Universität Innsbruck Winter Term 2011/2012

Lecture Notes

Logic (MSc)

Notes for the Lecture in the Winter Term 2011/2012

Georg Moser

Winter 2011

This document has been produced with the help of KOMA-Script and LaTeX. For the intensive guidance on LaTeX, I'd like to express my sincere thanks to Christian Sternagel.

Contents

Preface

This course on logic is aimed at students in the Master of Science program of Computer Science at the University of Innsbruck. In the course the following topics will be discussed:

- Syntax, semantics and formal systems of propositional logic.
- Syntax, semantics and formal systems of first-order logic (including equality).
- Extensions of first-order logic like second-order logic.
- Automated reasoning with equality.

In addition to the lecture homework assignments will be provided that will be discussed during the time of the lecture.[1] 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 will be done 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[2] or by text books covering this topic (see for example [20, 2, 14]).

Similar material as is covered in these lecture notes can be found in the following text books (in the order of importance): [3, 8, 18, 22]. For additional references see [7, 11, 1, 13].

[1] Participation in this discussion is not a requirement in the technical sense, but strongly recommended.
[2] See `http://cl-informatik.uibk.ac.at/teaching/ws10/lics` for the online information on the course "Logic in Computer Science" as offered in winter 2010.

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:

assume	$\forall x \forall y((\mathsf{R}_1(x,y) \vee \mathsf{R}_2(x,y)) \rightarrow \mathsf{R}_3(x,y))$
assume	$\forall x \forall y \forall z((\mathsf{R}_3(x,y) \wedge \mathsf{R}_3(y,z)) \rightarrow \mathsf{R}_3(x,z))$
assume	$\mathsf{R}_1(\mathsf{c}_1,\mathsf{c}_2) \wedge \mathsf{R}_2(\mathsf{c}_2,\mathsf{c}_3)$
thus	$\mathsf{R}_3(\mathsf{c}_1,\mathsf{c}_3)$.

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

[1] also known as

argument. Using a bit more formalism (again see Chapter 3 for details) we can write the generalised argument as follows.

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

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). In Chapter 8 we see that 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 [26]). 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]

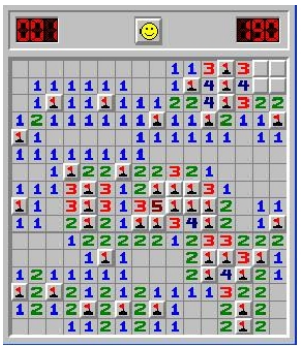

Figure 1.1: A Minesweeper Configuration

Richard Kaye has shown in [16] 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

[2] http://en.wikipedia.org/wiki/Minesweeper_(computer_game).
[3] http://minisat.se/

has to be guessed). Such a Minesweeper solver has been implemented by Christoph Rungg (see [24]).

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]

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 [17]).

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 [6] 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 [12]) and based on the Nelson-Oppen method for combining decision procedures of different theories (compare [19]). 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 [12] allows the modularisation of the analysis of programs via abstract interpretations. Modularisation is possible for both stages of the analysis:

[4] See http://cl-informatik.uibk.ac.at/software/puzzles/ for a collection of logic puzzle solvers.
[5] To be precise the theories T_1, T_2 are supposed to be *convex*, *disjoint*, and *stably infinite*, see [19].

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 [4]). It is widely used in knowledge representation systems, see for example [10]. 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 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 [9]). 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.

1.4 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 [21] Clifford Neuman and Stuart Stubblebine invented a key exchange protocol.[6] 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 (compare Chapter 9). 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 [28]. 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.

We will study the use of a first-order theorem prover to show that the *Neuman-Stubblebine* key exchange protocol can be broken in more detail in Chapter 9.

1.5 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 [25]), 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.[7] 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.[8] 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

[6] http://en.wikipedia.org/wiki/Neuman-Stubblebine_protocol
[7] http://research.microsoft.com/en-us/um/cambridge/projects/terminator/
[8] http://en.wikipedia.org/wiki/Pentium_FDIV_bug

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 $\rightarrow_{\mathsf{P}}$ (with respect to P) is contained in R. A relation R that is covered by finitely many well-founded relations $U_1, \ldots, U_n$ 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 [5]).

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.

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 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. We use the following precedence: $\neg$ binds stronger than $\vee$ and $\wedge$, which in turn bind stronger than $\rightarrow$.

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 $\mathsf{v}\colon \mathsf{AT} \to \{\mathsf{T}, \mathsf{F}\}$ 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 $\mathsf{v}(A_1)$, $\ldots$, $\mathsf{v}(A_n)$ 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 [14] 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. Note that the symbol $\bot$, representing contradiction, or falsity, is not part of our language of propositional logic. Instead $\bot$ should be understood as an abbreviation for an unsatisfiable formula like $p \wedge \neg p$.

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, $\ldots$, 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.

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

Figure 2.1: Natural Deduction for Propositional Logic

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 [14].

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.

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 [14].

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 $\square$) is a clause,

(ii) literals are clauses, and

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

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

$$p = \neg\neg p \qquad \square \vee \square = \square \qquad C \vee \square = \square \vee C = C \ ,$$

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

Note that the symbol $\square$ (like $\bot$) representing contradiction, is not part of our language of propositional logic. Instead $\square$ should be understood as an abbreviation for an unsatisfiable formula like $p \wedge \neg p$.

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 [18] 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 [14].

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

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:

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

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

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

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

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 $\mathcal{C}$, $\square \notin \mathcal{C}$, 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 [18, 14].

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*

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

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 axioms 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 [27] 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) $p \wedge \neg(\neg p \to q))$

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

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

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

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

(iv) $(p \vee q) \wedge \neg p \models q$

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

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.

