula or its negation. A *clause* is a disjunction of literals.

Let C be a clause. We write $\mathcal{V}\mathsf{ar}(C)$ for the set of variables (from $\mathcal{V}$) that occur in C. Let $\mathcal{L}$ denote a standard first-order language (as defined in Section 3.1) and let $\mathcal{L}'$ be the restriction of $\mathcal{L}$ according to the above settings. Let F be a sentence (of $\mathcal{L}$). Due to Theorem 6.2 there exists a sentence G in SNF such that $F \approx G$. By definition G is an universal formula, whose matrix is in conjunctive normal from and for wlog., we can suppose G has the following shape:

$$\forall x_1 \cdots \forall x_n \ (H_1(x_1, \ldots, x_n) \wedge \cdots \wedge H_m(x_1, \ldots, x_n)) \ ,$$

where each H_i $(i = 1, \ldots, m)$ is a disjunction of literals. Thus each H_i is actually a clause and we can represent G as a set $\mathcal{C}$ of clauses. This set is called *clause form* of G (and of F).

Theorem 10.4. *For any first-order sentence F (of $\mathcal{L}$) there exists a computable set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ (of $\mathcal{L}'$) such that $F \approx \forall x_1 \cdots \forall x_n (C_1 \wedge \cdots \wedge C_m)$.*

Proof. The theorem follows from the considerations above. $\square$

In order to make the clause form $\mathcal{C}$ unique for a given formula F, we fix the specific transformation steps applied to obtain $\mathcal{C}$. Thus we can speak of *the* clause form $\mathcal{C}$ of F. The next definition fixes the representation of clauses we will use in the sequel, compare also the corresponding definition given in Section 2.3.

Definition 10.4. We define a *clause* inductively.

(i) $\square$ is a *clause* (the *empty clause*),

(ii) literals are *clauses*, and

(iii) if C, D are clauses, then $C \vee D$ is a *clause*.

When speaking about clauses, we use the equivalences $A \equiv \neg\neg A$, where A denotes an atomic formula. Moreover disjunction $\vee$ is associative and commutative. In addition we define the following identities: $\square \vee \square = \square$ and $C \vee \square = \square \vee C = C$, where C is an arbitrary clause.

Let $\mathcal{T}$ denote the set of terms in our language. Terms are denoted by $s, t, u, v, w, \ldots$ A *substitution* σ is a mapping $\mathcal{V} \to \mathcal{T}$, such that $\sigma(x) = x$, for almost all x. Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, the empty substitution is denoted by ϵ. We call the set $\mathsf{dom}(\sigma) = \{x \mid \sigma(x) \neq x\}$ the *domain* of σ. The set $\mathsf{rg}(\sigma) = \{\sigma(x) \mid x \in \mathsf{dom}(\sigma)\}$ is called the *range* of σ. $\mathcal{V}\mathsf{ar}(\mathsf{rg}(\sigma))$ is abbreviated by $\mathsf{vrg}(\sigma)$. A substitution σ is called *ground* if $\mathsf{vrg}(\sigma) = \varnothing$.

For a given expression E the application of a substitution σ to E is denoted as $E\sigma$; $E\sigma$ is called an *instance* of E. The composition of substitutions $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, $\tau = \{y_1 \mapsto r_1, \dots, y_1 \mapsto r_m\}$ (denoted as $\sigma\tau$) is defined as follows:

$$\{x_1 \mapsto t_1\tau, \dots, x_n \mapsto t_n\tau\} \cup \{y_i \to r_i \mid \text{for all } j = 1, \dots, n,\ y_i \neq x_j\} \ .$$

A substitution σ is *more general* than a substitution τ, if there exists a substitution ρ such that $\sigma\rho = \tau$.

Definition 10.5. A *unifier* σ of expressions E and F is a substitution such that $E\sigma = F\sigma$. A unifier σ is *most general* if σ is more general than any other unifier (of E, F). Unifiers and most general unifiers naturally generalise to sets of expressions.

The sequence $E = u_1 \overset{?}{=} v_1, \dots, u_n \overset{?}{=} v_n$ is called an *equality problem.* Here u_i, v_i denotes either terms or atomic formulas. The unifier of an equality problem $E = x_1 \overset{?}{=} v_1, \dots, x_n \overset{?}{=} v_n$ is defined as the unifier of the set $\{u_1 = v_1, \dots, u_n = v_n\}$. The rules in Figure 10.3 define a simple, rule based unification algorithm that acts on equality problems.

$$u \overset{?}{=} u, E \Rightarrow E$$

$$f(s_1, \dots, s_n) \overset{?}{=} f(t_1, \dots, t_n), E \Rightarrow s_1 \overset{?}{=} t_1, \dots, s_n \overset{?}{=} t_n, E$$

$$f(s_1, \dots, s_n) \overset{?}{=} g(t_1, \dots, t_m), E \Rightarrow \perp \quad \text{if } f \neq g$$

$$x \overset{?}{=} v, E \Rightarrow x \overset{?}{=} v, E\{x \mapsto v\} \quad \text{if } x \in \mathcal{V}\mathsf{ar}(E),\ x \notin \mathcal{V}\mathsf{ar}(v)$$

$$x \overset{?}{=} v, E \Rightarrow \perp \quad \text{if } x \neq v,\ x \in \mathcal{V}\mathsf{ar}(v)$$

$$v \overset{?}{=} x, E \Rightarrow x \overset{?}{=} v, E \quad \text{if } v \notin \mathcal{V}$$

For brevity the symbol f and g may either denote a function or a predicate constant.

Figure 10.3.: Rule Based Standard Unification

If $E = x_1 \overset{?}{=} v_1, \dots, x_n \overset{?}{=} v_n$, with x_i pairwise distinct and $x_i \notin \mathcal{V}\mathsf{ar}(v_j)$, for all i, j, then E is an equality problem in *solved form.* An equality problem $E = x_1 \overset{?}{=} v_1, \dots, x_n \overset{?}{=} v_n$ in solved form *induces* the substitution $\sigma_E := \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$.

Theorem 10.5. *An equality problem E is unifiable iff the unification algorithm of Figure 10.3 stops with a solved form. Moreover if $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is a most general unifier (*mgu *for short) of E.*

Proof. It suffices to verify the following three properties:

(i) If $E \Rightarrow E'$, then σ is a unifier of E iff σ is a unifier of E'.

(ii) If $E \Rightarrow^* \bot$, then E is not unifiable.

(iii) If $E \Rightarrow^* E'$ such that E' is a solved form, then $\sigma_{E'}$ is a mgu of E.

The first item follows by case distinction on each rule. The remaining items are consequences of the first, together with the fact that if $E = x_1 \overset{?}{=} v_1, \dots, x_n \overset{?}{=} v_n$ is a solved form, then the induced substitution $\sigma_E = \{x_1 \mapsto v_1, \dots, x_n \mapsto v_n\}$ is a mgu of E. $\square$

It is not difficult to see that the algorithm terminates, but may produce exponentially large terms. Now, we are ready to state the two *inference rules* of the resolution calculus in Figure 10.4. In the application of these inferences, we can always assume that the premises are variable disjoint. Otherwise, we make them variable disjoint by renaming variables consistently.

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad\qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

Here σ is a mgu of the atomic formulas A and B. The first inference is called *resolution*, while the second one is called *factoring*.

Figure 10.4.: Resolution Calculus

Remark. Observe that the factoring rule is only defined for *atoms* A and B, that is, factoring is restricted to positive literals. In this sense factoring defined as in Figure 10.4 is more restrictive than the definition in Chapter 2.

The next definition lifts Definition 2.7 to first-order logic, or more precisely to first-order clause logic.

Definition 10.6. Let $\mathcal{C}$ be a set of clauses. We define the *resolution operator* $\mathsf{Res}(\mathcal{C})$ as follows:

$$\mathsf{Res}(\mathcal{C}) = \left\{ D \mid \begin{array}{l} D \text{ is conclusion of an inference in Figure 10.4 with premises} \\ \text{in } \mathcal{C} \end{array} \right\} .$$

We define $\mathsf{Res}^0(\mathcal{C}) := \mathcal{C}$ and $\mathsf{Res}^{n+1}(\mathcal{C}) := \mathsf{Res}^n(\mathcal{C}) \cup \mathsf{Res}(\mathsf{Res}^n(\mathcal{C}))$. Finally, we set: $\mathsf{Res}^*(\mathcal{C}) := \bigcup_{n \geqslant 0} \mathsf{Res}^n(\mathcal{C})$. We say the empty clause is derivable from $\mathcal{C}$ if $\square \in \mathsf{Res}^*(\mathcal{C})$.

Recall that if $\mathsf{Res}(\mathcal{C}) \subseteq \mathcal{C}$, then the clause set $\mathcal{C}$ is called *saturated.* Obviously, we have that $\mathsf{Res}^*(\mathcal{C})$ is saturated. If for a clause D, $D \in \mathsf{Res}^*(\mathcal{C})$, then we say that D is *derived* from $\mathcal{C}$ by resolution. If for a clause set $\mathcal{C}$, $\square \notin \mathsf{Res}^*(\mathcal{C})$, then $\mathcal{C}$ is called *consistent.*

Theorem 10.6. *Resolution is sound. Moreover let F be a sentence and $\mathcal{C}$ its clause form such that $\square \in \mathsf{Res}^*(\mathcal{C})$. Then F is unsatisfiable.*

Sketch of Proof. We only sketch the proof of the theorem, see [36] for a complete proof. Similar to the proof of the soundness theorem for natural deduction, we have to verify that every inference rule of the resolution calculus is sound. For this one shows that if the assumptions of resolution and factoring are modelled by an interpretation $\mathcal{M}$, then the consequence (of the rule) holds in $\mathcal{M}$ as well. $\square$

In order to proof completeness of resolution, we make use of the following lemmas.

Lemma 10.1. *Let S denote the set of all consistent ground clause sets. A clause is called* ground *if it doesn't contain variables. Then S has the satisfaction properties.*

Proof. As the syntax of clause logic is restricted, it suffices to verify the properties (i)–(iv) of the satisfaction properties. The other properties are trivially satisfied.

We exemplarily consider property (iv) and suppose there exists ground clauses E, F such that $E \vee F \in \mathcal{C}$ for $\mathcal{C} \in S$. We have to show that either $\mathcal{C} \cup \{E\}$ or $\mathcal{C} \cup \{F\}$ is consistent. Assume to the contrary that $\mathcal{C} \cup \{E\}$ and $\mathcal{C} \cup \{F\}$ are not consistent. This implies that $\square \in \mathsf{Res}^*(\mathcal{C} \cup \{E\})$ and $\square \in \mathsf{Res}^*(\mathcal{C} \cup \{F\})$. We name the first derivation of $\square$ from $\mathcal{C} \cup \{E\}$ by D_1 and the second derivation of $\square$ from $\mathcal{C} \cup \{F\}$ is denoted as D_2.

As $\mathcal{C}$ is free of variables, these proofs are free of variables, too. Thus we take the derivation D_1 and replace in this derivation the clause E with the clause $E \vee F$. The result will be a valid derivation of clause F in the resolution calculus: the only condition in each inference that could possibly be affected is the condition on the unifiers. However, as all clauses are ground, this does not cause any problems. Thus we obtain a derivation D of the clause F from the set of clauses $\mathcal{C}$. Now, we consider the derivation D_2 of $\square$ from $\mathcal{C} \cup \{F\}$. We transform D_2 as follows: at any position in the proof, where the clause F is used, the derivation D is used instead. In sum, we obtain a derivation of the empty clause from the set of clauses $\mathcal{C}$. This contradicts the assumption. $\square$

The next two lemma allow us to lift a ground resolution derivation to the general level. The lemmas follow essentially by definition of a most general unifier, cf. Definition 10.5. See [36] for a complete proof.

Lemma 10.2 (Lifting Lemma)**.** *A* ground *substitution is a substitution whose range contains only terms without variables. Let τ_1 and τ_2 be a ground substitutions and consider the following ground resolution step:*

$$\frac{C\tau_1 \vee A\tau_1 \quad D\tau_2 \vee \neg B\tau_2}{C\tau_1 \vee D\tau_2} \ ,$$

where $A\tau_1 = B\tau_2$. Then there exists a mgu σ of A and B, such that σ is more general then τ_1 and τ_2 and the following resolution step is valid:

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \ .$$

Lemma 10.3 (Lifting Lemma)**.** *Let τ be a ground substitution and consider the following ground factoring step:*

$$\frac{C\tau \lor A\tau \lor B\tau}{C\tau \lor A\tau} \quad ,$$

where $A\tau = B\tau$. Then there exists a mgu σ, such that σ is more general then τ and the following factoring step is valid:

$$\frac{C \lor A \lor B}{(C \lor A)\sigma}$$

Our completeness proof for resolution follows the pattern of the proof of the completeness theorem for natural deduction, that is, we want to apply the model existence theorem in conjunction with Lemma 10.1. However, we have not yet proven the model existence theorem in the full generality that is required here. The arguments given upto now restricted base language such that function constants were not allowed. This is not a major restriction in the context of first-order logic, but it is a rather strong restriction in the context of clause logic, as the latter depends on Skolemisation. We recall the crucial lemma (compare Chapter 4.3) and extend it suitably to the current context.

Lemma 10.4. *Let $\mathcal{G}$ be a set of formulas (of $\mathcal{L}$) admitting the closure properties. Suppose that $\mathcal{L}$ is free of the equality symbol. Then there exists an interpretation $\mathcal{M}$ such that every element of the domain of $\mathcal{M}$ is the denotation of a term (of $\mathcal{L}^+$) and $\mathcal{M} \models \mathcal{G}$.*

Proof. Based on the proof of Lemma 4.4 it suffices to extend the definition of the *Herbrand model* $\mathcal{M}$ of $\mathcal{G}$ as follows. Let $t_1, \dots, t_n$ denote elements of $\mathcal{M}$ and f an n-ary function symbol in $\mathcal{L}^+$. We define:

$$f^{\mathcal{M}}(t_1, \dots, t_n) := f(t_1, \dots, t_n) \ .$$

Following the argument given in the proof of Lemma 4.4 it is an easy exercise to verify that $\mathcal{M} \models \mathcal{G}$ holds. □

Theorem 10.7. *Resolution is complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then $\Box \in \mathsf{Res}^*(\mathcal{C})$ if F is unsatisfiable.*

Proof. If F is unsatisfiable then due to Corollary 6.2 there exists a set of ground clauses $\mathcal{C}'$ that are instances of the clauses in $\mathcal{C}$ such that $\mathcal{C}'$ is unsatisfiable.

Suppose $\Box \not\in \mathsf{Res}^*(\mathcal{C}')$. Then by definition the clause set $\mathsf{Res}^*(\mathcal{C}')$ is saturated and thus consistent. By the model existence theorem in conjunction with Lemma 10.1 we conclude that $\mathcal{C}'$ is satisfiable. This is a contradiction to our assumption. Hence $\Box \in \mathsf{Res}^*(\mathcal{C}')$.

It remains to lift this derivation of the empty clause from $\mathcal{C}'$ to a derivation of the empty clause from the original set of clauses $\mathcal{C}$. This is possible due to the lifting lemmas, Lemmas 10.2 and 10.3. □

10.3. Tableaux Provers

In this section we briefly present the method of semantic tableaux, first for the propositional case, then for the case for first-order logic. The insight we obtain here is that semantic tableaux (without further restrictions) is very close to natural deduction.

10.3.1. Propositional Semantic Tableaux

In the literature on tableaux it is customary to categories formulas according to a uniform notation due to Smullyan [48]. Following this tradition, we group all formulas of the form $A \odot B$ and $\neg(A \odot B)$ into those that act *conjunctively*, which are called *α-formulas*, and those that act *disjunctively*, called *β-formulas*. The components of these formulas are denoted as α_i $(i = 1, 2)$ and β_j $(j = 1, 2)$ respectively. See Figure 10.5 for this categorisation.

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$A \wedge B$	A	B	$\neg(A \wedge B)$	$\neg A$	$\neg B$
$\neg(A \vee B)$	$\neg A$	$\neg B$	$A \vee B$	A	B
$\neg(A \to B)$	A	$\neg B$	$A \to B$	$\neg A$	B

Figure 10.5.: Uniform Notation for Propositional Connectives

Similar to resolution, semantic tableaux is a refutation based technique, but in contrast to resolution it does not presuppose conjunctive normal form. Instead the *expansion rules* given in the next definition act on arbitrary formulas.

Definition 10.7 (Propositional Expansion Rules)**.**

$$\frac{\neg\neg A}{A} \qquad\qquad \frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad\qquad \frac{\beta}{\beta_1 | \beta_2}$$

Definition 10.8. Let $\{A_1, \dots, A_n\}$ be a set of propositional formulas

– The following one-branch tree T is a *tableau* for $\{A_1, \dots, A_n\}$:

$$\begin{array}{c} A_1 \\ A_2 \\ \vdots \\ A_n \end{array}$$

– Suppose T is a tableau for $\{A_1, \dots, A_n\}$ and T^* is obtained by applying a tableau expansion rule to T, then T^* is a *tableau* for $\{A_1, \dots, A_n\}$.

Definition 10.9. A branch is *closed* if the formulas F and $\neg F$ occur on it; if F is atomic, then the branch is said to be *atomically closed.* A tableau is *closed* if every branch is closed. A *tableau proof* of F is a closed tableau for $\{\neg F\}$. In a *strict* tableau no formula is expanded twice on the same branch.

Theorem 10.8. *Let F be a formula. The tableau procedure is* sound *and* complete *for propositional logic, that is, the following holds:*

$$F \text{ is a tautology} \iff F \text{ has a tableau proof}.$$

Despite the non-surprising assertion of the above theorem, cf. Theorems 2.1 and 2.2, we will prove it, as this allows us to present a simple completeness argument which easily extends to a restricted calculus. In order to show soundness it suffices to prove the following lemma, which we leave as exercise to the reader.

Lemma 10.5. *Any application of a tableau expansion rule to a satisfiable tableau yields another satisfiable tableau.*

In the remainder of the section, we consider completeness. We make the following refinement to our tableau procedure. First of all, we only consider *atomically closed* tableau proofs. Moreover, we demand that the constructed tableau proofs are *strict*: a tableau expansion rule can be applied to a formula on a branch at most once. Our completeness argument will work for this restricted calculus. As exercise we leave the completeness proof for unrestricted tableua, as straightforward adaption of the model existence theorem, cf. Problem 10.8.

Lemma 10.6. *Suppose F is a valid formula. A strict tableau construction for $\{\neg F\}$ that is continued as long as possible must terminate in an atomically closed tableau.*

Proof. First observe that any strict tableau construction for $\{\neg F\}$ has to terminate as each expansion rule reduces the logical complexity of the formulas. To show the lemma, we argue indirectly. Suppose T is a strict tableau for $\{\neg F\}$ that is not atomically closed. Then there exists a branch in T which does not contain conflicting literals. Furthermore all possible expansion rules have been applied on the non-literal formulas on this branch. Hence we can read-off an assignment for the atoms in $\neg F$ from this branch. This contradicts the assumption that F is a tautology. $\square$

10.3.2. First-Order Semantic Tableaux

We adapt the *uniform notation* to first-order formulas, see Figure 10.6 for the categorisation into γ- and δ-formulas.

The next definition enlarges the set of expansion rules; as usally $\mathcal{L}$ denotes the base language, that is, the language of the original formula that we consider.

Definition 10.10 (First-Order Expansion Rules)**.**

$$\frac{\gamma}{\gamma(t)} \quad t \text{ term in } \mathcal{L}^+ \qquad\qquad \frac{\delta}{\delta(k)} \quad k \text{ fresh constant in } \mathcal{L}^+$$

universal		existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$\forall x A(x)$	$A(t)$	$\exists x A(x)$	$A(t)$
$\neg\exists x A(x)$	$\neg A(t)$	$\neg\forall x A(x)$	$\neg A(t)$

Figure 10.6.: Uniform Notation for First-Order

Here $\mathcal{L}^+$ denotes extension of $\mathcal{L}$ by new constants as introduced in applications of δ-rules. *Fresh* means that the constant k is new to the branch of the tableau.

With respect to propositional tableaux we defined the notion of *strict* tableau, cf. Definition 10.9. We can only keep this notion for the propositional part and the δ-rules of first-order tableau, but must not extend it to γ-rules, if we want to preserve completeness.

Definition 10.11. We call a tableau branch *satisfiable*, if the set $\mathcal{G}$ of sentences on it is satisfiable, that is, there exists a model of $\mathcal{G}$. A tableau is *satisfiable* if some branch is satisfiable.

Theorem 10.9. *If F has a tableau proof, then F is valid.*

Sketch of Proof. We only sketch the proof: if any tableau expansion rule is applied to a satisfiable tableau, the result is satisfiable. See [23] for a complete proof. □

Theorem 10.10. *If a sentence F is valid, then F has a tableau proof*

Proof. We call a set $\mathcal{G}$ *tableau consistent* if there is no closed tableau for $\mathcal{G}$. The collection of all tableau consistent sets fulfills the satisfaction properties. Hence by an application of the Model Existence Theorem 4.3 we conclude that if F does not have a tableau proof, then $\neg F$ is satisfiable. Contradiction. □

The expansion rules presented in Definition 10.10 are not suitable for automation. The reason is that we have not essentially advanced from Gilmore's prover, if we have to enumerate all possible ground terms (over $\mathcal{L}^+$) as it is required in the application of γ-rules as defined above. To overcome this, we introduce *free-variable* semantic tableaux. However, note that we still restrict to tableau proofs of *sentences* although the below defined expansion rules introduce free variables.

Definition 10.12 (First-Order Expansion Rules)**.**

$$\frac{\gamma}{\gamma(x)} \quad x \text{ a free variable} \qquad\qquad \frac{\delta}{\delta(f(x_1,\dots,x_n))} \quad f \text{ a Skolem function}$$

Here the arguments $x_1,\dots,x_n$ denote all free variables of the formula δ and the Skolem function f must be new on the branch.

The above notion of free-variable tableau, and in particular the formulation of the δ-rule leave a lot of room for improvement. The requirement that f must be new on the branch forces the introduction of inefficiently many new Skolem functions, which can be prevented with cleverer notions of the δ-rule. See [4] and follow-up work for this development.

Definition 10.13 (Atomic Closure Rule)**.** We define the following *atomic closure rule*: if T is a tableau and such that some branch in T contains two literals A and $\neg B$, where σ is a mgu of A and B. Then $T\sigma$ is also a tableau.

Note that the proposed tableau substitution rule is a restriction of the following general substitution rule that could alternatively be chosen: if T is a tableau for $\mathcal{G}$ and σ is free for any sentence in $\mathcal{G}$, then $T\sigma$ is also a tableau.

In the sequel of this section we prove soundness and completeness of free-variable semantic tableaux. Like for propositional tableaux, we do not follow the proof pattern of using the model existence theorem, but give a strong completeness proof that also takes care of an arbitrary strategy employed in proof search.

First, we consider soundness. For this we adapt the notion of satisfiable tableau to free-variable tableau.

Definition 10.14. A branch in a free-variable tableau is called *satisfiable*, if there exists a structure $\mathcal{A}$ and for any environment ℓ, we have $(\mathcal{A}, \ell) \models \mathcal{G}$. Analogous to above, we have that a free-variable tableau is *satisfiable*, if there exists a satisfiable branch.

Lemma 10.7. *Let T be a satisfiable (free-variable) tableau. If an expansion rule according to Definition 10.7 or Definition 10.12 is applied to T, then the result is satisfiable.*

Proof. In proof, we only consider the δ-rule, the other cases are left to the reader as exercise.

Suppose B is a branch in T such that δ occurs on B. We extend B with $\delta(f(x_1, \ldots, x_n))$ and call the result B'. Let T' denote the tableau T where B is replaced by B'. It suffices to prove that if B is satisfiable, then B' is satisfiable. Let $\mathcal{G}$ collect all formulas on B and assume $(\mathcal{A}, \ell) \models \mathcal{G}$ for a structure $\mathcal{A}$ and any interpretation ℓ. Furthermore let x denote the existentially bound variable x replaced by the term $f(x_1, \ldots, x_n)$ in the formula $\delta(f(x_1, \ldots, x_n))$.

We momentarily fix the environment ℓ. By the definition of the satisfaction relation $\models$ we find a witness $a \in \mathcal{A}$ for x such that $(\mathcal{A}, \ell\{x \mapsto a\}) \models \delta(x)$. Thus we can construct a new structure $\mathcal{A}'$ (interpreting the language extended by the Skolem function f) such that:

$$f^{\mathcal{A}'}(\ell(x_1), \ldots, \ell(x_n)) := a \ .$$

As ℓ is abitrary and $(\mathcal{A}, \ell) \models \delta$ for any environment ℓ, we can extend this interpretation of f to a total definition. Finally, it is easy to check that $(\mathcal{A}, \ell) \models \delta$ implies that $(\mathcal{A}', \ell) \models \delta(f(x_1, \ldots, x_n))$, from which the lemma follows. $\square$

Lemma 10.8. *If the atomic closure rule is applicable to a tableau T and T is satisfiable, then the result is also satisfiable.*

Proof. In proof, we show a more general statement. If the substitution rule is applied to a satisfiable tableau T, then its result is satisfiable. From this claim the lemma follows, as the atomic closure rule is a restricted form of the substitution rule.

Now with respect to the claim, we first observe observe that for any environment ℓ there exists an environment ℓ' such that for any term t: $t^{(\mathcal{A},\ell')} = t\sigma^{(\mathcal{A},\ell)}$. This observation follows easily by induction on t. Based on this observation the claim follows from the definition of satisfiability. □

Theorem 10.11. *If the sentence F has a free-variable tableau proof, then F is valid.*

Proof. Consequence of Lemmata 10.7 and 10.9. □

On our way to show completeness of free-variable semantic tableaux, we first observe that we may consider a sequence of atomic closure rules that leads to an (atomically closed) tableau as one block. This motivates the following definition.

Definition 10.15. Let T be a tableau with branches $B_1, \dots, B_n$ and for each i A_i and $\neg B_i$ are literals on B_i. If σ is a mgu of $A_1 = B_1, \dots, A_n = B_n$, then σ is called *most general atomic closure substitution.*

Recall that finding a mgu for a list of equations essentially boils down to finding an mgu for (instances of) each equation. Thus a sequence of atomic closure rules that leads to an (atomically closed) yields a most general atomic closure substitution, while vice versa the existence of such a substitution implies that the tableau T can be closed by n applications of the atomic closure rule.

Lemma 10.9 (Lifting Lemma)**.** *Suppose T is a tableau and τ a substitution free for T such that each branch in $T\tau$ is atomically closed. Then there exists a most general atomic closure substitution σ and $T\sigma$ is closed by n applications of the atomic closure rule.*

Proof. Suppose $B_1, \dots, B_n$ are the branches of T and consider the set of equations $A_1 = B_1, \dots, A_n = B_n$ representing the closure literals as above. Then τ is a unifier for this equations. Hence there exists a mgu σ such that $A_i\sigma = B_i\sigma$ for each i. Put differently σ is a most general atomic closure substitution. This concludes the argument for the first part of the lemma; the second part follows from the observations above. □

In construction a tableau proof we cannot use any strategy, but need to restrict to *fair* strategies. To be precise a *strategy S for constructing a tableau* (*strategy S* for short) has to detail, perhaps using extra information carried along, which expansion rule is supposed to be applied to a given tableau or that no further expansion is possible. For the first alternative the strategy may also update the extra information. We say a sequence of tableaux is sequence of tableau *following S*, if the tableau expansion rules are only applied in conformance with the strategy S.

Definition 10.16. A strategy S is *fair* if for any sequence of tableaux $T_1, T_2, \ldots$ following S we have for each $i \in \mathbb{N}$:

(i) Every non-literal formula in T_i is eventually expanded on each branch it occurs, and

(ii) every γ-formula occurrence in T_i has the γ-rule applied to it arbitrarily often on each branch it occurs.

Theorem 10.12 (Strong Completeness)**.** *Let S be a fair strategy and let F be a valid sentence. Then F has a tableau proof with the following properties:*

*(*i*) All tableau expansion rules are considered first and follow the strategy S, and*

*(*ii*) a block of atomic closure rules closes the tableau.*

Proof. In proof, we show the equivalent statement that the claimed tableau proof will end in one single tableau substitution rule employing a most general atomic closure substitution σ, cf. Lemma 10.9.

Let $T_1, T_2, \ldots$ denote a sequence of tableaux for $\neg F$ following S, where no T_i admits a most general atomic closure substitution. We show that $\neg F$ is satisfiable. Wlog. we assume that the sequence is infinite and picture its limit as an infinite tree T. Futhermore we can assume an enumeration $x_1, x_2, \ldots$ of the free variables introduced by γ-rules in T and an enumeration of closed terms $t_1, t_2, \ldots$ over $\mathcal{L}^+$, the extension of $\mathcal{L}$ by all Skolem function introduced by δ-rules in T.

Define a substitution τ as follows: $\tau(x_i) = t_i$. We claim $T\tau$ is not atomically closed. Suppose $T\tau$ would be atomically closed, then there exists a finite subtree $T' \subseteq T$ such that $T'\tau$ is atomically closed and $T' \subset T_i$ for some i. Hence $T_i\tau$ is atomically closed and thus by Lemma 10.9 there exists a most general atomic closure substitution σ. Contradiction to the assumption that no T_i admits a most general atomic closure substitution.

Thus $T\tau$ is not atomically closed and there exists a branch B in $T\tau$ such that for no formula F and $\neg F$ occurs on B. Let $\mathcal{G}$ collect the sentences on B. From the definition of fairness we conclude that $\mathcal{G}$ admits the closure properties. Hence, Lemma 4.4 becomes applicable to conclude that $\mathcal{G}$ is satisfiable. As $\neg F \in \mathcal{G}$ we obtain that $\neg F$ is satisfiable, contrary to our assumption. □

10.4. Skolemisation

In this section we will first be concerned with *inner* and *outer* Skolemisation, the presentation is partly based on [46]. Before we can make these notions more precise some additional definitions are necessary. In particular we review some results on lower and upper bounds of the *Herbrand complexity* of a clause set $\mathcal{C}$.

We recall the following definition of the ground instances of a set of universal sentences $\mathcal{G}$ from Chapter 6:

$$\mathsf{Gr}(\mathcal{G}) := \{F(t_1, \ldots, t_n) \mid \forall x_1 \cdots \forall x_n F(x_1, \ldots, x_n) \in \mathcal{G}, \text{where the } t_i \text{ are closed}\} .$$

Definition 10.17. Let $\mathcal{C}$ be an unsatisfiable set of clauses and let $\mathsf{Gr}(\mathcal{C})$ denote the ground instances of $\mathcal{C}$. Then the *Herbrand complexity of* $\mathcal{C}$ is defined as follows:

$$\mathsf{HC}(\mathcal{C}) = \min\{|\mathcal{C}'| \colon \mathcal{C}' \text{ is unsatisfiable and } \mathcal{C}' \subseteq \mathsf{Gr}(\mathcal{C})\} \ .$$

Theorem 10.13. *Let Γ be a resolution refutation of a clause set $\mathcal{C}$ and let n denote the* length $|\Gamma|$ *of this refutation (counting the number of clauses in the refutation). Then* $\mathsf{HC}(\mathcal{C}) \leqslant 2^{2n}$.

Proof. In proof it suffices to construct a suitable ground refutation Γ' of the refutation Γ, as $\mathsf{HC}(\mathcal{C}) \leqslant |\Gamma'|$. We show the following slight generalisation: let Γ be a derivation of C_n from $\mathcal{C}$ with $|\Gamma| \leqslant n$, then there exists a ground derivation Γ' of a ground instance C'_n (of clause C_n) from a subset $\mathcal{C}' \subseteq \mathsf{Gr}(\mathcal{C})$. The length of Γ' is $\leqslant 2^{2n}$.

To show the claim, we argue inductively and only consider the step case. We fix a derivation Γ of length $n+1$. Wlog. let $C_{n+1} = E\sigma \vee F\sigma$, which is the result of a resolution step between $E \vee A$ and $F \vee \neg B$. Here σ is the mgu of A and B. There exists a ground substitution τ such that $A\tau = B\tau$ and by induction hypothesis there exist derivations Γ'_1, Γ'_2 of $E\tau \vee A\tau$ and $F\tau \vee \neg B\tau$, respectively, where $|\Gamma'_1| \leqslant 2^{2n}$ and $|\Gamma'_2| \leqslant 2^{2n}$. Hence there exists a ground derivation Γ' of $C'_{n+1} = E\tau \vee F\tau$ such that

$$|\Gamma'| \leqslant 2 \cdot 2^{2n} + 1 \leqslant 2^{2(n+1)} \ ,$$

Hence the theorem follows. $\square$

In order to precisely formulate the next result, we need the following definition:

$$2_0 := 1 \qquad 2_{n+1} = 2^{2_n} \ .$$

We remark that 2_n is a non-elementary function.

Theorem 10.14. *There exists a (finite) set of clauses $\mathcal{C}_n$ such that* $\mathsf{HC}(\mathcal{C}_n) \geqslant \frac{1}{2} \cdot 2_n$.

The proof of the theorem is essentially based on the following example by Statman.

Example 10.1. Consider the following clause set $\mathcal{C}_n$ parametrised in n:

$$\begin{aligned}
\mathcal{C}_n &:= \mathsf{ST} \cup \mathsf{ID} \cup \{\mathsf{p} \cdot \mathsf{q} \neq \mathsf{p} \cdot ((\mathsf{T}_n \cdot \mathsf{q}) \cdot \mathsf{q})\} \\
\mathsf{ST} &:= \{\mathsf{S}xyz = (xz)(yz), \mathsf{B}xyz = x(yz), \mathsf{C}xyz = (xz)y, \mathsf{I}x = x, \mathsf{p}x = \mathsf{p}(\mathsf{q}x)\} \\
\mathsf{T} &:= (\mathsf{SB})((\mathsf{CB})\mathsf{I}) \\
\mathsf{T}_1 &:= \mathsf{T} \\
\mathsf{T}_{k+1} &:= \mathsf{T}_k\mathsf{T} \ ,
\end{aligned}$$

where ID abbreviates a suitably chosen set of equality axioms for the language of $\mathcal{C}_n$. Note that the only function symbol is application (denoted as $\cdot$), in particular p and q are individual constants.