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 the usual *propositional connectives* and *quantifiers*:

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

(ii) 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 $t_1, \ldots, t_n$ are terms, f an n-ary function symbol, then $f(t_1, \ldots, t_n)$ 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 $t_1, \ldots t_n$ are terms, then $P(t_1, \ldots, t_n)$ 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 $(A \rightarrow B)$ 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 a 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 $a(f)$, $a(P)$, $a(=)$ we usually write $f^{\mathcal{A}}$, $P^{\mathcal{A}}$, $=^{\mathcal{A}}$, 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 [14]. 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:

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

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.

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

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 [14, 8]. Note that this is not the only possibility to define that a given interpretation $\mathcal{I}$ models a formula F. Indeed in [13, 3] different approaches are taken that are essentially equivalent. The above approach has the advantage that the satisfaction relation is defined directly for *formulas*, whereas [13, 3] 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 [13, 3].

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. □

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 $\models (F \to G) \wedge (G \to F)$. 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 $a_1, \ldots, a_n \in A$ we have

$$m(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(m(a_1), \ldots, m(a_n)) \text{ , and}$$

(iii) for any n-ary predicate constant P and all elements $a_1, \ldots, a_n \in A$ we have:

$$(a_1, \ldots, a_n) \in P^{\mathcal{A}} \iff (m(a_1), \ldots, m(a_n)) \in P^{\mathcal{B}} .$$

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 $F(x) = \exists x G$.

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

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—traditionally called 'syllogism in Felapton'—using formulas:

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

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

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: $\mathsf{colour}(a)$.
- predicates: $\mathsf{Smurf}(a)$, $\mathsf{Large}(a)$, $\mathsf{ReallySmall}(a)$, $\mathsf{Happy}(a)$, $\mathsf{Child}(a, b)$ ("a is child of b"), $\mathsf{Ancestor}(a, b)$ ("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 $\mathcal{FV}\mathsf{ar}(F)$, 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. [13, 14, 8]. 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 $n = 1$).

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

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 $c^{\mathcal{J}} := f(c^{\mathcal{I}})$ 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 either $\mathcal{G} \cup \{E\} \in S$ 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 $\neg(E \vee F) \in \mathcal{G}$, 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. This is lifted later in Chapter 8.

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 $F = P(t_1, \ldots, t_n)$, 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 7 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 can be eventually satisfied such that all constructed sets $\mathcal{G}_n$ belong to S^*. This completes the proof. □

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*
$=$	$\dfrac{}{t = t} =\!:\mathsf{i}$	$\dfrac{s = t \quad F(s)}{F(t)} =\!:\mathsf{e}$

Figure 4.1: Natural Deduction: Equality Rules

	introduction	*elimination*
$\exists$	$\dfrac{F(t)}{\exists x F(x)} \exists\!:\mathsf{i}$	$\dfrac{\exists x F(x) \qquad \boxed{\begin{array}{ll} a & F(a) \\ & \vdots \\ & G \end{array}}}{G} \exists\!:\mathsf{e}$
$\forall$	$\dfrac{\boxed{\begin{array}{l} a \\ \vdots \\ F(a) \end{array}}}{\forall x F(x)} \forall\!:\mathsf{i}$	$\dfrac{\forall x F(x)}{F(t)} \forall\!:\mathsf{e}$

Here the variable a in $\exists\!:\mathsf{e}$ and in $\forall\!:\mathsf{i}$ 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 $A_1, \ldots, A_n \vdash B$, asserts that B is derived from the assumptions $A_1, \ldots, A_n$. 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 [3].

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. □

In order to prepare for the completeness theorem, we state two lemmas, whose proof is

left to the reader (compare also [3]). 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. □

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. Prove Lemma 4.6.

Problem 4.3. Prove Lemma 4.7.

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 1. Apart from the statement of the interpolation theorem and the joint consistency theorem this chapter is optional.

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 $A :\Longleftrightarrow \exists x F(x) \wedge \exists x \neg F(x)$ 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

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

Then $\exists x(x \neq x)$ serves as interpolant: Clearly $\exists x(F(x) \wedge \neg F(x)) \to \exists x(x \neq x)$ 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 7.1 and 7.2.

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. In order to prove this we make use of the model existence theorem. For that 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.

We define a collection S of sets of sentences such that $\{A, \neg C\} \in S$ and S will fulfil the satisfaction properties, cf. Definition 4.1. Then Theorem 4.3 yields that any set of formulas of $\mathcal{L}$ in S has a model and thus $\{A, \neg C\}$ is satisfiable.

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 $(A, \neg C)$. Due to Lemma 5.1 no sentence of $\mathcal{L}^+$ can bar $(A, \neg C)$ if this assumption holds. We are ready to define the collection S. Let S be the set of all sets of sentences $\mathcal{G}$ that admit an *unbarred division*, where this means that 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 $\mathcal{G}_1, \mathcal{G}_2$. It remains to verify that S admits the satisfaction properties.

We consider the only interesting case.

– Let $\mathcal{G} \in S$. If $(E \vee F) \in \mathcal{G}$, 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 $(E \vee F) \in \mathcal{G}_1$. 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 $\mathcal{G} \models \neg E$ and $\mathcal{G} \models E \vee F$ holds. Hence $\mathcal{G} \models F$. Then it is easy to see that $(\mathcal{G}_1 \cup \{F\}, \mathcal{G}_2)$ forms an unbarred division. 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 have to 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. Then $\mathcal{G}_1 \models B_1 \vee B_2$ as $\mathcal{G}_1 \cup \{E\} \models B_1$ and $\mathcal{G}_1 \cup \{F\} \models B_2$ hold. Moreover $\mathcal{G}_2 \models \neg B_1$ and $\mathcal{G}_2 \models \neg B_2$. From which we conclude that $\mathcal{G}_2 \models \neg(B_1 \vee B_2)$. Therefore $(B_1 \vee B_2)$ bars the pair $(\mathcal{G}_1, \mathcal{G}_2)$, which is a contradiction to the assumption that $\mathcal{G} \in S$. 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 and the proof is complete. □

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 1.