Lemma 10.10. *$\mathsf{T}yx = y(yx)$ is derivable*

Proof.

$$(\mathsf{SB})((\mathsf{CB})\mathsf{I})yx = (\mathsf{B}y)((\mathsf{CB})\mathsf{I}y)x =$$
$$= (\mathsf{B}y)((\mathsf{B}y)\mathsf{I})x = y((\mathsf{B}y\mathsf{I})x) = y(y(\mathsf{I}x)) = y(yx)\ .$$

□

We make use of the following abbreviations:

$$\mathsf{H}_1(y) = \forall x\ \mathsf{p}x = \mathsf{p}(yx) \qquad \mathsf{H}_{m+1}(y) = \forall x\ (\mathsf{H}_m(x) \to \mathsf{H}_m(yx))\ .$$

The proof of the next lemma is not difficult and delegated to the problem section.

Lemma 10.11. *$\mathsf{H}_1(y) \to \mathsf{H}_1(\mathsf{T}y)$ and $\forall y\ (\mathsf{H}_1(y) \to \mathsf{H}_1(\mathsf{T}y))$ are derivable.*

Lemma 10.12. *$\mathsf{H}_{m+1}(y) \to H_{m+1}(\mathsf{T}y)$ and $\forall y\ (\mathsf{H}_{m+1}(y) \to H_{m+1}(\mathsf{T}y))$ are derivable.*

Proof. Following the pattern of the proof of Lemma 10.11 one shows that $\forall x\ (A(x) \to A(yx)) \to \forall x(A(x) \to A(y(yx)))$ is derivable. Using $y(yx) = \mathsf{T}yx$ and setting $A = \mathsf{H}_m$ this implies

$$\mathsf{H}_{m+1}(y) \to H_{m+1}(\mathsf{T}y) \qquad \forall y\ (\mathsf{H}_{m+1}(y) \to H_{m+1}(\mathsf{T}y))\ .$$

Hence the lemma follows. □

We say a proofs in family of proofs $(\Pi_n)_{n\geqslant 0}$ are *short* if the lenght of the proofs Π_n are independent of n.

Corollary 10.1. *$\mathsf{H}_2(\mathsf{T}), \ldots, \mathsf{H}_{n+1}(\mathsf{T})$ are derivable by short proofs.*

Proof. To see the corollary, it suffices to formalise the informal proofs used for the correctness of Lemmas 10.11 and 10.12. □

Lemma 10.13. *Statman's example is unsatisfiable; which can be shown with a proof linear in n.*

Proof.

$$\begin{prooftree}
\AxiomC{$\mathsf{pq} \neq \mathsf{p}(\mathsf{T}_n\mathsf{q})\mathsf{q}$}
\AxiomC{$\forall x\ \mathsf{p}x = \mathsf{p}(\mathsf{q}x)$}
\AxiomC{$\mathsf{H}_n(\mathsf{T})$}
\AxiomC{$\forall x\ (\mathsf{H}_n(x) \to \mathsf{H}_n(\mathsf{T}x))$}
\UnaryInfC{$\mathsf{H}_n(\mathsf{T}) \to \mathsf{H}_n(\mathsf{T}_2)$}
\BinaryInfC{$\forall x\ (\mathsf{H}_{n-1}(x) \to \mathsf{H}_{n-1}(\mathsf{T}_2 x))$}
\doubleLine
\UnaryInfC{$\mathsf{H}_2(\mathsf{T}_n)$}
\UnaryInfC{$\forall x\ \mathsf{p}x = \mathsf{p}(\mathsf{q}x) \to \forall x\ \mathsf{p}x = \mathsf{p}(\mathsf{T}_n\mathsf{q})x$}
\BinaryInfC{$\forall x\ \mathsf{p}x = \mathsf{p}(\mathsf{T}_n\mathsf{q})x$}
\UnaryInfC{$\mathsf{pq} = \mathsf{p}(\mathsf{T}_n\mathsf{q})\mathsf{q}$}
\BinaryInfC{$\square$}
\end{prooftree}$$

□

As a corollary to Theorem 10.14 and Lemma 10.13 we conclude the following result.

Corollary 10.2. *There exists clause sets whose refutation in resolution is non-elementarily longer than its refutation in natural deduction*

Proof. Consider Statman's example $\mathcal{C}_n$ in conjunction with Theorem 10.13 The shortest resolution refutation of $\mathcal{C}_n$ is $\geqslant 2_{n-1}$, but the length of the (informal) refutation given in Lemma 10.13 is $\mathrm{O}(n)$. The informal refutation can be formalised in natural deduction. □

It can be shown that a similar speed-up in proof length can be achieved by *structural (outer) Skolemisation* in comparion to (standard) *prenex Skolemisation.* A formula is called *rectified* if different quantifiers bind different variables.

Definition 10.18. Let A be a rectified formula and $\mathsf{Q}x\ G$ a subformula of A. For any subformula $\mathsf{Q}'y\ H$ of G we say $\mathsf{Q}'y$ is *in the scope* of $\mathsf{Q}x$. This is denoted as $\mathsf{Q}x <_A \mathsf{Q}'y$.

Based on the notion of scope, we clarify the concepts of *inner* and *outer* Skolemisation. Recall the definition of *negation normal form* (*NNF* for short). A formula is in NNF, if it does not contain implication, and every negation signs occur directly in front of an atomic formula.

Definition 10.19. Let A be a rectified sentence in NNF and let $\exists x B$ a subformula of A at position p. Furthemore let $\{y_1, \dots, y_k\} = \{y \mid \forall y <_A \exists x\}$ and let $\{z_1, \dots, z_l\} = \mathcal{FV}\mathsf{ar}(\exists x B)$. We say that $A[B\{x \mapsto f(y_1, \dots, y_k)\}]$ is obtained by an *outer Skolemisation step*, while $A[B\{x \mapsto f(z_1, \dots, z_l)\}]$ is obtained by an *inner Skolemisation step.*

We define the *structural (outer) Skolem form.*

Definition 10.20. Let A be a rectified sentence in NNF. We define the mapping rsk as follows:

$$\mathsf{rsk}(A) := \begin{cases} A & \text{no existential quantifier in } A \\ \mathsf{rsk}(A_{-\exists y})\{y \mapsto f(x_1, \dots, x_n)\} & \forall x_1, \dots, \forall x_n <_A \exists y \end{cases}$$

where

(i) $\exists y$ is the first existential quantifier in A,

(ii) $A_{-\exists y}$ denotes A after omission of $\exists y$, and

(iii) the Skolem function symbol f is fresh.

The formula $\mathsf{rsk}(A)$ is the *structural (outer) Skolem form* of A.

We remark that in the literature (in particular in [5]) the above definition is generalised to arbitrary formulas. For that one distinguishes between *strong* and *weak* quantifiers. Let A be a formula. If $\forall x$ occurs positively (negatively) in A then the quantifier $\forall x$ is called *strong* (*weak*); dual for $\exists x$. Then one adapts

Definition 10.20 to a Skolemisation of *weak* quantifers; the resulting definition is called *refutational Skolem form* in [5]. To clarify this connection we employ the notation from [5].

Definition 10.21. Let A be a sentence in NNF and A' a prenex normal form of A. Then $\mathsf{rsk}(A')$ is the *prenex Skolem form* of A. On the other hand the *antiprenex form* of A is obtained my minimising the quantifier range by quantifier shifting rules. Then if A' is the antiprenex form of A, then $\mathsf{rsk}(A')$ is the *antiprenex Skolem form*

Theorem 10.15. *Let A be a sentence in NNF, then $A \sim \mathsf{rsk}(A)$.*

Proof. The proof follows the pattern of the proof for equivalence of satisfiabilty for "standard" Skolemisation, see Chapter 6. □

The following theorem is stated without proof, the interested reader is referred to [5].

Theorem 10.16. *There exists a set of sentences $\mathcal{D}_n$ with $\mathsf{HC}(\mathcal{D}'_n) = 2^{2^{2^{\mathrm{O}(n)}}}$ for the structural Skolem form $\mathcal{D}'_n$ of $\mathcal{D}_n$. On the other hand $\mathsf{HC}(\mathcal{D}''_n) \geqslant \frac{1}{2}2_n$ for the prenex Skolem form $\mathcal{D}''_n$ of $\mathcal{D}_n$.*

Before we turn to *inner* Skolemisation we present *Andrew's Skolem form* a mixture of inner and outer Skolemisation.

Definition 10.22. Let A be a rectified in NNF; Andrew's Skolem form is defined as follows:

$$\mathsf{rsk}_A(A) := \begin{cases} A & \text{no existential quantifiers} \\ \mathsf{rsk}_A(A_{-\exists y})\{y \mapsto f(\vec{x})\} & \forall_1 x_1, \dots, \forall_n x_n <_A \exists y \end{cases},$$

where

(i) $\exists y\ B$ is a subformula of A and $\exists y$ is the first strong quantifer in A and

(ii) all $x_1, \dots, x_n$ occur free in $\exists y\ B$.

The proof of the following theorem can be found in [1, 2].

Theorem 10.17. *Let A be a sentence in NNF, then $A \sim \mathsf{rsk}_A(A)$.*

In the sequel of this section we study inner Skolemisation techniques.

Definition 10.23 (Optimised Skolemisation)**.** Let A be a sentence in NNF and $B = \exists x_1 \cdots x_k(E \land F)$ a subformula of A with $\mathcal{FV}\mathsf{ar}(\exists\vec{x}(E \land F)) = \{y_1, \dots, y_n\}$. Suppose $A = C[B]$ and suppose $A \to \forall y_1, \dots, y_n \exists x_1 \cdots x_k E$ is valid. Then we define an *optimised Skolemisation step* as follows:

$$\mathsf{opt\_step}(A) := \forall \vec{y} E\{\dots, x_i \mapsto f_i(\vec{y}), \dots\} \land C[F\{\dots, x_i \mapsto f_i(\vec{y}), \dots\}] ,$$

where $f_1, \dots, f_k$ are new Skolem function symbols.

Theorem 10.18. *Optimised Skolemisation preserves satisfiability.*

Proof. We restrict our attention to the interesting case, were we show satisfiability of $\mathsf{opt\_step}(A)$ from the satisfiabilty of A. A full proof can be found in [42].

Suppose A is satisfiable with some interpretation $\mathcal{I}$ and consider an arbitrary sequence $a_1, \dots, a_n$ of domain elements in $\mathcal{I}$. We define an interpretation $\mathcal{I}'$ of the Skolem functions $f_1, \dots, f_k$ occurring in $\mathsf{opt\_step}(A)$ by case distinction.

Case $\mathcal{I}\{\vec{y} \mapsto \vec{a}\} \models \exists \vec{x}(E \wedge F)$: we set $f_i^{\mathcal{I}'}(a_1, \dots, a_n)$ according to the witness for the existentially quantified variables x_i in $E \wedge F$. *Case* $\mathcal{I}\{\vec{y} \mapsto \vec{a}\} \not\models \exists \vec{x}(E \wedge F)$: we set $f_i^{\mathcal{I}'}(a_1, \dots, a_n) := b_i$, where $\mathcal{I}\{\vec{y} \mapsto \vec{a}, \vec{x} \mapsto b_i\} \models E$. (Here we exploit the extra condition $A \to \forall y_1, \dots, y_n \exists x_1 \cdots x_k E$.)

The definition of the interpretation of the f_i is clearly well-defined. Furthermore we have:

$$\mathcal{I}' \models A \wedge (\exists \vec{x}(E \wedge F) \to F\{\dots, x_i \mapsto f_i(\vec{y}), \dots\}) \ ,$$

from which we conclude (i) $\mathcal{I}' \models C[F\{\dots, x_i \mapsto f_i(\vec{y}), \dots\}]$ as $\mathcal{I}' \models A$. Furthermore, we obtain:

$$\mathcal{I}' \models \forall y_1, \dots, y_n \exists x_1 \cdots x_k E \wedge (\exists x_1 \cdots x_k E \to E\{\dots, x_i \mapsto f_i(\vec{y}), \dots\} \ .$$

This follows from the assumption $A \to \forall y_1, \dots, y_n \exists x_1 \cdots x_k E$ and the definition of $f_i^{\mathcal{I}'}$. Thus we obtain (ii) $\mathcal{I}' \models \forall \vec{y} E\{\dots, x_i \mapsto f_i(\vec{y}), \dots\}$.

Due to (i) and (ii) we have $\mathcal{I}' \models \mathsf{opt\_step}(A)$ and the theorem follows. □

We remark that in comparison to (standard) inner Skolemisation is that some literals from clauses are deleted. We say a clause C *subsumes* clause D, if $\exists\ \sigma$ such that the multiset of literals of $C\sigma$ is contained in the multiset of literals of D (denoted $C\sigma \subseteq D$). A clause C is a *condensation* of D if C is a proper (multiple) factor of D that subsumes D.

Definition 10.24. Let $B = \exists \vec{x}(E_1 \wedge \cdots \wedge E_\ell)$ be a formula and let $\{\vec{z}_1\} := \mathcal{FV}\mathsf{ar}(E_1) \backslash \{\vec{x}\}$ and for all $i = 2, \dots, \ell$: $\{\vec{z}_i\} = \mathcal{FV}\mathsf{ar}(E_i) \backslash \left(\bigcup_{j<i} \mathcal{FV}\mathsf{ar}(E_j) \cup \{\vec{x}\}\right)$. Then we call $\langle \{\vec{z}_1\}, \dots, \{\vec{z}_\ell\} \rangle$ the *(free variable) splitting* of B.

Note that in a splitting $\{\vec{z}_i\}$ contains the free variables of E_i which have not yet occurred minus $\vec{x}$. Suppose that each conjunction E_i contains at least one of the variables from $\overline{x}$. Then we have the following easy observation: Let $\langle \{\vec{z}_1\}, \dots, \{\vec{z}_\ell\} \rangle$ be a splitting of $\exists \vec{x}(E_1 \wedge \cdots \wedge E_\ell)$. Then $\langle \{\vec{z}_1, \vec{z}_2\}, \dots, \{\vec{z}_\ell\} \rangle$ is a splitting of $\exists \vec{v}(E_2 \wedge \cdots \wedge E_\ell)\{x_i \mapsto f_i(\vec{z}_1, \vec{v})\}$ where $\vec{v}$ are new.

Definition 10.25. Let A be a sentence in NNF and $B = \exists x_1, \dots, x_k(E_1 \wedge \cdots \wedge E_\ell)$ a subformula such that $A = C[B]$. Further let $\langle \{\vec{z}_1\}, \dots, \{\vec{z}_\ell\} \rangle$ be a free variable splitting of B. Then a *strong Skolemisation step* is defined as $\mathsf{str\_step}(A) = C[D]$ where

$$D := \forall \vec{w}_2, \dots, \vec{w}_\ell E_1\{x_i \mapsto f_i(\vec{z}_1, \vec{w}_2, \dots, \vec{w}_\ell)\} \wedge \cdots$$
$$\cdots \wedge E_\ell\{x_i \mapsto f_i(\vec{z}_1, \vec{z}_2, \dots, \vec{z}_\ell)\} \ .$$

Here every variable sequence $\vec{w}_i$ has equal length as the variable sequence $\vec{z}_i$ and for all $j = 1, \dots, \ell$: f_j is a new Skolem function symbol.

Lemma 10.14. *If $\exists x_1, \dots, x_k(E \land F)$ is satisfiable, then the following formula is satisfiable as well:*

$$\forall w_1, \dots, w_k E\{x_i \mapsto f_i(\vec{y}, \vec{w})\} \land \exists v_1, \dots, v_k F\{x_i \mapsto f_i(\vec{y}, \vec{v})\}$$

where $\{y_1, \dots, y_n\} = \mathcal{FV}\mathsf{ar}(E) \setminus \{x_1, \dots, x_k\}$.

Proof. We restrict our attention to the case $k = 1$ as the general case follows similarly. Suppose $\exists x(E \land F)$ is satisfiable with interpretation $\mathcal{I}$. Then for any sequence $a_1, \dots, a_n$ of domain elements, there exists $b \in \mathcal{I}$ such that:

$$\mathcal{I}\{\vec{y} \mapsto a_i, x \mapsto b\} \models E \land F \; . \tag{10.1}$$

Let $S(\vec{a}) = \{b_1, \dots, b_m\}$ denote the set of b's such that b fullfils (10.1). Then we have (i) for all $b \in S(\vec{a})$: $\mathcal{I}\{\vec{y} \mapsto a_i, x \mapsto b\} \models E$ and (ii) for some $b \in S(\vec{a})$: $\mathcal{I}\{\vec{y} \mapsto a_i, x \mapsto b\} \models F$.

Let D denote the domain of $\mathcal{I}$. We define a mapping $h \colon D^n \times D \to D$ such that $h(\vec{a}, b) = c$ iff $b \in S(\vec{a})$. Thus by construction, we have (i) for all $c \in D$: $\mathcal{I}\{\vec{y} \mapsto a_i, x \mapsto h(\vec{a}, c)\} \models E$ and (ii) for some $c \in D$: $\mathcal{I}\{\vec{y} \mapsto a_i, x \mapsto h(\vec{a}, c)\} \models F$. Finally we extend $\mathcal{I}$ to an interpretation $\mathcal{I}'$ of the new function symbol f such that $f^{I'}(\vec{a}, c) := h(\vec{a}, c)$. In sum, we obtain:

$$\begin{aligned} &\mathcal{I}'\{\vec{y} \mapsto a_i, w \mapsto c\} \models E\{x \mapsto f(\vec{y}, w)\} && \text{for all } x \in D \\ &\mathcal{I}'\{\vec{y} \mapsto a_i, w \mapsto c\} \models F\{x \mapsto f(\vec{y}, w)\} && \text{for some } x \in D \; . \end{aligned}$$

Thus the lemma follows. ☐

Theorem 10.19. *Strong Skolemisation preserves satisfiability.*

Proof. For the interesting direction that $\mathsf{str\_step}(A)$ is satisfiable, whenever A is satisfiable, one proceeds by induction on the context C. We only treat the case where the context is empty as the inductive step essentially follows from the induction hypothesis. The full proof can be found in [42].

Let $A = B = \exists \vec{x}(E_1 \land \cdots \land E_\ell)$. We proceed by side-induction on ℓ. The base case follows from the satisfaction preservence of inner Skolemisation (see Problem 10.12). Hence we continue with the induction step.

By assumption A is satifiable in an interpretation $\mathcal{I}$. By Lemma 10.14 there exists an extension $\mathcal{I}'$ of $\mathcal{I}'$ such that

$$\mathcal{I}' \models \forall \vec{w} E_1\{x_i \mapsto h_i(\vec{z}_1, \vec{w})\} \land \exists \vec{v}(E_2 \land \cdots \land E_\ell)\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\} \; ,$$

where $\langle\{\vec{z}_1\}, \dots, \{\vec{z}_\ell\}\rangle$ denotes the splitting of B. Induction hypothesis is applicable on

$$\exists \vec{v}(E_2 \land \cdots \land E_\ell)\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\} \; ,$$

as $\langle\{\vec{z}_1, \vec{z}_2\}, \dots, \{\vec{z}_\ell\}\rangle$ is its splitting (recall the above observation). Hence there exists an extension $\mathcal{I}''$ of $\mathcal{I}'$ that models the following formula:

$$\begin{aligned}\forall \vec{w}_3, \ldots, \vec{w}_\ell E_2\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\}\{v_j \mapsto g_j(\vec{z}_1, \vec{z}_2, \vec{w}_3, \ldots, \vec{w}_\ell)\} \wedge \\ \wedge \forall \vec{w}_k E_{\ell-1}\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\}\{v_j \mapsto g_j(\vec{z}_1, \vec{z}_2, \ldots, \vec{w}_\ell)\} \wedge \\ \wedge E_\ell\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\}\{v_j \mapsto g_j(\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_\ell)\} .\end{aligned}$$

Moreover we have that $\mathcal{I}'' \models \forall \vec{w} E_1\{x_i \mapsto h_i(\vec{z}_1, \vec{w}_1)\}$ and thus in particular $\mathcal{I}''$ models:

$$\forall \vec{w}_2, \ldots, \vec{w}_\ell E_1\{x_i \mapsto h_i(\vec{z}_1, \vec{v})\}\{v_j \mapsto g_j(\vec{z}_1, \vec{w}_2, \vec{w}_3, \ldots, \vec{w}_\ell)\} .$$

Finally we set

$$f_i(\vec{u}_1, \ldots, \vec{u}_\ell) := h_i(\vec{u}_1, g_1(\vec{u}_1, \ldots, \vec{u}_\ell), \ldots, g_m(\vec{u}_1, \ldots, \vec{u}_\ell)) ,$$

where $m = |\vec{z}_1|$. Then $\mathcal{I}''' \models \mathsf{str\_step}(A)$. □

10.5. Redundancy Criteria and Deletion

We recall the definition of the resolution operator Res and its n-fold iteration Res^n from Definition 10.6.

Definition 10.26. Let $\mathsf{d}(\mathcal{C}) := \min\{n \mid \square \in \mathsf{Res}^n(\mathcal{C})\}$. The *search complexity* of Res with respect to a clause set $\mathcal{C}$ is defined as $\mathsf{scomp}(\mathcal{C}) := |\mathsf{Res}^{\mathsf{d}(\mathcal{C})}(\mathcal{C})|$.

Refinements reduce the search space as fewer derivations are possible, however the minimal proof length may be increased. On the other hand redundancy tests cannot increase the proof length, but may be costly. The next lemma essentially follows from the definitions.

Lemma 10.15. *Application of subsumption and tautology elimination as preprocession steps preserves completeness.*

A more interesting question is whether the application of these redundancy criteria during proof search affect (refutational) completeness. In general we speak of *tautology elimination* if newly derived tautological clauses are removed. Subsumption and resolution can be combined in the following ways: In *forward subsumption* newly derived clauses subsumed by existing clauses are deleted. In *backward subsumption* existing clauses subsumed by newly derived clauses become inactive. Finally *replacement* means that the the set of all clauses (derived and intital) are frequently reduced under subsumption.

The proof of the next lemma can be found in [36].

Lemma 10.16. *Let C and D be clauses and C a tautology. Any resolvent of C and D is either a tautology or subsumed by D.*

Theorem 10.20. *Resolution remains complete under forward subsumption and elimination of tautologies.*

Proof. Straighforward application of Lemma 10.16. □

Problems

Problem 10.1. Consider the class of first-order formulas of the following form:

$$\forall x_1 \cdots \forall x_n \exists y_1 \cdots \exists y_n M \ ,$$

where M is a matrix, that is, quantifier-free. Show that the DPLL-method constitutes a decision procedure for this class.

Problem 10.2. Consider the following family of clause sets $\mathcal{C}_n$:

$$\mathcal{C}_n = \{\mathsf{P}(\mathsf{a})\} \cup \bigcup_{i=0}^{n-1} \{\neg \mathsf{P}(\mathsf{f}^{(i)}(\mathsf{a})) \vee \mathsf{P}(\mathsf{f}^{(i+1)}(\mathsf{a}))\} \cup \{\neg \mathsf{P}(\mathsf{f}^{(n)}(\mathsf{b}))\} \ ,$$

where $\mathsf{f}^0(t) := t$, $\mathsf{f}^{i+1}(t) := \mathsf{f}(\mathsf{f}^i(t))$ for any term t. Show (using the DPLL-method) that all $\mathcal{C}_n$ are satisfiable.

Problem 10.3. Show that the satisfiability of a set of ground Horn clauses, that is at most one positive literal, can be decided by the DPLL rules without splitting rule.

Problem 10.4. Employ Problem 10.3 to show that satisfiability of ground Horn clauses is polytime computable.

Problem 10.5. Show Lemma 10.5.

Problem 10.6. Let S be a collection of sets of propositional formulas. Following Fitting, we call S a *propositional consistency property* if the following conditions are met for every $\mathcal{G} \in S$:

(i) For no propositional atom A, both A and $\neg A$ are in $\mathcal{G}$.

(ii) If $\neg\neg F \in \mathcal{G}$, then $\mathcal{G} \cup \{F\} \in S$.

(iii) If $\alpha \in \mathcal{G}$, then $\mathcal{G} \cup \{\alpha_1, \alpha_2\} \in S$.

(iv) If $\beta \in \mathcal{G}$, then $\mathcal{G} \cup \{\beta_1\} \in S$ or $\mathcal{G} \cup \{\beta_2\} \in S$.

Show the following *propositional* model existence property: if S is a propositional consistency property and $\mathcal{G} \in S$, then $\mathcal{G}$ is satisfiable.

Problem 10.7. Let a set $\mathcal{G}$ of propositional formulas be *tableau consistent* if there is no closed tableau for $\mathcal{G}$. Show that the collection of all tableau consistent formulas is a propositional consistency property.

Problem 10.8. Employ the model existence property from Problem 10.7 to show that propositional tableau is complete.

Problem 10.9. Employ the model existence property for first-order (Theorem 4.3) to show completeness of first-order semantic tableaux.

Problem 10.10. Complete the proof of Lemma 10.7.

Problem 10.11. Give a proof of Lemma 10.11 and show more generally that $\forall x\ (A(x) \to A(yx)) \to \forall x(A(x) \to A(y(yx)))$ is derivable.

Problem 10.12. Show that structural (outer) Skolemisation preserveres satisfiability, cf. Theorem 10.15.

Problem 10.13. Show that inner Skolemisation preserves satisfiability. Compare to the proof for structural (outer) Skolemisation.

11. Automated Reasoning with Equality

In this chapter we introduce the theory of automated reasoning with equality as well as more advanced techniques of automated reasoning for predicate logic without equality. Most importantly we introduce ordered resolution in Section 11.1. In Section 11.2 we extend resolution by suitably defined inference rules to overcome this (practical) restriction. The obtained calculus is called *paramodulation calculus*. As preparation step for the superposition calculus, we recall its origin in completion techniques introduced in rewriting. In Section 11.3 we study ordered completion and proof orders. Finally, in Section 11.4 we study a refined version of the paramodulation calculus, the *superposition calculus*,

11.1. Ordered Resolution

If the inference rules in Chapter 10 are implemented, the inefficiency quickly becomes apparent. One of the reasons for their inefficiency is the large search space. To overcome this restriction *ordered resolution* has been invented.

A *proper order* $\succ$ is an irreflexive and transitive relation. The converse of $\succ$ is written as $\prec$. A *quasi-order* is a reflexive and transitive relation and a *partial order* is an anti-symmetric quasi-order. A proper order $\succ$ on a set A is *well-founded* (on A) if there exists no infinite descending sequence $a_1 \succ a_2 \succ \cdots$ of elements of A. A well-founded proper order is called a *well-founded order*. A proper order is called *linear* (or *total*) on A if for all $a, b \in A$, a different from b, a and b are comparable by $\succ$. A linear well-founded order is called a *well-order*.

Definition 11.1. Given an arbitrary well-founded and total order $\succ$ on ground atomic formulas, we extend $\succ$ to a well-founded proper order $\succ_{\mathsf{L}}$ on ground literals such that the following conditions are fulfilled:

(i) If $A \succ B$, then $(\neg)A \succ_{\mathsf{L}} (\neg)B$

(ii) $\neg A \succ_{\mathsf{L}} A$.

Let $\succ_{\mathsf{L}}$ be a total order on ground literals according to Definition 11.1. We say a (not necessarily ground) literal L is *maximal* if there exists a ground substitution σ such that for no other literal M: $M\sigma \succ_{\mathsf{L}} L\sigma$. We say L is *strictly maximal* if there exists a (ground) substitution σ such that for no other literal M: $M\sigma \succcurlyeq_{\mathsf{L}} L\sigma$. Here $\succcurlyeq_{\mathsf{L}}$ denotes the reflexive closure of $\succ_{\mathsf{L}}$.

In Figure 11.1 we give the inference rules for *ordered resolution*. This variant of the resolution calculus remains sound and complete, but allows to narrow the search space considerably.

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad\qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

Here σ is a mgu of A and B. The first inference is called *ordered resolution*, while the second one is called *ordered factoring*.

- For ordered resolution $A\sigma$ is strictly maximal with respect to $C\sigma$ and $\neg B\sigma$ is maximal with respect to $D\sigma$.
- For ordered factoring $A\sigma$ is strictly maximal with respect to $C\sigma$.

Figure 11.1.: Ordered Resolution Calculus

Definition 11.2. Let $\mathcal{C}$ be a set of clauses. We define the *ordered resolution operator* $\mathsf{Res}_{\mathsf{OR}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{OR}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of inferencex in Fig. 11.1 with premises in } \mathcal{C}\}\ .$$

The n^{th} (unrestricted) iteration $\mathsf{Res}^n_{\mathsf{OR}}$ ($\mathsf{Res}^*_{\mathsf{OR}}$) of the operator $\mathsf{Res}_{\mathsf{OR}}$ is defined as above.

Theorem 11.1. *Ordered resolution is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\Box \in \mathsf{Res}^*_{\mathsf{OR}}(\mathcal{C})$.*

Sketch of Proof. Soundness of ordered resolution is a consequence of Theorem 10.6 as ordered resolution *restricts* resolution.

In order to adapt the completeness proof we first have to extend the underlying order $\succ_{\mathsf{L}}$ on literals (see Definition 11.1) to an order on clauses. For that one usually employs the so called multiset extension of an order (see [51] for a definition). In this context we only need to know that any well-founded and total order on literals is extensible to a well-founded and total order on clauses (denoted as $\succ_{\mathsf{C}}$).

We can refine Corollary 6.2 in such a way that if F is an unsatisfiable formula corresponding to $\mathcal{C}$ there exists a maximal set of clauses $\mathcal{D}$ such that $\mathcal{D}$ is unsatisfiable and each clause in $\mathcal{D}$ is ground. Furthermore, any clause in $\mathcal{D}$ is an instance of a clause in $\mathcal{C}$. Here a set of clauses $\mathcal{D}$ is called *maximal* if there exists no set of clauses $\mathcal{D}' \cup \{D\}$, fulfilling the above requirements, such that $\mathcal{D} = \mathcal{D}' \cup \{D_1, \ldots, D_n\}$ and for all $1 \leqslant i \leqslant n$ we have $D \succ_{\mathsf{C}} D_i$, while there is no clause in $\mathcal{D}'$ that is larger than D. Then completeness of *ground* ordered resolution follows if we follow the pattern of the proof of Theorem 10.7 but replace the application of Corollary 6.2 by the refinement described above. Finally, in order to prove completeness of ordered resolution it remains to adapt the lifting lemmas, Lemmas 10.2 and 10.3, suitably, which does not provide any problems. $\Box$

11.2. Paramodulation and Ordered Paramodulation

We are ready to admit the equality sign $=$ to our base language. In principle we can eliminate equality from our language and apply the aforementioned (ordered) resolution calculi to deal with formulas containing $=$. This is a consequence of Lemmas 6.1 and and 6.2 studied in Chapter 6. However, this would be hopelessly inefficient. Instead one expands the (ordered) resolution calculus by a new inference rule, designated to deal with equality. This rules is called *paramodulation*. In order to give a precise definition, we need an additional definition.

Let s, t be terms and let A be a formula. In Chapter 3 we used the notation $A(x)$ to indicate an occurrence of the variable x in A and we wrote $A(t)$ to indicate the simultaneous replacement of x by t in A. In the following we need to make this definition more precise.

Let $\Box$ be a fresh constant and let $\mathcal{L}$ be our basic language. Then terms of $\mathcal{L} \cup \{\Box\}$ such that $\Box$ occurs exactly once, are are called *contexts*. The empty context is denoted as $\Box$. For a context $C[\Box]$ and a term t (of $\mathcal{L}$), we write $C[t]$ for the replacement of $\Box$ by t.

In Figure 11.2 we give the inference rules for the *paramodulation calculus*. This extension of the resolution calculus to languages that contain $=$ remains sound and complete. However, due to the presence of equality the search space explodes.

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'} \quad .$$

Here σ is a mgu of A and B and σ' is a mgu of s and s'.

Figure 11.2.: Paramodulation Calculus

Definition 11.3. Let $\mathcal{C}$ be a set of clauses. We define the *paramodulation operator* $\mathsf{Res}_{\mathsf{P}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{P}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of inferences in Fig. 11.2 with premises in } \mathcal{C}\} \ .$$

The nth (unrestricted) iteration $\mathsf{Res}_{\mathsf{P}}^n$ ($\mathsf{Res}_{\mathsf{P}}^*$) of the operator $\mathsf{Res}_{\mathsf{P}}$ is defined as above.

Before we can prove soundness and completeness of the paramodulation calculus, we need to update the proof of the model existence theorem. More precisely we have to adapt the proof of Lemma 4.4 to a language containing the equality symbol $=$ (compare also Section 10.2). Finally, we are in the position to state the lemma in its full generality.

Lemma 11.1. *Let $\mathcal{G}$ be a set of formulas (of $\mathcal{L}$) admitting the closure properties.*

Then there exists an interpretation $\mathcal{M}$ such that every element of the domain of $\mathcal{M}$ is the denotation of a term (of $\mathcal{L}^+$) and $\mathcal{M} \models \mathcal{G}$.

Proof. Let $\mathcal{M}$ denote the Herbrand model defined in the proof of Lemma 10.4. Now, the crucial difference is the presence of the equality sign $=$ in $\mathcal{L}$. Suppose $(s = t) \in \mathcal{G}$, where s and t are syntactically different. Then $\mathcal{M} \not\models s = t$ as in $\mathcal{M}$ the terms s and t are interpreted by different symbols.

To overcome this, we define a variant of the term model $\mathcal{M}$, denoted as $\mathcal{M}'$. For that it suffices to consider the set $\mathcal{E}$ of all equations induced by $\mathcal{G}$:

$$\mathcal{E} := \{s = t \mid \mathcal{G} \models s = t\} .$$

Note that the assumption that $\mathcal{G}$ fulfils the closure properties implies that the definition of $\mathcal{E}$ is well-defined and that $\mathcal{E}$ gives rise to an equivalence relation $\sim$.