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. *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 7.1 and observe that the existence of a valid implication $A \to C$ is equivalent to the statement that $A \wedge \neg C$ 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.

6

Extensions of First-Order Logic

In this chapter we consider the limits of expressivity of first-order logic (see Section 6.1) and consider a specific extension of first-order logic: second-order logic (see Section 6.2). 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.

6.1 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 $F(a, b)$ such that F holds in an interpretation with environment $\ell(a) = u$, $\ell(b) = v$ 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 6.1. 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 $G_n = \{0, \ldots, n\}$ and

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

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

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

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

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 sentences $(\neg F_n)_{n \geqslant 2}$ 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.

6.2 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: $V_0^i, V_1^i, \ldots, V_j^i, \ldots$

(iii) Function variables with i arguments: $u_0^i, u_1^i, \ldots, u_j^i, \ldots$

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

Definition 6.2. *Second-order terms* are defined like first-order terms together with the following clause:

(iv) If $t_1, \ldots, t_n$ are second-order terms, u an n-ary function variable, then $u(t_1, \ldots, t_n)$ 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, ..., are used to denote constants and function symbols, while the meta-symbols u, v, w are used to denote function variables. P, Q, R, ..., vary through predicate symbols or predicate variables. Individual variables are denoted as x, y, z, ..., and predicate variables are denoted by V,X,Y,Z, etc.

Definition 6.3. *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.

Remark. We could employ different notions for free and bound second order variables, but this gets cumbersome and would constitute a distraction here. Moreover it is easy to see that function variables are not necessary and their use can be circumvented. However, the use of function symbols simplifies the clarity of exposition.

Definition 6.4. 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 6.5. 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:

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

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

Example 6.1. Consider the following second order formula $F(x, y)$:

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

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 [3] or [8].

Theorem 6.1. *(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.*

6.3 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 [23].

Definition 6.6. 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 $X_1, \ldots, X_n$.

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 [13].

Lemma 6.1. *If F is $\exists$SO, then the F-$\mathcal{K}$ problem is in* NP.

Lemma 6.2. *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 6.1 and Lemma 6.2 we obtain the following characterisation theorem due to Fagin.

Theorem 6.2. *A sentence F (of $\mathcal{L}$) is equivalent to a sentence in $\exists$SO iff F-$\mathcal{K} \in$ NP. Moreover if F-$\mathcal{K} \in$ 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 6.1 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 6.2, F is equivalent to an $\exists$SO sentence G. Moreover it follows from the proof of Lemma 6.2 (see [13]) that the first-order part of G is universal. □

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 [23].)

Corollary 6.1. 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 6.1. 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 6.2 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. □

The next corollary to Theorem 6.2 we state without proof.

Corollary 6.2. *The following is equivalent:*

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

Problems

Problem 6.1. 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 6.2. 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.

7

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 7.3 Herbrand's theorem is proven together with some corollaries that will be used later. In Section 7.4 it is shown that equality, individual and function constants can be eliminated from formulas without affecting the satisfiability. Apart from the statement of Theorem 7.4 this section is optional.

7.1 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 7.1. 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 7.1. 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 7.1. Consider $\forall x F(x) \leftrightarrow G(a)$, which abbreviates:

$$(\forall x F(x) \rightarrow G(a)) \wedge (G(a) \rightarrow \forall x F(x)) \; .$$

One logically equivalent CNF would be

$$\forall x \exists y ((\neg F(y) \vee G(a)) \wedge (\neg G(a) \vee F(x)) \; .$$

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

$$\exists y \forall x ((\neg F(y) \vee G(a)) \wedge (\neg G(a) \vee F(x)) \; ,$$

is also a CNF of F.

Theorem 7.1. *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 $\rightarrow$ in F, employing the equivalence $(E \rightarrow F) \equiv (\neg E \vee F)$.

(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 7.2.

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

$$\neg \forall x F(x) \equiv \exists x \neg F(x) \qquad \neg \exists x F(x) \equiv \forall x \neg F(x) \qquad \mathsf{Q} x E(x) \odot F \equiv \mathsf{Q} x (E(x) \odot F)$$

where $\mathsf{Q} \in \{\forall, \exists\}$, $\odot \in \{\wedge, \vee\}$, 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 7.1.

□

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

7.2 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

$$\exists x_1 \cdots \exists x_n \; G(x_1, \ldots, x_n) \; ,$$

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

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

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

Definition 7.2. 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 7.3. 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

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

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

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

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

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

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 7.2. *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 7.1. 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 $a_1, \ldots, a_{i-1}$

$$\mathcal{M}^+ \models H(a_1, \ldots, a_{i-1}, f(a_1, \ldots, a_{i-1})) \,. \tag{7.1}$$

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 $a_1, \ldots, a_i$ are interpreted as $b_1, \ldots, b_{i-1}, b$, 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 (7.1) 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. □

7.3 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 7.4. 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. Instead, in preparation for Herbrand's theorem, we argue directly. Let $t_1, \ldots, t_n$ be terms. Then the formula $F(t_1, \ldots, t_n)$ is called an instance of $\forall x_1 \cdots \forall x_n F(x_1, \ldots, x_n)$. If all terms t_i $(1 \leqslant i \leqslant n)$ are ground, $F(t_1, \ldots, t_n)$ is called a *ground instance*.