Based on the relation $\sim$ we define the domain of $\mathcal{M}'$ as the set of equivalent classes for the set of terms of $\mathcal{L}^+$. Let $[t]_\sim$ denote the equivalence class of t with respect to the equivalence $\sim$. We define the structure underlying $\mathcal{M}'$ as follows:

(i) $c^{\mathcal{M}} := [c]_\sim$ for any individual constant c,

(ii) $f^{\mathcal{M}}([t_1]_\sim, \ldots, [t_n]_\sim) := [f(t_1, \ldots, t_n)]_\sim$ for any n-ary function constant f and any tuple of equivalence classes $[t_1]_\sim$, ..., $[t_n]_\sim$ in $\mathcal{M}'$.

Furthermore, for any predicate constant P and for any sequence of equivalence classes $[t_1]_\sim$, ..., $[t_n]_\sim$ in $\mathcal{M}'$ we set:

$$P^{\mathcal{M}}([t_1]_\sim, \ldots, [t_n]_\sim) \Longleftrightarrow P(t_1, \ldots, t_n) \in \mathcal{G} ,$$

and interpret equality $=$ as the equivalence $\sim$. (Note that this amounts to the interpretation of $=$ as syntactic equality on the domain of $\mathcal{M}'$.)

Finally, we lift this structure to an interpretation $\mathcal{M}'$ by defining the look-up table as follows:

$$\ell(x) := [x]_\sim \qquad \text{for any variable } x .$$

This completes the definition of the interpretation $\mathcal{M}'$. The fact that $\mathcal{M}'$ is a model of $\mathcal{G}$ follows by induction on formulas as before. $\square$

Recall the lifting lemmas for resolution, Lemmas 10.2 and 10.3. The following example shows that lifting of paramodulation steps is not possible without further ado.

Example 11.1. Consider the clause set $\mathcal{C} = \{\mathsf{a} = \mathsf{b}, \mathsf{f}(x) = \mathsf{c}\}$. The only possible (non-ground) paramodulation inference is $\mathsf{f}(\mathsf{b}) = \mathsf{c}$. On the other hand the following inference is a correct ground step:

$$\frac{\mathsf{a} = \mathsf{b} \quad \mathsf{f}(\mathsf{f}(\mathsf{a})) = \mathsf{c}}{\mathsf{f}(\mathsf{f}(\mathsf{b})) = \mathsf{c}} .$$

No lifting for this inference is possible. In order to overcome this problem, one has to add so called *functional reflexivity equation* $\mathsf{f}(x) = \mathsf{f}(x)$. Then lifting becomes possible (using two steps):

$$\cfrac{\cfrac{\mathsf{a} = \mathsf{b} \quad \mathsf{f}(x) = \mathsf{f}(x)}{\mathsf{f}(\mathsf{a}) = \mathsf{f}(\mathsf{b})} \qquad \mathsf{f}(x) = \mathsf{c}}{\mathsf{f}(\mathsf{f}(\mathsf{b})) = \mathsf{c}} \quad .$$

For any n-ary function symbol the equation $f(x_1, \dots, x_n) = f(x_1, \dots, x_n)$ is called a *functional reflexivity equation.* We employ the following variant of the lifting lemma, whose proof is delegated to the problem section.

Lemma 11.2. *Let τ_1 and τ_2 be a ground substitution and consider the inference:*

$$\frac{C\tau_1 \lor (s = t)\tau_1 \quad D\tau_2 \lor L\tau_2[x\tau_2]}{C\tau_1 \lor D\tau_2 \lor L\tau_2[f(t\tau_1)]} \quad ,$$

where $x\tau_2 = f(s'\tau_3)$ and $s\tau_1 = s'\tau_3$. Then the following derivation is admissible:

$$\cfrac{\cfrac{C \lor s = t \quad f(x) = f(x)}{C \lor f(s) = f(t)} \qquad D \lor L[x]}{C \lor D \lor L[f(t)]} \quad .$$

Theorem 11.2. *Paramodulation is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form (containing all functional reflexive equations). Then F is unsatisfiable iff $\Box \in \mathsf{Res}^*_{\mathsf{OR}}(\mathcal{C})$.*

Sketch of Proof. To show soundness we have to verify that every inference rule of the resolution calculus is sound. For this one shows that if the assumptions of resolution and factoring are modelled by a model $\mathcal{M}$, then the consequence (of the rule) holds in $\mathcal{M}$ as well.

In order to show completeness it remains to show that the set of consistent set of ground clauses fulfils the satisfaction properties. For that we need to take into account the properties (viii) and (ix) which have not yet been considered. Due to the presence of paramodulation among the rules of the paramodulation calculus this is an easy exercise and left to the reader.

Then ground completeness of paramodulation follows as completeness of natural deduction or resolution. In order to lift this to a proof of completeness of paramodulation we employ Lemma 11.2. $\Box$

It is not difficult to see that the paramodulation calculus is still inefficient due to the presence of the paramodulation rule

$$\frac{C \lor s = t \quad D \lor L[s']}{(C \lor D \lor L[t])\sigma} \quad ,$$

where σ is a mgu of s and s' in the calculus. A first step to restrict the search space is to combine paramodulation with ordered resolution instead of the unrestricted resolution calculus. The corresponding rules are given in Figure 11.3.

Definition 11.4. Let $\mathcal{C}$ be a set of clauses. We define the *ordered paramodu-*

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'} \quad .$$

Here σ is a mgu of A and B and σ' is a mgu of s and s'. The last rule is called *ordered paramodulation.*

– For ordered resolution $A\sigma$ is strictly maximal with respect to $C\sigma$ and $\neg B\sigma$ is maximal with respect to $D\sigma$.

– For ordered factoring $A\sigma$ is strictly maximal with respect to $C\sigma$.

– For ordered paramodulation the equation $(s = t)\sigma'$ and the literal $L[s']\sigma'$ is maximal with respect to $D\sigma'$

Figure 11.3.: Ordered Paramodulation Calculus

lation operator $\mathsf{Res}_{\mathsf{OP}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{OP}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of inferences in Fig. 11.3 with premises in } \mathcal{C}\} \; .$$

The n^{th} (unrestricted) iteration $\mathsf{Res}^n_{\mathsf{OP}}$ ($\mathsf{Res}^*_{\mathsf{OP}}$) of the operator $\mathsf{Res}_{\mathsf{OP}}$ is defined as above.

As a consequence of Theorem 2.3 and 11.2 we conclude the following theorem.

Theorem 11.3. *Ordered paramodulation is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form (containing all functional reflexive equations). Then F is unsatisfiable iff $\Box \in \mathsf{Res}^*_{\mathsf{OP}}(\mathcal{C})$.*

11.3. Ordered Completion and Proof Orders

We assume familiarity with term rewriting, see [3, 51]. We recall the fact that a convergent (confluent and terminating) *term rewrite system* (*TRS* for short) forms a decision procedure for the underlying equational theory: $s \leftrightarrow^* t$ iff $s \downarrow t$. More precisely the *word problem* becomes decidable. We define the word problem as the validity of the consequence relation $\mathcal{E} \models s = t$, where $\mathcal{E}$ is an equational system and $s = t$ an arbitrary equation.

Definition 11.5. Consider the following inference:

$$\frac{s \rightarrow t \quad w[u] \rightarrow v}{(w[t] = v)\sigma} \quad .$$

Here σ is a mgu of s and u and u *not a variable.* The equation $(w[t] = v)\sigma$ is called *critical pair* and the term $w[u]\sigma$ is the *overlapping term.*

deduction	$\mathcal{E};\mathcal{R} \vdash \mathcal{E} \cup \{s = t\};\mathcal{R}$ if $s \leftrightarrow_{\mathcal{E}\cup\mathcal{R}} w \leftrightarrow_{\mathcal{E}\cup\mathcal{R}} t$, $s \not\succeq w$, $t \not\succeq w$	
orientation	$\mathcal{E} \cup \{s = t\};\mathcal{R} \vdash \mathcal{E};\mathcal{R} \cup \{s \to t\}$	if $s \succ t$
deletion	$\mathcal{E} \cup \{s = s\};\mathcal{R} \vdash \mathcal{E};\mathcal{R}$	
simplification	$\mathcal{E} \cup \{s = t\};\mathcal{R} \vdash \mathcal{E} \cup \{u = t\};\mathcal{R}$	if $s \to_{\mathcal{R}} u$
composition	$\mathcal{E};\mathcal{R} \cup \{s \to t\} \vdash \mathcal{E};\mathcal{R} \cup \{s \to u\}$	if $t \to_{\mathcal{R}} u$
collapse	$\mathcal{E};\mathcal{R} \cup \{s[w] \to t\} \vdash \mathcal{E} \cup \{s[u] \to t\};\mathcal{R}$ if $w \to_{\mathcal{R}} u$, either $t \succ u$ or $w \neq s$	

Figure 11.4.: Ordered (or Unfailing) Completion

Theorem 11.4. *A terminating TRS $\mathcal{R}$ is confluent iff all critical pairs between rules in $\mathcal{R}$ converge.*

Reduction orders that are total on ground terms are called *complete*. Suppose $\succ$ is a reduction order and $\mathcal{E}$ a set of equations. Consider the set of *reductive instances* of equations in $\mathcal{E}$ defined as follows:

$$\mathcal{E}^{\succ} := \{s\sigma \to t\sigma \mid s = t \in \mathcal{E}, s\sigma \succ t\sigma\} .$$

Ordered rewriting is rewriting over the rewrite relation $\to_{\mathcal{E}^{\succ}}$.

Definition 11.6. Let $\succ$ be a reduction order. The equations $\mathcal{E}$ are called *ground convergent wrt* $\succ$ if $\mathcal{E}^{\succ}$ is ground convergent, that is, the induced rewrite relation on ground terms is confluent.

The next definition generalises Definition 11.5 to equational unit clauses.

Definition 11.7. Consider the following inference over equational unit clauses:

$$\frac{s = t \quad w[u] = v}{(w[t] = v)\sigma} \quad .$$

Here σ is mgu of s and u; $t\sigma \not\succeq s\sigma$, $v\sigma \not\succeq w[u]\sigma$ and u is not a variable. The equation $(w[t] = v)\sigma$ is called an *ordered critical pair.*

The proof of the following theorem can be found in [34].

Theorem 11.5. *Let $\succ$ be a complete reduction order. A set of equations E is ground convergent wrt $\succ$ iff for all ordered critical pairs $(w[t] = v)\sigma$ (with overlapping term $w[u]\sigma$) and for all ground substitutions τ, we have: if $w[u]\sigma\tau \succ w[t]\sigma\tau$ and $w[u]\sigma\tau \succ v\sigma\tau$ then $w[t]\sigma\tau \downarrow v\sigma\tau$.*

In Figure 11.4 we give the rules for ordered (or unfailing) completion, cf. [7]. A sequence $(\mathcal{E}_0;\mathcal{R}_0) \vdash (\mathcal{E}_1;\mathcal{R}_1) \vdash \cdots$ is called a *derivation*, where usually $\mathcal{E}_0$ is the set of initial equations and $\mathcal{R}_0 = \varnothing$. The *limit* of a derivation is defined as $(\mathcal{E}_\infty; \mathcal{R}_\infty)$, where $\mathcal{E}_\infty := \bigcup_{i\geqslant 0} \bigcap_{j\geqslant i} \mathcal{E}_j$ and $\mathcal{R}_\infty := \bigcup_{i\geqslant 0} \bigcap_{j\geqslant i} \mathcal{R}_j$.

Definition 11.8. A *proof* of $s = t$ wrt $\mathcal{E}; \mathcal{R}$, denoted as $s \leftrightarrow_{\mathcal{E}\cup\mathcal{R}} t$, is defined as follows:

$$s = s_0 \; \rho_0 \; s_1 \; \rho_1 \; s_2 \cdots s_{n-1} \; \rho_{n-1} \; s_n = t \qquad n \geqslant 0 \ ,$$

where

(i) $(s_i \; \rho_i \; s_{i+1}) = (w[u\sigma] \leftrightarrow w[v\sigma])$ with $u = v \in \mathcal{E}$,

(ii) $(s_i \; \rho_i \; s_{i+1}) = (w[u\sigma] \to w[v\sigma])$ with $u \to v \in \mathcal{E}^\succ \cup \mathcal{R}$, or

(iii) $(s_i \; \rho_i \; s_{i+1}) = (w[u\sigma] \leftarrow w[v\sigma])$ with $v \to u \in \mathcal{E}^\succ \cup \mathcal{R}$.

A proof of the following form

$$s = s_0 \to s_1 \to s_2 \cdots \to s_m \leftarrow \cdots s_{n-1} \leftarrow s_n = t \ ,$$

is called a *rewrite proof.*

There exists a rewrite proof iff the equations converge wrt $\mathcal{R} \cup \mathcal{E}^\succ$. Furthermore, whenever $\mathcal{E}; \mathcal{R} \vdash \mathcal{E}'; \mathcal{R}'$ then the same equations are provable in $\mathcal{E}; \mathcal{R}$ as in $\mathcal{E}'; \mathcal{R}'$. However proofs may become simpler. In the following we define a proof order that precises our notion of "simpler". First, we recall the *encompassment order.* The term s *encompasses* t if $s = C[t\sigma]$ for some context C and some substitution σ. If s encompasses t, then s is larger than t in the encompassment order. The encompassment order is a preorder, cf. [3].

Definition 11.9. We define the *costs* of proof steps as follows:

$$\text{cost of } s[u] \; \rho \; s[v] := \begin{cases} (\{s[u]\}, u, \rho, s[v]) & \text{if } s[u] \succ s[v] \\ (\{s[v]\}, v, \rho, s[u]) & \text{if } s[v] \succ s[u] \\ (\{s[u], s[v]\}, \bot, \bot, \bot) & \text{otherwise} \end{cases}$$

These costs are compared by the lexicographic product of the following orders:

(i) the multiset extension of $\succ$,

(ii) the encompassment order,

(iii) some order with $\leftrightarrow \, > \, \to$ and $\leftrightarrow \, > \, \leftarrow$, and

(iv) the reduction order $\succ$.

Here $\bot$ is supposed to be minimal in all orders.

We call a derivation *fair* if each ordered critical pair $u = v \in \mathcal{E}_\infty \cup \mathcal{R}_\infty$ is an element of some $\mathcal{E}_i$.

The proof of the following theorem can be found in [6].

Theorem 11.6. *Let $(\mathcal{E}_0; \mathcal{R}_0), (\mathcal{E}_1; \mathcal{R}_1), \ldots$ be a fair ordered completion derivation with $\mathcal{R}_0 = \varnothing$; then the following is equivalent:*

(i) $s = t$ is a consequence of $\mathcal{E}_0$,

(ii) $s = t$ has a rewrite proof in $\mathcal{E}_\infty^\succ \cup \mathcal{R}_\infty$, and

(iii) $\exists\ i$ such that $s = t$ has a rewrite proof in $\mathcal{E}_i^\succ \cup \mathcal{R}_i$

The word problem becomes a refutation theorem proving problem once we consider the clause form of the negation of the word problem. Then the corresponding clause set contains positive unit equations from $\mathcal{E}$ and a ground disequation obtained by negation and Skolemisation of $s = t$.

Corollary 11.1. *Superposition with equations is sound and complete, that is, if $\mathcal{C}$ is the clause representation of the (negated) word problem $\mathcal{E} \models s = t$, then the saturation of $\mathcal{C}$ wrt to superposition (and equality resolution) contains $\Box$ iff $\mathcal{E} \models s = t$.*

Proof. Let $\mathcal{C}'$ denote a saturation of $\mathcal{C}$ and suppose $\Box \in \mathcal{C}'$. Then $\mathcal{E} \models s = t$ is due to soundness of superposition. Otherwise assume $\Box \notin \mathcal{C}'$. Then the equation $s = t$ does not have a proof in $\mathcal{C}'$. From Theorem 11.6 we conclude that $\mathcal{E} \not\models s = t$. $\Box$

11.4. Superposition Calculus

Unfortunately the *ordered paramodulation calculus* as defined in Section 11.2 is still to inefficient to be used. While the literals in the considered clauses are now ordered, no restriction on the way the equality $s = t$ is used in paramodulation is present. To overcome this one incorporates ideas from ordered rewriting and completion to combine ordered resolution and paramodulation to the *superposition calculus.* First, we restrict our attention to equational Horn logic.

Definition 11.10. An equational Horn clause $C \equiv (u_1 = v_1, \dots, u_k = v_k \to s = t)$ is *reductive* for $s \to t$ (with respect to a reduction order $\succ$) if s is strictly maximal in C, that is (i) $s \succ t$, (ii) for all i: $s \succ u_i$, and (iii) for all i: $s \succ v_i$.

If C is reductive for $s \to t$, we can write C as a *conditional rewrite rule* as follows: $u_1 = v_1, \dots, u_k = v_k \supset s \to t$. Here we use the alternative notation $\supset$ for logical implication to be able to keep the standard notion for rewrite rules.