Suppose that $\mathcal{I}$ models $\mathcal{G}$ and let $\forall x_1 \cdots \forall x_n F(x_1, \ldots, x_n) \in \mathcal{G}$. By definition of the satisfaction relation $\mathcal{I}$ also models every ground instance $F(t_1, \ldots, t_n)$ of $\forall x_1 \cdots x_n F(x_1, \ldots, x_n)$. Consider a Herbrand interpretation $\mathcal{J}$ (of the language of $\mathcal{G}$) that satisfies exactly the same instances $F(t_1, \ldots, t_n)$ as $\mathcal{I}$. This amounts to set $\mathcal{J}$ as the collection of all true atoms $F(t_1, \ldots, t_n)$ in the interpretation $\mathcal{I}$. Then $\mathcal{J} \models \forall x_1 \cdots \forall x_n F(x_1, \ldots, x_n)$ 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 $a_1, \ldots, a_n$ denote the set of (free) variables in F. Then only the values $\ell(a_1), \ldots \ell(a_n)$ of the environment ℓ are important. Thus instead of $(\mathcal{A}, \ell) \models F$ we sometimes write:

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

Theorem 7.3. *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

$$\mathsf{Gr}(\mathcal{G}) := \{F(t_1, \ldots, t_n) \mid \forall x_1 \cdots \forall x_n F(x_1, \ldots, x_n) \in \mathcal{G}, t_1, \ldots, t_n \text{ closed terms of } \mathcal{L}\} .$$

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

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

Corollary 7.2. *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. □

We can paraphrase Herbrand's theorem (and Corollary 7.2) 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 7.3 and Corollary 7.2 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 7.3. One way to overcome this problem is to assert that any formula $F(a_1, \ldots, a_n)$ (with free variables $a_1, \ldots, a_n$) is understood to be implicitly universally quantified as follows: $\forall x_1 \cdots x_n F(x_1, \ldots, x_n)$, see for example [13]. In Chapter 8 we make use of similar ideas in the definition of (first-order) clause logic.

Corollary 7.3. *If $F(a_1, \ldots, a_n)$ is a quantifier-free formula in a language $\mathcal{L}$ with at least one constant, then $\exists x_1 \cdots \exists x_n F(x_1, \ldots, x_n)$ is valid iff there are ground terms $t_1^k, \ldots, t_n^k$, $k \in \mathbb{N}$ such that the* Herbrand disjunction $F(t_1^1, \ldots, t_n^1) \vee \cdots \vee F(t_1^k, \ldots, t_n^k)$, *is valid.*

Proof. If $\exists x_1 \cdots \exists x_n F(x_1, \ldots, x_n)$ is valid, then $\forall x_1 \cdots \forall x_n \neg F(x_1, \ldots, x_n)$ is unsatisfiable and vice versa. By Corollary 7.2 there exists a finite disjunction of formulas $F(t_1^k, \ldots, t_n^k)$ that is valid. □

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 7.2 there exists a formula F' in SNF such that $F \approx F'$. Suppose F' has the following shape:

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

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 $A_0, A_1, \ldots$ 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 nodes in T are closed, then there exists a finite sets S of ground instances:

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

for closed terms $t_1^k, \ldots, t_n^k$, $k \geqslant 1$ 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.

In Chapter 8 we see how a refined variant of this idea can be used as the basis of a very efficient automated reasoning technique for first-order logic.

7.4 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: $a = f(b_1, \ldots, b_n)$, where the indicated terms a, b_1, $\ldots$, 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 $a = f(b_1, \ldots, b_n)$. Then we replace all occurrences of this equality by a $P(b_1, \ldots, b_n, a)$, 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*:

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

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

Lemma 7.1. *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 7.1.) We make use of an additional binary relation symbol $\leftrightharpoons$ together with the following *equivalence axioms* E.

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

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

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

For any formula F let F''' denote the result of replacing the equality sign $=$ everywhere by $\leftrightharpoons$ 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 7.1.

Lemma 7.2. *F is satisfiable if and only if $F''' \wedge E \wedge C(F)$ is satisfiable.*

Lemma 7.1 and 7.2 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 7.4. *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 7.1. 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 7.2. 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 7.3. 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 $\mathcal{G}_2 = \{\mathsf{P}(\mathsf{c}), \neg \mathsf{P}(a)\}$. Show that $\mathcal{G}_2$ is satisfiable, but doesn't have a Herbrand model.

Problem 7.4. Prove Lemma 7.1.

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 7.5. Prove Lemma 7.2.

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.

8

Automated Reasoning with Equality

In this chapter we introduce the 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. Hence, we will first introduce the *resolution calculus*, a system of inference rules well-suited for automation (see Section 8.1). 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.) In Section 8.2 we extend resolution by suitably defined inference rules to overcome this (practical) restriction. The obtained calculus is called *paramodulation calculus*. In order to improve the efficiency of this calculus, we finally study a refined version, the *superposition calculus*, in Section 8.3.

Note that the here introduced automated techniques are easily powerful enough to show the validity of the semantic entailment (1.1) mentioned in Chapter 1.

8.1 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 $k_0, k_1, \ldots, k_j, \ldots$, function constants (with i arguments) $f_0^i, f_1^i, \ldots, f_j^i, \ldots$ and predicate constants (with i arguments) $R_0^i, R_1^i, \ldots, R_j^i, \ldots$ In addition to these constants we make use of variables: $x_0, x_1, \ldots, x_j, \ldots$ 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 that this is of no real concern.

Definition 8.1. 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 7.2 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 the sake of the argument, we 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 8.1. *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. □

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 8.2. 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 $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, $\tau = \{y_1 \mapsto r_1, \ldots, y_1 \mapsto r_m\}$ (denoted as $\sigma\tau$) is defined as follows:

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

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

Definition 8.3. 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 $E = u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n$ is called an *equality problem*. Here u_i, v_i denotes either terms or atomic formulas. The unifier of an equality problem $E = x_1 \stackrel{?}{=} v_1, \ldots, x_n \stackrel{?}{=} v_n$ is defined as the unifier of the set $\{u_1 = v_1, \ldots, u_n = v_n\}$. The rules in Figure 8.1 define a simple, rule based unification algorithm that acts on equality problems.

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

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

$$f(s_1, \ldots, s_n) \stackrel{?}{=} g(t_1, \ldots, t_m), E \Rightarrow \bot \quad \text{if } f \neq g$$

$$x \stackrel{?}{=} v, E \Rightarrow x \stackrel{?}{=} v, E\{x \mapsto v\} \quad \text{if } x \in Var(E),\ x \notin Var(v)$$

$$x \stackrel{?}{=} v, E \Rightarrow \bot \quad \text{if } x \neq v,\ x \in Var(v)$$

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

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

Figure 8.1: Rule Based Standard Unification

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

Theorem 8.2. *An equality problem E is unifiable iff the unification algorithm of Figure 8.1 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 $E = x_1 \stackrel{?}{=} v_1, \ldots, x_n \stackrel{?}{=} v_n$ is a solved form, then the induced substitution $\sigma_E = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ is a mgu of E. □

It is not difficult to see that the algorithm terminated, but may produce exponentially large terms. Now, we are ready to state the two *inference rules* of the resolution calculus in Figure 8.2. 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.

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

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 8.2: 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 8.2 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 8.4. Let $\mathcal{C}$ be a set of clauses. We define the *resolution operator* $\mathsf{Res}(\mathcal{C})$ as follows:

$$\mathsf{Res}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 8.2 with premises in } \mathcal{C}\} .$$

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 saturated set $\mathcal{C}$, $\square \notin \mathcal{C}$, then $\mathcal{C}$ is called *consistent*.

Theorem 8.3. *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 [18] 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. □

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

Lemma 8.1. *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 $\square \in \mathsf{Res}^*(\mathcal{C} \cup \{E\})$ 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. □

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 8.3. See [18] for a complete proof.

Lemma 8.2 (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 8.3 (Lifting Lemma)**.** *Let τ be a ground substitutions and consider the following ground factoring step:*

$$\frac{C\tau \vee A\tau \vee B\tau}{C\tau \vee A\tau} \ ,$$

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

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

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 8.1. However, we have not yet proven the model existence theorem in the full generality that is required here. Our proof didn't allow function constants in our base language. 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 8.4. *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 $t_1, \ldots, t_n$ denote elements of $\mathcal{M}$ and f an n-ary function symbol in $\mathcal{L}^+$. We define:

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

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

Proof. If F is unsatisfiable then due to Corollary 7.2 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 $\square \notin \mathsf{Res}^*(\mathcal{C}')$. Then by definition the clause set $\mathsf{Res}^*(\mathcal{C}')$ is saturated and thus consistent. By the model existence theorem in conjunction with Lemma 8.1 we conclude that $\mathcal{C}'$ is satisfiable. This is a contradiction to our assumption. Hence $\square \in \mathsf{Res}^*(\mathcal{C}')$.

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 8.2 and 8.3. □

If the above inference rules 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 8.5. Given an arbitrary well-founded and total order $\succ$ on ground atomic formulas, we define the order $\succ_{\mathsf{L}}$ on ground literals as follows:

- If $A \succ B$, then $(\neg)A \succ_{\mathsf{L}} (\neg)B$
- $\neg A \succ_{\mathsf{L}} A$.

The next lemma is immediate from the definitions.

Lemma 8.5. *If $\succ$ is well-founded and total (on ground atoms), then $\succ_{\mathsf{L}}$ is well-founded and total (on ground literals).*

Let $\succ_{\mathsf{L}}$ be a total order on ground literals according to Definition 8.5. 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 8.3 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.

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

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 8.3: Ordered Resolution Calculus

Definition 8.6. Let $\mathcal{C}$ be a set of clauses. We define the *ordered resolution operator* $\mathsf{Res}_{\mathsf{OR}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{OR}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 8.3 with premises in } \mathcal{C}\}$$

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 8.5. *Ordered resolution is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\square \in \mathsf{Res}^*_{\mathsf{OR}}(\mathcal{C})$.*

Sketch of Proof. Soundness of ordered resolution is a consequence of Theorem 8.3 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 8.5) to an order on clauses. For that one usually employs the so called multiset extension of an order (see [25] 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 7.2 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 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 8.4 but replace the application of Corollary 7.2 by the refinement described above. Finally, in order to prove completeness of ordered resolution it remains to adapt the lifting lemmas, Lemmas 8.2 and 8.3, suitably, which does not provide any problems. □

8.2 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 7.1 and and 7.2 studied in Chapter 7. 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 8.4 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.

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

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'} \ .$$

Here σ is a mgu of A and B and σ' is a mgu of s and s'.

Figure 8.4: Paramodulation Calculus

Definition 8.7. Let $\mathcal{C}$ be a set of clauses. We define the *paramodulation operator* $\mathsf{Res}_{\mathsf{P}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{P}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 8.4 with premises in } \mathcal{C}\}$$

The n^{th} (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 8.1). Finally, we are in the position to state the lemma in its full generality.

Lemma 8.6. *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 8.4. 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 $[t_1]_\sim$, $\ldots$, $[t_n]_\sim$ in $\mathcal{M}'$.

Furthermore, for any predicate constant P and for any sequence of equivalence classes $[t_1]_\sim$, $\ldots$, $[t_n]_\sim$ in $\mathcal{M}'$ we set:

$$P^{\mathcal{M}}([t_1]_\sim, \ldots, [t_n]_\sim) \iff P(t_1, \ldots, t_n) \in \mathcal{G} ,$$

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. □

Theorem 8.6. *Paramodulation is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\square \in \mathsf{Res}^*_{\mathsf{OR}}(\mathcal{C})$.*

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 a lifting lemma for paramodulation is required. This is stated similar to the Lemmas 8.2 and 8.3 above. □

It is not difficult to see that the paramodulation calculus is still inefficient due to the presence of the paramodulation rule

$$\frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma} \ ,$$

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 8.5.

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

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'} \ .$$

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 $L[t]\sigma'$ is maximal with respect to $D\sigma'$

Figure 8.5: Ordered Paramodulation Calculus

Definition 8.8. Let $\mathcal{C}$ be a set of clauses. We define the *ordered paramodulation operator* $\mathsf{Res}_{\mathsf{OP}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{OP}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 8.5 with premises in } \mathcal{C}\}$$

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.