Let $\mathcal{R}$ be a set of reductive clauses. Then $\mathcal{R}$ induces the rewrite relation $\to_\mathcal{R}$: $s \to_\mathcal{R} t$ if

(i) there exists a reductive clause $C \supset l \to r$,

(ii) there exists a substitution σ such that $s = l\sigma$ and $t = r\sigma$, and

(iii) for all equations $u' = v' \in C$: $u'\sigma \downarrow v'\sigma$ holds.

The next definition generalises critical pairs to conditional rewriting.

Definition 11.11.

$$\frac{C \supset s \to t \quad D \supset w[u] \to v}{(C, D \supset w[t] \to v)\sigma} \quad .$$

Here σ is a mgu of s and u and u is not a variable.

The proof of the next lemma and theorem can be found in [31]; for general reading material on conditional rewrite systems, see [43, Chapter 7].

Lemma 11.3. *A reductive conditional rewrite system is confluent iff all critical pairs converge.*

Theorem 11.7. *Let $\succ$ be a reduction order and let $\mathcal{C}$ be a set of reductive equational Horn clauses. Then the word problem for $\mathcal{C}$ is decidable if all critical pairs in $\mathcal{C}$ converge.*

We leave Horn logic, and consider full clause logic. First, we need to extend the underlying reduction order $\succ$ on terms to an order on literals and in particular on clauses. This is always possible in a way that we obtain a well-founded clause order $\succ_{\mathsf{C}}$ that is total on ground clauses, if the initial term order $\succ$ is well-founded and total on ground terms. For notational convenience we denote the obtained clause order $\succ_{\mathsf{C}}$ simply as $\succ$. The rules for the superposition calculus are given in Figure 11.5.

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

$$\frac{C \vee s = t \quad D \vee \neg A[s']}{(C \vee D \vee \neg A[t])\sigma} \qquad \frac{C \vee s = t \quad D \vee A[s']}{(C \vee D \vee A[t])\sigma}$$

$$\frac{C \vee s = t \quad D \vee u[s'] \neq v}{(C \vee D \vee u[t] \neq v)\sigma} \qquad \frac{C \vee s = t \quad D \vee u[s'] = v}{(C \vee D \vee u[t] = v)\sigma}$$

$$\frac{C \vee s \neq t}{C\sigma} \qquad \frac{C \vee u = v \vee s = t}{(C \vee v \neq t \vee u = t)\sigma}$$

The first two rules are called ordered resolution and ordered factoring respectively. They are restricted to atoms A and B that do not contain $=$ and the same order constraints hold as in Figure 11.1.
The last four rules are called *superposition rules*, the seventh is denoted as *equality resolution*, while the last one is called *equality factoring*.

– For the superposition rules: σ is a mgu of s and s', s' not a variable, $t\sigma \not\succeq s\sigma$, $v\sigma \not\succeq u[s']\sigma$, $(s = t)\sigma$ is strictly maximal with respect to $C\sigma$. Moreover $\neg A[s']$ and $u[s'] \neq v$ are maximal, while $A[s']$ and $u[s'] = v$ are strictly maximal with respect to $D\sigma$. And $(s = t)\sigma \not\succeq (u[s'] = v)\sigma$.

– For the equality resolution rule: σ is a mgu of s and t, and $(s \neq t)\sigma$ is maximal with respect to $C\sigma$.

– Finally for equality factoring: σ is mgu of s and u, $(s = t)\sigma$ is strictly maximal in $C\sigma$. And $(s = t)\sigma \not\succeq (u = v)\sigma$.

Figure 11.5.: Superposition Calculus

Definition 11.12. Let $\mathcal{C}$ be a set of clauses. We define the *superposition operator* $\mathsf{Res}_{\mathsf{SP}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{SP}}(\mathcal{C}) := \{D \mid D \text{ is conclusion of inference in Fig. 11.5 with premises in } \mathcal{C}\} \ .$$

The n^{th} (unrestricted) iteration $\mathsf{Res}^n_{\mathsf{SP}}$ ($\mathsf{Res}^*_{\mathsf{SP}}$) of the operator $\mathsf{Res}_{\mathsf{SP}}$ is defined as above.

The following example clarifies the need for the seemingly artificial equality factoring rule. If we delete this rule from the superposition calculus, we obtain *strict superposition.*

Example 11.2. Consider the following set of clauses $\mathcal{C}$:

$$\begin{aligned}
&\mathsf{c} \neq \mathsf{d} \\
&\mathsf{b} = \mathsf{d} \\
&\mathsf{a} \neq \mathsf{d} \lor \mathsf{a} = \mathsf{c} \\
&\mathsf{a} = \mathsf{b} \lor \mathsf{a} = \mathsf{d}
\end{aligned}$$

It is easy to see that $\mathcal{C}$ is unsatisfiable. However this contradiction cannot be derived by strict superposition if based on the term order $\succ$, where $\mathsf{a} \succ \mathsf{b} \succ \mathsf{c} \succ \mathsf{d}$. The only derivable clause is the following tautology:

$$\mathsf{a} \neq \mathsf{d} \lor \mathsf{b} = \mathsf{c} \lor \mathsf{a} = \mathsf{d} \ .$$

In order to show (refutational) completeness of the superposition calculus it is no longer possible to adapt the standard completeness proof for first-order logic as we did before. Furthermore it is also not possible to adapt the frequently used semantic tree method. Instead one has to re-design a suitable variant of the model existence theorem taking the underlying reduction order $\succ$ into account. For this on makes use of so called *candidate models.* First, we explain the general set-up of the completeness proof and state the necessary results for instantiating the framework for the superposition calculus.

Let $\mathcal{O}$ be a clause inference operator and let $\mathcal{I}$ denote a mapping that assigns to each ground clause set $\mathcal{C}$ an equality Herbrand interpretation, the *candidate model* $\mathcal{I}_{\mathcal{C}}$. If $\mathcal{I}_{\mathcal{C}} \models \mathcal{C}$, then the candidate model is indeed a model. Otherwise, suppose $\mathcal{I}_{\mathcal{C}} \not\models \mathcal{C}$. Then there exists a minimal counter-example, denoted as C. We say the inference operator $\mathcal{O}$ has the *reduction property* if for all clause sets $\mathcal{C}$ and all minimal counter-examples C for $\mathcal{I}_{\mathcal{C}}$ there exists an inference I from $\mathcal{C}$ that is admissible with respect to $\mathcal{O}$ of the following form:

$$\frac{C_1 \quad \ldots \quad C_n \quad C}{D} \ ,$$

where for all $i = 1, \ldots, n$: $C_i \in \mathcal{C}$ such that $\mathcal{I}_{\mathcal{C}} \models C_i$. Furthermore $\mathcal{I}_{\mathcal{C}} \not\models D$ and $C \succ D$. The next theorem is a consequence of the definitions.

Theorem 11.8. *Let $\mathcal{O}$ be a sound inference operator that has the reduction property and let $\mathcal{C}$ be a ground clause that is saturated with respect to $\mathcal{O}$, that is, $\mathcal{O}(\mathcal{C}) \subseteq \mathcal{C}$. Then the clause set $\mathcal{C}$ is unsatisfiable iff $\Box \in \mathcal{C}$.*

Without loss of generality we assume in the following that our basic language contains only the equality sign $=$ as predicate constant. Thus term interpretations $\mathcal{I}$ are respresentable as convergent ground rewrite systems (with respect to the reduction order $\succ$). This allows to describe truth in $\mathcal{I}$ through rewriting: a (ground) clause C is true in $\mathcal{I}$ iff whenever all negated equations in C have rewrite proofs then also some of the unnegated equations has a rewrite proof. Furthermore a reductive (general) clause can be conceived as a conditional rewrite rule, where negation is interpreted as non-derivability. For the next definition we tactily assume the existence of a clause $\top$ larger than all considered clauses. This allows us to denote every clause set as a set $\mathcal{C}_C$ for some clause C.

Definition 11.13. Let $\mathcal{C}$ denote a clause set; we set $\mathcal{C}_C := \{D \in \mathcal{C} \mid C \succ D\}$ and define a mapping $\mathcal{I}$ that assigns to all clause sets $\mathcal{C}_C$ a rewrite system $\mathcal{I}_C$.

The definition is by induction on $\succ$. $\mathcal{I}_C$ is the set of all ground rewrite rules $s \to t$ such that there exists a clause $D = C' \vee s = t \in \mathcal{C}$ with $C \succ D$ and

(i) D is reductive for $s = t$,

(ii) D is counter-example for $\mathcal{I}_D$,

(iii) s is in normal form with respect to $\mathcal{I}_D$, and

(iv) C' is counter-example for $\mathcal{I}_D \cup \{s = t\}$

If such a clause D exists, it is called *productive*.

Note that $\mathcal{I}_C$ is convergent (with respect to $\succ$) by construction. Furthermore, we have the following result, cf. [7].

Theorem 11.9. *Let $\mathcal{C}$ be a ground clause set and let C be a minimal counter-example to the candidate model $\mathcal{I}_{\mathcal{C}}$ constructed as above. Then there exists a clause $D \in \mathsf{Res}_{\mathsf{SP}}(\mathcal{C})$ such that $C \succ D$ and D is also a counter-example.*

As a consequence of this theorem, the superposition calculus has the reduction property and hence is refutational complete for *ground* clauses. In order to obtain completeness also for non-ground inferences, we need a suitable variant of the lifting lemmas. According to Example 11.1 this is a non-trivial task. But, we observe that the initial ground inference is actually *redundant*, for a suitable definition of redundancy of inferences.

Definition 11.14. A ground clause C is *redundant* with respect to a ground clause set $\mathcal{C}$ if there exists clauses C_1, ..., C_k in $\mathcal{C}$ such that

$$C_1, \ldots, C_k \models C \ ,$$

where for all $i = 1, \ldots, k$: $C \succ C_i$. Let $\mathcal{R}(\mathcal{C})$ denote the set of redundant clauses with respect to $\mathcal{C}$. Furthermore a ground inference

$$\frac{C_1 \quad \ldots \quad C_n \quad C}{D}$$

is *redundant* (with respect to $\mathcal{C}$) if either $D \succeq C$ or there exists clauses $D_1, \dots, D_k$ in $\mathcal{C}_C$ such that

$$D_1, \dots, D_k, C_1, \dots, C_n \models D \ .$$

By $\mathcal{R}_{\mathcal{O}}(\mathcal{C})$ we denote the set of redundant inferences in $\mathcal{O}$ with respect to $\mathcal{C}$.

Note that an inference is redundant for any (ground) clause set $\mathcal{C}$ for which the conclusion of the inference is contained in $\mathcal{C} \cup \mathcal{R}(\mathcal{C})$. Furthermore note that the redundancy operators $\mathcal{R}$ and $\mathcal{R}_{\mathcal{O}}$ are monotone: if $\mathcal{C} \subseteq \mathcal{C}'$, then $\mathcal{R}(\mathcal{C}) \subseteq \mathcal{R}(\mathcal{C}')$ and $\mathcal{R}_{\mathcal{O}}(\mathcal{C}) \subseteq \mathcal{R}_{\mathcal{O}}(\mathcal{C}')$. Moreover if $\mathcal{C}' \subseteq \mathcal{R}(\mathcal{C})$, then $\mathcal{R}(\mathcal{C}) \subseteq \mathcal{R}(\mathcal{C} \setminus \mathcal{C}')$ and $\mathcal{R}_{\mathcal{O}}(\mathcal{C}) \subseteq \mathcal{R}_{\mathcal{O}}(\mathcal{C} \setminus \mathcal{C}')$.

We say a (ground) clause set is *saturated upto redundancy* if all inferences from non-redundant premises are redundant. The next result generalises Theorem 11.8 to the proposed notion of redundancy.

Theorem 11.10. *Let $\mathcal{O}$ be a sound inference operator that has the reduction property and let $\mathcal{C}$ be a ground clause that is saturated upto redundancy. Then the clause set $\mathcal{C}$ is unsatisfiable iff $\Box \in \mathcal{C}$.*

Proof. Suppose $\Box \notin \mathcal{C}$ and let $\mathcal{C}' := \mathcal{C} \setminus \mathcal{R}(\mathcal{C})$. We consider the candidate model $\mathcal{I}_{\mathcal{C}'}$. Suppose $\mathcal{I}_{\mathcal{C}}'$ is a model of $\mathcal{C}'$. Then we are done, as any model of $\mathcal{C}'$ is a model of $\mathcal{C}$.

Thus suppose otherwise $\mathcal{I}_{\mathcal{C}'} \not\models \mathcal{C}'$. Hence there exists a minimal counter-example C. Since $\mathcal{O}$ has the reduction property there is an inference I from $\mathcal{C}'$ with respect to $\mathcal{O}$ such that I has premises $C_1, \dots, C_n, C$ and conclusion D, where D is a smaller counter-example to $\mathcal{I}_{\mathcal{C}'}$ and for all $i = 1, \dots, n$: $\mathcal{I}_{\mathcal{C}'} \models C_i$.

Due to the assumption that $\mathcal{C}$ is saturated, the inference I has to be redundant. Thus there are clauses $D_j \in \mathcal{C}_C$ such that $D_1, \dots, D_k, C_1, \dots, C_n \models D$ holds. Wlog. all D_j are non-redundant and true in $\mathcal{I}_{\mathcal{C}'}$. The former can be achieved by an iteration of the construction and the latter follows as C is the *minimal* counter-example to $\mathcal{I}_{\mathcal{C}'}$. Then for all $j = 1, \dots, k$: $\mathcal{I}_{\mathcal{C}'} \models D_j$ and for all $i = 1, \dots, n$: $\mathcal{I}_{\mathcal{C}'} \models C_i$, from which $\mathcal{I}_{\mathcal{C}'} \models D$ is immediate, which is a contradiction.

In sum $\mathcal{I}_{\mathcal{C}'}$ has to be a model of $\mathcal{C}'$ (and thus a model of $\mathcal{C}$). $\Box$

Lemma 11.4. *Non-redundant superposition inferences are liftable.*

Proof. Let $\mathcal{C}$ be a (non-ground) clause set. Wlog we suppose the existence of a ground superposition inference I (with respect to $\mathsf{Gr}(\mathcal{C})$) where we replace a substitution position:

$$\frac{C' \vee s = t \quad D' \vee L'[s]}{C' \vee D' \vee L'[t]} \ ,$$

where $D' \vee L'[s]$ is a (ground) instance of some clause $D \vee L[x] \in \mathcal{C}$ such that $(D \vee L[x])\tau = D \vee L'[s]$ and $x\tau = u[s]$ for a substitution τ. We define the reduced substitution ρ as follows:

$$\rho(y) := \begin{cases} u[t] & \text{if } y = x \\ \tau(y) & \text{otherwise .} \end{cases}$$

Then $(D \lor L[x])\rho \in \mathsf{Gr}(\mathcal{C})$ and $(D \lor L[x])\rho, C' \lor s = t \models C' \lor D' \lor L'[t]$. As $D' \lor L'[s] \succ (D \lor L[x])\rho$ by admissibility of the inference, we obtain that I is redundant (with respect to $\mathsf{Gr}(\mathcal{C})$). $\square$

As a consequene of Theorems 11.10 and 11.9 together with Lemma 11.4 we finally obtain refutational completeness of the superposition calculus.

Theorem 11.11. *Superposition is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\square \in \mathsf{Res_{SP}}^*(\mathcal{C})$.*

Problems

Problem 11.1. Consider the unification algorithm given in Figure 10.3. Show that this algorithm produced exponential large terms in the worst case.

Problem 11.2. Consider the (propositional) clauses:

$$C_1 \lor A \qquad C_2 \lor \neg A \lor B \qquad C_3 \lor \neg B$$

(i) Give two different resolution derivations of the clause $C_1 \lor C_2 \lor C_3$.

(ii) Can this behaviour be avoided by the use of ordered resolution?

Problem 11.3. Consider the following clause set:

$$\mathsf{P}(x) \lor \mathsf{Q}(x) \quad \neg\mathsf{P}(x) \lor \mathsf{Q}(\mathsf{f}(y)) \quad \mathsf{P}(x) \lor \neg\mathsf{Q}(f(x)) \quad \neg\mathsf{P}(x) \lor \neg\mathsf{Q}(x) .$$

– Decide its satisfiability using ordered resolution.

– Consider the variant where the clause $\neg\mathsf{P}(x) \lor \mathsf{Q}(\mathsf{f}(y))$ is replaced by $\neg\mathsf{P}(x) \lor \mathsf{Q}(\mathsf{f}(x))$ and again decide satisfiability.

Problem 11.4. Use (the propositional variant) of ordered resolution to show Theorem 2.3 for propositional logic.

Hint: Let $A \to C$ be the considered implication. Then choose the order $\succ$ underlying ordered resolution such that those variables that occur in A but not in C are maximal.

Problem 11.5. Formulate and prove the lifting lemmas for ordered resolution.

Problem 11.6. Show the following claim:

> *Let S denote the set of all consistent ground clause sets with respect to paramodulation. Then S has the satisfaction properties.*

Problem 11.7. Prove the adapted lifting lemma for paramodulation, Lemma 11.2.

Problem 11.8. Complete the proof of Theorem 11.2.

Problem 11.9. Show that all ordered completion inference rules simplify proofs with respect to the cost measure for proofs defined in Definition 11.9.