Theorem 8.7. *Ordered paramodulation is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\square \in \mathsf{Res}^*_{\mathsf{OP}}(\mathcal{C})$.*

Proof. Soundness and completeness follow due to the combination of Theorem 2.3 and 8.6. □

8.3 Superposition Calculus

Unfortunately the *ordered paramodulation calculus* as defined in Section 8.2 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 rewriting (see for example [25]) to combine ordered resolution and paramodulation to the *superposition calculus*. The rules are the rules of Figure 8.6.

Definition 8.9. Let $\mathcal{C}$ be a set of clauses. We define the *superposition operator* $\mathsf{Res}_{\mathsf{SP}}(\mathcal{C})$ as follows:

$$\mathsf{Res}_{\mathsf{SP}}(\mathcal{C}) = \{D \mid D \text{ is conclusion of an inference in Figure 8.6 with premises in } \mathcal{C}\}$$

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 8.1. Consider the following set of clauses $\mathcal{C}$:

$$\mathsf{c} \neq \mathsf{d}$$
$$\mathsf{b} = \mathsf{d}$$
$$\mathsf{a} \neq \mathsf{d} \vee \mathsf{a} = \mathsf{c}$$
$$\mathsf{a} = \mathsf{b} \vee \mathsf{a} = \mathsf{d}$$

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:

$$\mathsf{a} \neq \mathsf{d} \vee \mathsf{b} = \mathsf{c} \vee \mathsf{a} = \mathsf{d}\ .$$

We state soundness and completeness of the superposition calculus without proof. Together with the definition of a variety of further refinements the proof can be found in [1].

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

$$\frac{C \vee s = t \quad D \vee \neg A[s']}{(C \vee D \vee \neg A[t])\sigma} \qquad \frac{C \vee s = t \quad D \vee A[s']}{(C \vee D \vee A[t])\sigma}$$

$$\frac{C \vee s = t \quad D \vee u[s'] \neq v}{(C \vee D \vee u[t] \neq v)\sigma} \qquad \frac{C \vee s = t \quad D \vee u[s'] = v}{(C \vee D \vee u[t] = v)\sigma}$$

$$\frac{C \vee s \neq t}{C\sigma} \qquad \frac{C \vee u = v \vee s = t}{(C \vee v \neq t \vee u = t)\sigma}$$

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 8.3.
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', $t\sigma \not\succcurlyeq s\sigma$, $v\sigma \not\succcurlyeq u[s']\sigma$, $(s = t)\sigma$ 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 $(s = t)\sigma \not\succcurlyeq (u[s'] = v)\sigma$.
- For the equality resolution rule, σ is a mgu of s and t, and $(s \neq t)\sigma$ is strictly 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 $(s = t)\sigma \not\succcurlyeq (u = v)\sigma$.

Figure 8.6: Superposition Calculus

Theorem 8.8. *Superposition is sound and complete. Let F be a sentence and $\mathcal{C}$ its clause form. Then F is unsatisfiable iff $\square \in \mathsf{Res}_{\mathsf{SP}}{}^*(\mathcal{C})$.*

Problems

Problem 8.1. Consider the unification algorithm given in Figure 8.1. Show that this algorithm produced exponential large terms in the worst case.

Problem 8.2. Consider the (propositional) clauses:

$$C_1 \vee A \qquad C_2 \vee \neg A \vee B \qquad C_3 \vee \neg B$$

(i) Give two different resolution derivations of the clause $C_1 \vee C_2 \vee C_3$.

(ii) Can this behaviour be avoided by the use of ordered resolution?

Problem 8.3. 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 8.4. Formulate and prove the lifting lemmas for ordered resolution.

Problem 8.5. 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 8.6. Formulate and prove the lifting lemmas for paramodulation and complete the proof of Theorem 8.6.

9

Issues of Security

The Neuman-Stubblebinde key exchange protocol (see [21]) 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 9.1 we describe the protocol and indicate how it should work. In Section 9.2 we mention a possible attack, which makes the protocol erroneous. Further, we indicate how this attack can be prevented. Finally, in Section 9.3 we show how the protocol can be formalised in first-order logic.

9.1 Neuman-Stubblebine Key Exchange Protocol

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.

- $\mathsf{N_b}$ 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 $\mathsf{E}_{\mathsf{K_{ab}}}(\mathsf{N_b})$.

9.2 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 $\mathsf{I}(\mathsf{A})$ means that the intruder impersonates Alice, while $\mathsf{I}(\mathsf{T})$ means that the intruder plays the role of the server.

The intruder only needs to send the message $\mathsf{E}_{\mathsf{K_{bt}}}(A, \mathsf{N_a}, \mathsf{Time})$ 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 [15]) who already described a solution to this attack. It suffices to distinguish nonces and keys, so that they cannot be confused.

9.3 Formalisation in First-Order

Following [28] 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:

$$\mathsf{Ak} \quad \mathsf{Bk} \quad \mathsf{Tk} \quad \mathsf{P} \quad \mathsf{M} \quad \mathsf{Fresh} \quad \mathsf{Nonce} \quad \mathsf{Store_a} \quad Sb\,.$$

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.

9.3.1 $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{A}, \mathsf{N_a}$

The first message of Alice is represented by the following set of formulas

$1\colon \mathsf{Ak}(\mathsf{key}(\mathsf{at}, \mathsf{t}))$

$2\colon \mathsf{P}(\mathsf{a})$

$3\colon \mathsf{M}(\mathsf{sent}(\mathsf{a}, \mathsf{b}, \mathsf{pair}(\mathsf{a}, \mathsf{na}))) \wedge \mathsf{Store_a}(\mathsf{pair}(\mathsf{b}, \mathsf{na}))$

9.3.2 $\mathsf{B} \longrightarrow \mathsf{T}\colon \mathsf{B}, \mathsf{E}_{\mathsf{K_{bt}}}(\mathsf{A}, \mathsf{N_a}, \mathsf{Time}), \mathsf{N_b}$

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.