Problem 11.10. Consider the following clause set:

$$\begin{array}{lll} \mathsf{f}(\mathsf{f}(x)) \neq x \lor \mathsf{f}(x) = \mathsf{g}(x) & \mathsf{a} \neq \mathsf{c} \lor \mathsf{f}(\mathsf{c}) = \mathsf{c} & \\ \mathsf{a} = \mathsf{b} & \mathsf{b} = \mathsf{c} & \mathsf{g}(\mathsf{a}) \neq \mathsf{a} \end{array}$$

Show that the clause set is unsatisfiable, using superposition.

Problem 11.11. Show the following properties:

– If $\mathcal{C} \subseteq \mathcal{C}'$, then $\mathcal{R}(\mathcal{C}) \subseteq \mathcal{R}(\mathcal{C}')$ and $\mathcal{R}_{\mathcal{O}}(\mathcal{C}) \subseteq \mathcal{R}_{\mathcal{O}}(\mathcal{C}')$.

– If $\mathcal{C}' \subseteq \mathcal{R}(\mathcal{C})$, then $\mathcal{R}(\mathcal{C}) \subseteq \mathcal{R}(\mathcal{C} \setminus \mathcal{C}')$ and $\mathcal{R}_{\mathcal{O}}(\mathcal{C}) \subseteq \mathcal{R}_{\mathcal{O}}(\mathcal{C} \setminus \mathcal{C}')$

12. Applications of Automated Reasoning

In this chapter we study two applications of automated reasoning machinery. The first (see Sections 12.1–12.2) is concerned with security of protocols. The second presents McCune's proof of Robbin's conjecture (see Section 12.4).

12.1. Neuman-Stubblebine Key Exchange Protocol

The Neuman-Stubblebine key exchange protocol (see [40]) aims to establish a secure key between two agents that already share secure keys with a trusted third party. In this chapter we give a formalisation of this protocol in first-order logic and show how it can be analysed by a state-of-the-art theorem prover for first-order logic.

In Section 12.1 we describe the protocol and indicate how it should work. In Section 12.2 we mention a possible attack, which makes the protocol erroneous. Further, we indicate how this attack can be prevented. Finally, in Section 12.3 we show how the protocol can be formalised in first-order logic.

The protocol aims to establish a secure key between two agents *Alice* and *Bob* that already share secure keys with a trusted third party, the *server*. We use the following notations:

– A is the identifier of Alice.

– B is the identifier of Bob.

– T is the identifier of the server.

– K_{at} is the symmetric key shared between Alice and the server.

– K_{bt} is the symmetric key shared between Bob and the server.

– K_{ab} is the symmetric key shared between Alice and Bob to be established.

– N_{a} denotes a nonce created by Alice. Here a *nonce* is a fresh number used to prevent replay attacks.

– N_{b} denotes a nonce created by Bob.

– $\mathsf{E}_{key}(message)$ denotes the encryption of *message* using the key *key*.

– Time defines the time span of the validity of the key $\mathsf{K_{ab}}$.

The protocol proceeds as follows, where we write $\mathsf{A} \longrightarrow \mathsf{B}\colon M$ when Alice sends Bob the message M. Further, message composition is denoted by ",".

(i) $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{A}, \mathsf{N_a}$, that is, Alice sends her identifier and a freshly generated nonce.

(ii) $\mathsf{B} \longrightarrow \mathsf{T}\colon \mathsf{B}, \mathsf{E}_{\mathsf{K_{bt}}}(A, \mathsf{N_a}, \mathsf{Time}), \mathsf{N_b}$, that is, Bob encrypts the triple $(\mathsf{A}, \mathsf{N_a}, \mathsf{Time})$ using his shared key with the server and sends this together with his identity and a freshly generated nonce.

(iii) $\mathsf{T} \longrightarrow \mathsf{A}\colon \mathsf{E}_{\mathsf{K_{at}}}(\mathsf{B}, \mathsf{N_a}, \mathsf{K_{ab}}, \mathsf{Time}), \mathsf{E}_{\mathsf{K_{bt}}}(\mathsf{A}, \mathsf{K_{ab}}, \mathsf{Time}), \mathsf{N_b}$, that is, the server generates the shared key $\mathsf{K_{ab}}$ and sends it encrypted to Alice using the shared key. Furthermore he encrypts the shared key with the key shared with Bob, which is also sent to Alice. Finally, the nonce $\mathsf{N_b}$ is part of the message to Alice.

(iv) $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{E}_{\mathsf{K_{bt}}}(\mathsf{A}, \mathsf{K_{ab}}, \mathsf{Time}), \mathsf{E}_{\mathsf{K_{ab}}}(\mathsf{N_b})$, that is, Alice encrypts Bob's nonce with the new key and forwards part of the message to Bob.

After reception of the message from Alice, Bob can first extract the shared key $\mathsf{K_{ab}}$ and then verifies that the key comes from Alice by decrypting $\mathsf{E}_{\mathsf{K_{ab}}}(\mathsf{N_b})$.

12.2. The Attack

The behaviour of a possible intruder (denoted as I) is governed by the following assumptions.

(i) The intruder can record all sent messages.

(ii) The intruder can send messages and can forge the sender of a message.

(iii) The intruder can encrypt messages, when he finds out a key.

(iv) The intruder has no access to information private to Alice, Bob, or the server.

(v) The intruder cannot break any secure key.

Based on these assumptions the intruder can impersonate Alice and the server and thus convince Bob to share all secrets with him as follows:

(i) $\mathsf{I}(\mathsf{A}) \longrightarrow \mathsf{B}\colon \mathsf{A}, \mathsf{N_a}$.

(ii) $\mathsf{B} \longrightarrow \mathsf{I}(\mathsf{T})\colon \mathsf{B}, \mathsf{E}_{\mathsf{K_{bt}}}(\mathsf{A}, \mathsf{N_a}, \mathsf{Time}), \mathsf{N_b}$.

(iii) $I(\mathsf{A}) \longrightarrow \mathsf{B}\colon \mathsf{E}_{\mathsf{K_{bt}}}(\mathsf{A}, \mathsf{N_a}, \mathsf{Time}), \mathsf{E}_{\mathsf{N_a}}(\mathsf{N_b})$.

Here $\mathsf{I(A)}$ means that the intruder impersonates Alice, while $\mathsf{I(T)}$ means that the intruder plays the role of the server.

The intruder only needs to send the message $\mathsf{E}_{\mathsf{K}_{\mathsf{bt}}}(A, \mathsf{N_a}, \mathsf{Time})$ back to Bob, and uses the nonce $\mathsf{N_a}$ to encrypt the message $\mathsf{E}_{\mathsf{N_a}}(\mathsf{N_b})$. From Bob's point of view the nonce $\mathsf{N_a}$ is actually the new shared key $\mathsf{K_{ab}}$ and everything seems to be in order.

This possible attack was first found by Hwang et al. (see [30]) who already described a solution to this attack. It suffices to distinguish nonces and keys, so that they cannot be confused.

12.3. Formalisation in First-Order

Following [55] we formalise the set of messages sent during the execution of the protocol. The formalisation makes use of unary predicate symbols only. This is necessary to make sure that the obtained consequence relations can be verified automatically.

We start with fixing the first-order language $\mathcal{L}$ used and then consider each of the four messages sent during the protocol in turn. We assert that $\mathcal{L}$ contains the following individual constants:

$$\mathsf{a} \quad \mathsf{b} \quad \mathsf{t} \quad \mathsf{na} \quad \mathsf{at} \quad \mathsf{bt}\ ,$$

where a, b, t are to be interpreted as the identifiers A, B, and T, respectively. The constant na refers to Alics's nonce and at (bt) represents the key $\mathsf{K_{at}}$ ($\mathsf{K_{bt}}$).

Further, $\mathcal{L}$ contains the following function constants:

$$\mathsf{nb} \quad \mathsf{tb} \quad \mathsf{kt} \quad \mathsf{key} \quad \mathsf{sent} \quad \mathsf{pair} \quad \mathsf{triple} \quad \mathsf{encr} \quad \mathsf{quadr}\ ,$$

where nb, tb, kt are unary and compute Bob's fresh nonce and the time-stamp Time, while kt formalises the computation of the new key by the server. The symbols key, pair, encr are binary, sent, triple are ternary, and quadr is 4-ary. These latter symbols essentially serve as containers.

Finally, $\mathcal{L}$ contains the following predicate constants:

$$\mathsf{Ak} \quad \mathsf{Bk} \quad \mathsf{Tk} \quad \mathsf{P} \quad \mathsf{M} \quad \mathsf{Fresh} \quad \mathsf{Nonce} \quad \mathsf{Store_a} \quad \mathsf{Store_b}\ .$$

Here the constants Ak, Bk, Tk will be used in conjunction with the function symbol key to assert the existence of shared keys. The constant P will only be true for principals of the protocol and M is used to encode the messages sent. Furthermore Fresh asserts that its argument is a fresh nonce. The latter is necessary, as we assume that Bob is only interested in fresh nonces. The predicate Nonce denotes that its argument is a nonce and the predicates $\mathsf{Store_a}$, $\mathsf{Store_b}$ denote information that is in the store of Alice or Bob.

To simplify the readability of the formalisation, we indicate the type of a bound variable in its name as subscript. For example the bound variable x_{na} indicates that this variable plays the role of the nonce $\mathsf{N_a}$ in the protocol. This is only a notational simplification and doesn't affect the semantics.

12.3.1. $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{A}, \mathsf{N_a}$

The first message of Alice is represented by the following set of formulas

1: $\mathsf{Ak(key(at, t))}$

2: $\mathsf{P(a)}$

3: $\mathsf{M(sent(a, b, pair(a, na)))} \wedge \mathsf{Store_a(pair(b, na))}$

12.3.2. $\mathsf{B} \longrightarrow \mathsf{T}\colon \mathsf{B}, \mathsf{E_{K_{bt}}(A, N_a, Time)}, \mathsf{N_b}$

The second message of Bob to the server is represented by the following set of formulas. The formalisation asserts that Bob is only sending a message if he has received a message from Alice.

4: $\mathsf{Bk(key(bt, t))}$

5: $\mathsf{P(b)}$

6: $\mathsf{Fresh(na)}$

7: $\forall x_{\mathsf{a}} \forall x_{\mathsf{na}} \, (\mathsf{M(sent(}x_{\mathsf{a}}, \mathsf{b}, \mathsf{pair(}x_{\mathsf{a}}, x_{\mathsf{na}}))) \wedge \mathsf{Fresh(}x_{\mathsf{na}}) \rightarrow$
$\rightarrow \mathsf{Store_b(pair(}x_{\mathsf{a}}, x_{\mathsf{na}})) \wedge$
$\wedge \mathsf{M(sent(b, t, triple(b, nb(}x_{\mathsf{na}}), \mathsf{encr(triple(}x_{\mathsf{a}}, x_{\mathsf{na}}, \mathsf{tb(}x_{\mathsf{na}})), \mathsf{bt})))))$

Formula 7 expresses that Bob reacts to any message sent by any principal that need not be known in advance. Hence the formalisation is slightly more general than the protocol and allows repeated execution.

12.3.3. $\mathsf{T} \longrightarrow \mathsf{A}\colon \mathsf{E_{K_{at}}(B, N_a, K_{ab}, Time)}, \mathsf{E_{K_{bt}}(A, K_{ab}, Time)}, \mathsf{N_b}$

On seeing the second message, generated by the right-hand side of the implication in formula 7, the server sends the third message. This is formalised as follows.

8: $\mathsf{Tk(key(at, a))} \wedge \mathsf{Tk(key(bt, b))}$

9: $\mathsf{P(t)}$

10: $\forall x_{\mathsf{b}} \forall x_{\mathsf{nb}} \forall x_{\mathsf{a}} \forall x_{\mathsf{na}} \forall x_{\mathsf{time}} \forall x_{\mathsf{bt}} \forall x_{\mathsf{at}}$
$(\mathsf{M(sent(}x_{\mathsf{b}}, \mathsf{t}, \mathsf{triple(}x_{\mathsf{b}}, x_{\mathsf{nb}}, \mathsf{encr(triple(}x_{\mathsf{a}}, x_{\mathsf{na}}, x_{\mathsf{time}}), x_{\mathsf{bt}})))) \wedge \mathsf{Tk(key(}x_{\mathsf{bt}}, x_{\mathsf{b}})) \wedge$
$\wedge \mathsf{Tk(key(}x_{\mathsf{at}}, x_{\mathsf{a}})) \wedge \mathsf{Nonce(}x_{\mathsf{na}}) \rightarrow \mathsf{M(sent(t,} x_{\mathsf{a}},$
$\mathsf{triple(encr(quadr(}x_{\mathsf{b}}, x_{\mathsf{na}}, \mathsf{kt(}x_{\mathsf{na}}), x_{\mathsf{time}}), x_{\mathsf{at}}),$
$\mathsf{encr(triple(}x_{\mathsf{a}}, \mathsf{kt(}x_{\mathsf{na}}), x_{\mathsf{time}}), x_{\mathsf{bt}}), x_{\mathsf{nb}}))))$

11: $\mathsf{Nonce(na)}$

12: $\forall x \neg \mathsf{Nonce(kt(}x))$

13: $\forall x \, (\mathsf{Nonce(tb(}x)) \wedge \mathsf{Nonce(nb(}x)))$

The last 3 formulas represent that the server will not accept his generated key as nonce. Accordingly the assumption for sending his message has been strengthened. This requirement is not part of the protocol, but prevents that the intruder can generate arbitrarily many keys. These could possible be used to learn the key.

12.3.4. $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{E}_{\mathsf{K}_{\mathsf{bt}}}(\mathsf{A}, \mathsf{K}_{\mathsf{ab}}, \mathsf{Time}), \mathsf{E}_{\mathsf{K}_{\mathsf{ab}}}(\mathsf{N}_{\mathsf{b}})$

Alice sees the server message and tries to decrypt the first part of the message using the secure key at she shares with the server. If this succeeds she checks her store that this part of the message starts with the same identifier, she sent her first message to. In this case she sends the fourth message.

14: $\forall x_{\mathsf{nb}} \forall x_{\mathsf{k}} \forall x_{\mathsf{m}} \forall x_{\mathsf{b}} \forall x_{\mathsf{na}} \forall x_{\mathsf{time}}$
$((\mathsf{M}(\mathsf{sent}(\mathsf{t}, \mathsf{a}, \mathsf{triple}(\mathsf{encr}(\mathsf{quadr}(x_{\mathsf{b}}, x_{\mathsf{na}}, x_{\mathsf{k}}, x_{\mathsf{time}}), \mathsf{at}), x_{\mathsf{m}}, x_{\mathsf{nb}}))) \wedge$
$\wedge\, \mathsf{Store}_{\mathsf{a}}(\mathsf{pair}(x_{\mathsf{b}}, x_{\mathsf{na}}))) \to \mathsf{M}(\mathsf{sent}(\mathsf{a}, x_{\mathsf{b}}, \mathsf{pair}(x_{\mathsf{m}}, \mathsf{encr}(x_{\mathsf{nb}}, x_{\mathsf{k}})))) \wedge \mathsf{Ak}(\mathsf{key}(x_{\mathsf{k}}, x_{\mathsf{b}})))$

15: $\forall x_{\mathsf{k}} \forall x_{\mathsf{a}} \forall x_{\mathsf{na}}$
$((\mathsf{M}(\mathsf{sent}(x_{\mathsf{a}}, \mathsf{b}, \mathsf{pair}(\mathsf{encr}(\mathsf{triple}(x_{\mathsf{a}}, x_{\mathsf{k}}, \mathsf{tb}(x_{\mathsf{na}})), \mathsf{bt}), \mathsf{encr}(\mathsf{nb}(x_{\mathsf{na}}), x_{\mathsf{k}})))) \wedge$
$\wedge\, \mathsf{Store}_{\mathsf{b}}(\mathsf{pair}(x_{\mathsf{a}}, x_{\mathsf{na}}))) \to \mathsf{Bk}(\mathsf{key}(x_{\mathsf{k}}, x_{\mathsf{a}})))$

We collect these 15 sentences into the set $\mathcal{G}$. Then it is not difficult to verify by hand that the following consequence is valid:

$$\mathcal{G} \models \exists x (\mathsf{Ak}(\mathsf{key}(x, \mathsf{a})) \wedge \mathsf{Bk}(\mathsf{key}(x, \mathsf{b}))) \ .$$

This shows that the protocol terminates with the desired result that Alice and Bob share a symmetric key. Furthermore completeness tells us that this fact can also be formally proven, for example in natural deduction.

Fact 12.1. The formula $\exists x (\mathsf{Ak}(\mathsf{key}(x, \mathsf{a})) \wedge \mathsf{Bk}(\mathsf{key}(x, \mathsf{b})))$ is derivable from $\mathcal{G}$ fully automatically by SPASS in less than a second.

In the remainder of the section, we formalise the behaviour of the intruder. Recall the assumptions made above in Section 12.2. These are formalised as follows.

We extend our base language $\mathcal{L}$ by the predicate constants Ik and Im. Ik will be used to express that the intruder has learnt a key of another principal and Im states that a message is recorded or faked by the intruder.