$4\colon \mathsf{Bk}(\mathsf{key}(\mathsf{bt}, \mathsf{t}))$

$5\colon \mathsf{P}(\mathsf{b})$

$6\colon \mathsf{Fresh}(\mathsf{na})$

$$7\colon \forall x_{\mathsf{a}} \forall x_{\mathsf{na}}\, (\mathsf{M}(\mathsf{sent}(x_{\mathsf{a}}, \mathsf{b}, \mathsf{pair}(x_{\mathsf{a}}, x_{\mathsf{na}}))) \wedge \mathsf{Fresh}(x_{\mathsf{na}}) \rightarrow$$
$$\rightarrow \mathsf{Store_b}(\mathsf{pair}(x_{\mathsf{a}}, x_{\mathsf{na}})) \wedge \mathsf{M}(\mathsf{sent}(\mathsf{b}, \mathsf{t},$$
$$\mathsf{triple}(\mathsf{b}, \mathsf{nb}(x_{\mathsf{na}}), \mathsf{encr}(\mathsf{triple}(x_{\mathsf{a}}, x_{\mathsf{na}}, \mathsf{tb}(x_{\mathsf{na}})), \mathsf{bt})))))$$

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.

9.3.3 $\mathsf{T} \longrightarrow \mathsf{A}\colon \mathsf{E}_{\mathsf{K}_{\mathsf{at}}}(\mathsf{B}, \mathsf{N}_{\mathsf{a}}, \mathsf{K}_{\mathsf{ab}}, \mathsf{Time}), \mathsf{E}_{\mathsf{K}_{\mathsf{bt}}}(\mathsf{A}, \mathsf{K}_{\mathsf{ab}}, \mathsf{Time}), \mathsf{N}_{\mathsf{b}}$

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.

$8\colon \mathsf{Tk}(\mathsf{key}(\mathsf{at}, \mathsf{a})) \wedge \mathsf{Tk}(\mathsf{key}(\mathsf{bt}, \mathsf{b}))$

$9\colon \mathsf{P}(\mathsf{t})$

$10\colon \forall x_{\mathsf{b}} \forall x_{\mathsf{nb}} \forall x_{\mathsf{a}} \forall x_{\mathsf{na}} \forall x_{\mathsf{time}} \forall x_{\mathsf{bt}} \forall x_{\mathsf{at}}$

$(\mathsf{M}(\mathsf{sent}(x_{\mathsf{b}}, \mathsf{t}, \mathsf{triple}(x_{\mathsf{b}}, x_{\mathsf{nb}}, \mathsf{encr}(\mathsf{triple}(x_{\mathsf{a}}, x_{\mathsf{na}}, x_{\mathsf{time}}), x_{\mathsf{bt}})))) \wedge \mathsf{Tk}(\mathsf{key}(x_{\mathsf{bt}}, x_{\mathsf{b}})) \wedge$

$\wedge \mathsf{Tk}(\mathsf{key}(x_{\mathsf{at}}, x_{\mathsf{a}})) \wedge \mathsf{Nonce}(x_{\mathsf{na}}) \rightarrow \mathsf{M}(\mathsf{sent}(\mathsf{t}, x_{\mathsf{a}},$

$\mathsf{triple}(\mathsf{encr}(\mathsf{quadr}(x_{\mathsf{b}}, x_{\mathsf{na}}, \mathsf{kt}(x_{\mathsf{na}}), x_{\mathsf{time}}), x_{\mathsf{at}}),$

$\mathsf{encr}(\mathsf{triple}(x_{\mathsf{a}}, \mathsf{kt}(x_{\mathsf{na}}), x_{\mathsf{time}}), x_{\mathsf{bt}}), x_{\mathsf{nb}}))))$

$11\colon \mathsf{Nonce}(\mathsf{na})$

$12\colon \forall x \neg \mathsf{Nonce}(\mathsf{kt}(x))$

$13\colon \forall x\, (\mathsf{Nonce}(\mathsf{tb}(x)) \wedge \mathsf{Nonce}(\mathsf{nb}(x)))$

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 strengthend. 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.

9.3.4 $\mathsf{A} \longrightarrow \mathsf{B}\colon \mathsf{E}_{\mathsf{K}_{\mathsf{bt}}}(\mathsf{A}, \mathsf{K}_{\mathsf{ab}}, \mathsf{Time}), \mathsf{E}_{\mathsf{K}_{\mathsf{ab}}}(\mathsf{N}_{\mathsf{b}})$

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.

$14\colon \forall x_{\mathsf{nb}} \forall x_{\mathsf{k}} \forall x_{\mathsf{m}} \forall x_{\mathsf{b}} \forall x_{\mathsf{na}} \forall x_{\mathsf{time}}$

$((\mathsf{M}(\mathsf{sent}(\mathsf{t}, \mathsf{a}, \mathsf{triple}(\mathsf{encr}(\mathsf{quadr}(x_{\mathsf{b}}, x_{\mathsf{na}}, x_{\mathsf{k}}, x_{\mathsf{time}}), \mathsf{at}), x_{\mathsf{m}}, x_{\mathsf{nb}}))) \wedge$

$\wedge \mathsf{Store}_{\mathsf{a}}(\mathsf{pair}(x_{\mathsf{b}}, x_{\mathsf{na}}))) \rightarrow \mathsf{M}(\mathsf{sent}(\mathsf{a}, x_{\mathsf{b}}, \mathsf{pair}(x_{\mathsf{m}}, \mathsf{encr}(x_{\mathsf{nb}}, x_{\mathsf{k}})))) \wedge \mathsf{Ak}(\mathsf{key}(x_{\mathsf{k}}, x_{\mathsf{b}})))$

$15\colon \forall x_{\mathsf{time}} \forall x_{\mathsf{k}} \forall x_{\mathsf{nb}} \forall x_{\mathsf{a}} \forall x_{\mathsf{na}}$