16: $\forall x_{\mathsf{a}} \forall x_{\mathsf{b}} \forall x_{\mathsf{m}}\, (\mathsf{M}(\mathsf{sent}(x_{\mathsf{a}}, x_{\mathsf{b}}, x_{\mathsf{m}})) \to \mathsf{Im}(x_{\mathsf{m}}))$
17: $\forall u \forall v\, (\mathsf{Im}(\mathsf{pair}(u, v)) \to \mathsf{Im}(u) \wedge \mathsf{Im}(v))$
18: $\forall u \forall v \forall w\, (\mathsf{Im}(\mathsf{triple}(u, v, w)) \to (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w)))$
19: $\forall u \forall\ v \forall w \forall z\, (\mathsf{Im}(\mathsf{quadr}(u, v, w, z)) \to (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w) \wedge \mathsf{Im}(z)))$
20: $\forall u \forall v\, (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \to \mathsf{Im}(\mathsf{pair}(u, v)))$
21: $\forall u \forall v \forall w\, ((\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w)) \to \mathsf{Im}(\mathsf{triple}(u, v, w)))$
22: $\forall u \forall v \forall w \forall z\, ((\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w) \wedge \mathsf{Im}(z)) \to \mathsf{Im}(\mathsf{quadr}(u, v, w, z)))$
23: $\forall x \forall y \forall u\, ((\mathsf{P}(x) \wedge \mathsf{P}(y) \wedge \mathsf{Im}(u)) \to \mathsf{M}(\mathsf{sent}(x, y, u)))$
24: $\forall u \forall v\, ((\mathsf{Im}(u) \wedge \mathsf{P}(v)) \to \mathsf{Ik}(\mathsf{key}(u, v)))$
25: $\forall u \forall v \forall w\, ((\mathsf{Im}(u) \wedge \mathsf{Ik}(\mathsf{key}(v, w)) \wedge \mathsf{P}(w)) \to \mathsf{Im}(\mathsf{encr}(u, v)))$

Formula 24 represents that anything the intruder receives can be used as a key, while Formula 25 represents that the intruder can use such a key to encrypt messages. Based on this formalisation we can automatically detect that there is a problem with this protocol. Let $\mathcal{H}$ denote the extension of the formula set $\mathcal{G}$ by the sentences 16–25.

Fact 12.2. The formula $\exists x(\mathsf{Ik}(\mathsf{key}(x, \mathsf{b})) \land \mathsf{Bk}(\mathsf{key}(x, \mathsf{a})))$ is derivable from $\mathcal{H}$ fully automatically by SPASS in less than a second.

This fact expresses that the attack reported in [30] can indeed by detected fully automatically. As mentioned above we can get rid of this attack, if nonces are no longer to be confused with keys. If this is suitably formalised (see [55]), then one can formally and automatically verify that the (updated) protocol is safe.

12.4. Robbin's Conjecture

In this section we report on a remarkable achievement of the automated deduction community: the automatic proof of Robbin's conjecture. Robbin's conjecture states that all Robbin's algebra (see below for the definition) are Boolean algebra. This question, originally posed in the early 1930ies was positively answered in the 1990ies by William McCune [37, 15] using the equational prover EQP.

We recall the standard definition of Boolean algebras.

Definition 12.1. $\mathcal{B} = \langle B; +, \cdot, \overline{}, 0, 1\rangle$ is a *Boolean algebra* if

(i) $\langle B; +, 0\rangle$ and $\langle B; \cdot, 1\rangle$ are commutative monoids

(ii) $\forall\ a, b, c \in B$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \qquad a + (b \cdot c) = (a + b) \cdot (a + c)$$

(iii) $\forall\ a \in B$: $a + \overline{a} = 1$ and $a \cdot \overline{a} = 0$

$\overline{a}$ is called *complement* (or *negation*) of a

The next definition introduces Huntington's Basis, an alternative axiomatisation of Boolean algebras.

Definition 12.2. Consider the following axioms:

$$\begin{aligned} x + y &= y + x && \text{commutativity} \\ (x + y) + z &= x + (y + z) && \text{associativity} \\ \overline{\overline{x} + y} + \overline{\overline{x} + \overline{y}} &= x && \textit{Huntington equation}\ . \end{aligned}$$

Theorem 12.1 (Huntington)**.** *The provided axioms form a minimal axiomatisation of Boolean algebras, that is all axioms are independent from each other.*

The next definition introduces an alternative to the Huntingtion equation.

Definition 12.3. *Robbins equation* is defined as follows:

$$\overline{\overline{x+y}+\overline{x+\overline{y}}} = x \; . \tag{12.1}$$

Based on (12.1) we define a *Robbins algebra* as an algebra satisfying (i) commutativity (ii) associativity and (iii) Robbins equation.

In the sequel of the chapter we will answer the following question positively: Can Huntington's equation safely replaced by Robbins equation and still yields an axiomatisation of Boolean algebras? Put differently, we clarify that any Robbins algebra is Boolean. This proof was not found by a human, but by the equational prover EQP. Still the proof requires a number of number of auxiliary lemmas. We will not prove these lemmas, but rather report on the performance of EQP on these problems, see [37].

Lemma 12.1. *A Robbins algebra satisfying $\exists x(x+x=x)$ is a Boolean algebra.*

Proof (Sketch). This is automatically provable by EQP in about 5 seconds. □

Lemma 12.2. *A Robbins algebra satisfying $\exists x \exists y(x+y=x)$ is a Boolean algebra.*

Proof (Sketch). Originally the lemma was manually proven by Steve Winker.[1] Based on the above lemma EQP can find a proof in about 40 minutes. □

Lemma 12.3. *A Robbins algebra satisfying $\exists x \exists y(\overline{x+y} = \overline{x})$ is a Boolean algebra.*

Proof (Sketch). Originally the Lemma was manually proven by Steve Winker, but can also be proven automatically by EQP. However 8 *days* were necessary to prove this lemma. □

The next lemma is the main lemma in the proof of the theorem.

Lemma 12.4. *All Robbin algebras satisfy $\exists x \exists y(x+y=x)$.*

Proof. The proof by EQP required a very carefully crafted and incomplete heuristics, for examples SPASS cannot handle the problem in 12 hours. We present some of the crucial steps of the proof in Figure 12.1. The function $\mathsf{n}(\cdot)$ represents complement.

The last line in the proof asserts $\exists x \exists y(x+y=x)$. Furthermore the line fifth line from below proves $\exists x \exists y(\overline{x+y} = \overline{x})$, a lemma also first manually proven by Winker. □

Remark that this lemma is beyond the scope of SPASS.

[1] Steve Winker was a student of Larry Wos and the first to think of attacking Robbins problem by automated deduction.

$$
\begin{array}{ll}
\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+\mathsf{n}(x+y))=y & \text{7, (R)}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(x+y)+\mathsf{n}(x)+y)+y)=\mathsf{n}(x+y) & \text{10, [7} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+x+y)+y)=\mathsf{n}(\mathsf{n}(x)+y) & \text{11, [7} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+x+2y)+\mathsf{n}(\mathsf{n}(x)+y))=y & \text{29, [11} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+x+2y)+\mathsf{n}(\mathsf{n}(x)+y)+z)+ & \\
\qquad +\,\mathsf{n}(y+z))=z & \text{54, [29} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+x+2y)+\mathsf{n}(\mathsf{n}(x)+y)+ & \\
\qquad +\,\mathsf{n}(y+z)+z)+z)=\mathsf{n}(y+z) & \text{217, [54} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(x)+y)+x+2y)+\mathsf{n}(\mathsf{n}(x)+y)+ & \\
\qquad +\,\mathsf{n}(y+z)+z)+z+u)+\mathsf{n}(\mathsf{n}(y+z)+u))=u & \text{674, [217} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+\mathsf{n}(3x))+\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+5x))= & \\
\quad =\mathsf{n}(\mathsf{n}(3x)+x) & \text{6736, [10} \to \text{674]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+5x)=\mathsf{n}(3x) & \text{8855, [6736} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+\mathsf{n}(3x)+2x))=\mathsf{n}(\mathsf{n}(3x)+x)+2x & \text{8865, [8855} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+\mathsf{n}(3x))=x & \text{8866, [8855} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(\mathsf{n}(\mathsf{n}(3x)+x)+\mathsf{n}(3x)+y)+\mathsf{n}(x+y))=y & \text{8870, [8866} \to \text{7]}\\
\mathsf{n}(\mathsf{n}(3x)+x)+2x=2x & \text{8871, [8865]}
\end{array}
$$

Figure 12.1.: Automatic Proof of Main Lemma

12.5. Equational Prover EQP

In this section we report on the equational prover EQP; EQP is restricted to equational logic and performs AC unification and matching. It is based on basic superposition, that is, paramodulation into substitution parts of terms are forbidden. The heuristics is incomplete.

AC unifiers are found by finding a *basis* of a linear Diophantine equation. The complete set of unifiers is given as linear combinations of (members of) the basis. In contrast to standard unification, AC unification is not unique. Instead of a single most general unifier, there are finitely many most general unifiers. In particular in an automatic search a huge number of AC unifiers has to be generated and tested. In order to cut down the number of these unifiers the *super*-0 *strategy* is employed in EQP. For this one defines a subset of *potential unifier* such that unification conditions except unification of subterms are fulfilled. Then the super-0 strategy strategy restricts the number of AC unifiers by ignoring supersets if a potential unifier is found. This strategy is incomplete and causes the incompleteness of EQP. For AC matching a dedicated algorithm based on backtracking is used.

In addition various selection strategies are employed. The *weight* of a pair of equations be the sum of the size of its members. The *age* of a pair is the sum of the ages of its members. A *pairing algorithm* used to select the next equation: either the lightest or the oldest pair (not yet selected) is chosen. the *pair selection ratio* specifies the ratio $\frac{lightest}{oldest}$. The default setting is $\frac{1}{0}$, that is, always the lightest pair is chosen.

In a nutshell the use of EQP can be summarised as follows:

– Successful attack took place over the course of five weeks.

– The following search parameters were varied:

   (i) limit on the size of retained equations,

   (ii) with or without super-0 heuristics,

   (iii) with or without basic restriction,

   (iv) pair selection ratio $\frac{1}{0}$ or $\frac{1}{1}$.

– Subsequent experiments searched for shorter proofs.

– Yielded direct proof without the use of Winker's lemmas

Problems

Problem 12.1. Update the formula set $\mathcal{H}$ so that the additional requirement that keys are different from nonces is properly expressed.

Hint: Introduce a new unary predicate constant Key, update formula 7 correspondingly, and add formulas that express that nonces and keys are different.

Problem 12.2. Consider the formalisation in Problem 12.1 and let $\mathcal{H}'$ denote the corresponding set of formulas. Show that the following consequence

$$\mathcal{H}' \models (\exists x\ y\ z(\mathsf{Ik}(\mathsf{key}(x,y)) \land \mathsf{Bk}(\mathsf{key}(x,z))))$$

does *not* hold.

Problem 12.3. Download the theorem prover SPASS together with the formalisation of the Neuman-Stubblebine protocol from the SPASS homepage: `http://www.spass-prover.org/`. Verify Facts 12.1 and 12.2 using SPASS and show that you can automatically disprove the consequence $\mathcal{H}' \models (\exists x\ y\ z(\mathsf{Ik}(\mathsf{key}(x,y)) \land \mathsf{Bk}(\mathsf{key}(x,z))))$ in Problem 12.2.

Bibliography

[1] Peter B. Andrews. Resolution in type theory. *J. Symb. Logic*, 36(3):414–432, 1971.

[2] Peter B. Andrews. Theorem proving via general matings. *J. ACM*, 28(2): 193–214, 1981.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[4] M. Baaz and C.G. Fermüller. Non-elementary speedups between different versions of tableaux. In *Proc. 4th TABLEAUX*, volume 918 of *LNCS*, pages 217–230, 1995.

[5] M. Baaz, U. Egly, and A. Leitsch. Normal form transformations. In *Handbook of Automated Reasoning*, pages 273–333. 2001.

[6] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2&3):173–201, 1989.

[7] L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[8] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. Elsevier, second edition, 1985.

[9] H. Barendregt and E. Barendsen. Introduction to lambda calculus. In *Aspenæs Workshop on Implementation of Functional Languages, Göteborg.* Programming Methodology Group, University of Göteborg and Chalmers University of Technology, 1988. Available at `ftp://ftp.cs.kun.nl/pub/CompMath.Found/lambda.pdf`.

[10] M. Ben-Ari. *Mathematical Logic for Computer Science.* Springer Verlag, second edition, 2001.

[11] G.S. Boolos, J.P. Burgess, and R.C. Jeffrey. *Computability and Logic.* Cambridge University Press, fifth edition, 2007.

[12] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1(1): 146–166, 1989.

[13] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *Proceedings of 18th International Conference on Computer Aided Verification*, pages 415–418, 2006.

[14] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of 4th Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[15] B. Dahn. Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem. *Journal of Algebra*, pages 526–532, 1998.

[16] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[17] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[18] M. Dummett. *Elements of Intuitionism.* Oxford Logic Guides. Oxford University Press, second edition, 2000.

[19] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Undergraduate Texts in Mathematics. Springer Verlag, second edition, 1994.

[20] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik.* Hochschul Taschenbuch. Spektrum Akademischer Verlag, fünfte edition, 2007.

[21] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[22] D. Fensel, J. Angele, and R. Studer. The knowledge acquisition and representation language KARL. *IEEE Trans. Knowl. Data Eng.*, 10(4):527–550, 1998.

[23] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Graduate Texts in Computer Science. Springer Verlag, second edition, 1996. out of print.

[24] G. Gentzen. Untersuchungen über das logische Schließen I–II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

[25] P. Gilmore. A proof method for quantification theory; its justification and realization. *IBM J. Res. Develop*, 4:28–35, 1960.

[26] S. Gulwani and A. Tiwari. Combining abstract interpreters. In *Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation*, pages 376–386. ACM, 2006.

[27] S. Hedman. *A First Course in Logic.* Number 1 in Oxford Texts in Logic. Oxford University Press, second edition, 2006.

[28] J.R. Hindley and J.P. Seldin. *Lambda-Calculus and Combinators: An Introduction.* Cambridge University Press, second edition, 2008.

[29] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, third edition, 2006.

[30] T. Hwang, N.-Y. Lee, C.-M. Li, M.-Y. Ko, and Y.-H. Chen. Two attacks on neuman-stubblebine authentication protocols. *Inf. Process. Lett.*, 53(2): 103–107, 1995.

[31] J.-P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In *Proc. 3rd IFIP Working Conference on Formal Description of Programming Concepts*, pages 223–244, 1986.

[32] R. Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2): 9–15, 2000.

[33] D. Kroening and O. Strichman. *Decision Procedures – An Algorithmic Point of View.* Springer Verlag, 2008.

[34] D. Lankford. Canonical inference. Technical Report ATP-32, University of Austin, Dept. of Mathematics and Computer Science, 1975.

[35] J.G. Larrecq and I. Makie. *Proof Theory and Automated Deduction.* Number 6 in Applied Logic Series. Kluwer Academic Publishers, first edition, 2001.

[36] A. Leitsch. *The Resolution Calculus.* EATCS Texts in Theoretical Computer Science. Springer Verlag, first edition, 1997.

[37] W. McCune. Solution of the robbins problem. *J. Autom. Reasoning*, 19 (3):263–276, 1997.

[38] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[39] A. Nerode and R.A. Shore. *Logic for Applications.* Graduate Texts in Computer Science. Springer Verlag, second edition, 1997.

[40] B.C. Neuman and S.G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.

[41] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* LNCS. Springer Verlag, first edition, 2002. An updated version of this tutorial is available online at http://www4.in.tum.de/~nipkow/LNCS2283/.

[42] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In *Handbook of Automated Reasoning*, pages 335–367. 2001.

[43] E. Ohlebusch. *Advanced topics in term rewriting.* Springer, 2002.

[44] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[45] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Dover Pubn Inc, 1965. 2006 reprint of Prawitz's Phd Thesis.

[46] J.A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[47] C. Rungg. Minesweeper, 2008. Bachelor Thesis.

[48] R.M. Smullyan. *First-Order Logic*. Dover Press, New York, 1994.

[49] G. Stålmarck. Normalization theorems for full first order classical natural deduction. *J. Symb. Logic*, 56(1):129–149, 1991.

[50] C. Sternagel. *Functional Programming*. Institute for Computer Science, 2009. Available at http://cl-informatik.uibk.ac.at/teaching/ws09/fp/material/fpln.pdf.

[51] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.

[52] W. Thomas. Logic for computer science: The engineering challenge. In *Informatics - 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 257–267, 2001.

[53] T. Vetterlein and K-P. Adlassnig. The medical expert system Cadiag-2, and the limits of reformulation by means of formal logics. In *Proceedings of eHealth 2009 - Health Informatics meets eHealth*, pages 123 – 128, 2009.

[54] J. von Plato. Gentzen's proof of normalization for natural deduction. *Bulletin of Symbolic Logic*, 14(2):240–257, 2008.

[55] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proceedings of the 16th International Conference on Automated Deduction*, volume 1632 of *LNCS*, pages 314–328, 1999.