$((\mathsf{M}(\mathsf{sent}(x_{\mathsf{a}}, \mathsf{b}, \mathsf{pair}(\mathsf{encr}(\mathsf{triple}(x_{\mathsf{a}}, x_{\mathsf{k}}, \mathsf{tb}(x_{\mathsf{na}})), \mathsf{bt}), \mathsf{encr}(\mathsf{nb}(x_{\mathsf{na}}), x_{\mathsf{k}})))) \wedge$

$\wedge \mathsf{Store}_{\mathsf{b}}(\mathsf{pair}(x_{\mathsf{a}}, x_{\mathsf{na}}))) \rightarrow \mathsf{Bk}(\mathsf{key}(x_{\mathsf{k}}, x_{\mathsf{a}})))$

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 9.1. 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 a variant of ordered resolution 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 9.2. 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: $\forall x_\mathsf{a} \forall x_\mathsf{b} \forall x_\mathsf{m} \, (\mathsf{M}(\mathsf{sent}(x_\mathsf{a}, x_\mathsf{b}, x_\mathsf{m})) \rightarrow \mathsf{Im}(x_\mathsf{m}))$

17: $\forall u \forall v \, (\mathsf{Im}(\mathsf{pair}(u, v)) \rightarrow \mathsf{Im}(u) \wedge \mathsf{Im}(v))$

18: $\forall u \forall v \forall w \, (\mathsf{Im}(\mathsf{triple}(u, v, w)) \rightarrow (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w)))$

19: $\forall u \forall \, v \forall w \forall z \, (\mathsf{Im}(\mathsf{quadr}(u, v, w, z)) \rightarrow (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w) \wedge \mathsf{Im}(z)))$

20: $\forall u \forall v \, (\mathsf{Im}(u) \wedge \mathsf{Im}(v) \rightarrow \mathsf{Im}(\mathsf{pair}(u, v)))$

21: $\forall u \forall v \forall w \, ((\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w)) \rightarrow \mathsf{Im}(\mathsf{triple}(u, v, w)))$

22: $\forall u \forall v \forall w \forall z \, ((\mathsf{Im}(u) \wedge \mathsf{Im}(v) \wedge \mathsf{Im}(w) \wedge \mathsf{Im}(z)) \rightarrow \mathsf{Im}(\mathsf{quadr}(u, v, w, z)))$

23: $\forall x \forall y \forall u \, ((\mathsf{P}(x) \wedge \mathsf{P}(y) \wedge \mathsf{Im}(u)) \rightarrow \mathsf{M}(\mathsf{sent}(x, y, u)))$

24: $\forall u \forall v \, ((\mathsf{Im}(u) \wedge \mathsf{P}(v)) \rightarrow \mathsf{Ik}(\mathsf{key}(u, v)))$

25: $\forall u \forall v \forall w \, ((\mathsf{Im}(u) \wedge \mathsf{Ik}(\mathsf{key}(v, w)) \wedge \mathsf{P}(w)) \rightarrow \mathsf{Im}(\mathsf{encr}(u, v)))$

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 9.2. The formula $\exists x(\mathsf{Ik}(\mathsf{key}(x, \mathsf{b})) \wedge \mathsf{Bk}(\mathsf{key}(x, \mathsf{a})))$ is derivable from $\mathcal{H}$ fully automatically by a variant of ordered resolution in less than a second.

This fact expresses that the attack reported in [15] 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 [28]), then one can formally and automatically verify that the (updated) protocol is safe.

Problems

Problem 9.1. 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 9.2. Consider the formalisation in Problem 9.1 and let $\mathcal{H}'$ denote the corresponding set of formulas. Show that the following consequence

$$\mathcal{H}' \models (\exists x\ y\ z(\mathsf{Ik}(\mathsf{key}(x, y)) \wedge \mathsf{Bk}(\mathsf{key}(x, z))))$$

does *not* hold.

Problem 9.3. 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 9.1 and 9.2 using SPASS and show that you can automatically disprove the consequence $\mathcal{H}' \models (\exists x\ y\ z(\mathsf{Ik}(\mathsf{key}(x, y)) \wedge \mathsf{Bk}(\mathsf{key}(x, z))))$ in Problem 9.2.

Bibliography

[1] L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[2] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer Verlag, second edition, 2001.

[3] G.S. Boolos, J.P. Burgess, and R.C. Jeffrey. *Computability and Logic*. Cambridge University Press, fifth edition, 2007.

[4] 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.

[5] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *Proceedings of 18th International Conference on Computer Aided Verification*, pages 415–418, 2006.

[6] 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.

[7] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer Verlag, second edition, 1994.

[8] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Hochschul Taschenbuch. Spektrum Akademischer Verlag, fünfte edition, 2007.

[9] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[10] D. Fensel, J. Angele, and R. Studer. The knowledge acquisition and representation language KARL. *IEEE Trans. Knowl. Data Eng.*, 10(4):527–550, 1998.

[11] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer Verlag, second edition, 1996. out of print.

[12] 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.

[13] S. Hedman. *A First Course in Logic*. Number 1 in Oxford Texts in Logic. Oxford University Press, second edition, 2006.

[14] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, third edition, 2006.

[15] 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.

[16] R. Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.

[17] D. Kroening and O. Strichman. *Decision Procedures – An Algorithmic Point of View*. Springer Verlag, 2008.

[18] A. Leitsch. *The Resolution Calculus*. EATCS Texts in Theoretical Computer Science. Springer Verlag, first edition, 1997.

[19] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[20] A. Nerode and R.A. Shore. *Logic for Applications*. Graduate Texts in Computer Science. Springer Verlag, second edition, 1997.

[21] B.C. Neuman and S.G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.

[22] 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/.

[23] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[24] C. Rungg. Minesweeper, 2008. Bachelor Thesis.

[25] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.

[26] 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.

[27] 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.

[28] 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.