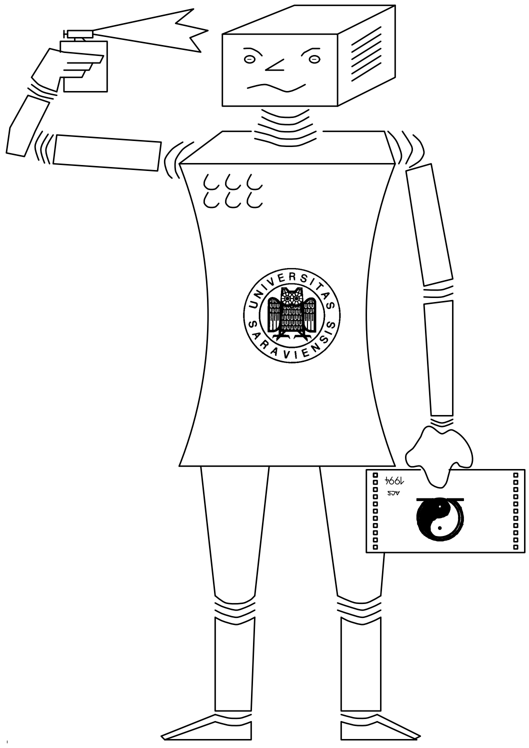
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



On the Representation of Mathematical Concepts and their Translation into First-Order Logic

Manfred Kerber

Published as: PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany

On the Representation of Mathematical Concepts and their Translation into First-Order Logic

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

von
Dipl.-Math. Manfred Kerber

Dekan des Fachbereichs:	Prof. Dr. Jürgen Avenhaus
Vorsitzender der Promotionskommission:	Prof. Dr. Ewald von Puttkamer
Berichterstatter:	Prof. Dr. Jörg H. Siekmann
	Prof. Dr. Klaus Madlener

Tag der wissenschaftlichen Aussprache: 24. Februar 1992

Kaiserslautern 1992
D 386

Contents

Abstract

To prove difficult theorems in a mathematical field requires substantial knowledge of that field. In this thesis a frame-based knowledge representation formalism including higher-order sorted logic is presented, which supports a conceptual representation and to a large extent guarantees the consistency of the built-up knowledge bases. In order to operationalize this knowledge, for instance, in an automated theorem proving system, a class of sound morphisms from higher-order into first-order logic is given, in addition a sound and complete translation is presented. The translations are bijective and hence compatible with a later proof presentation.

In order to prove certain theorems the comprehension axioms are necessary, (but difficult to handle in an automated system); such theorems are called truly higher-order. Many apparently higher-order theorems (i.e. theorems that are stated in higher-order syntax) however are essentially first-order in the sense that they can be proved without the comprehension axioms: for proving these theorems the translation technique as presented in this thesis is well-suited.

Chapter 1

Introduction

> Alles, was im bisherigen Sinne die Mathematik ausmacht, wird streng formalisiert, so daß die eigentliche Mathematik oder die Mathematik in engerem Sinne zu einem Bestande an Formeln wird.
>
> *David Hilbert*

This work is about the representation of mathematical factual knowledge (by applying representation techniques of artificial intelligence) and the operationalization of this knowledge for automated theorem proving (by translating it into first-order logic). The starting point was Jörg H. Siekmann's idea of proving a whole mathematical text book by the Markgraf Karl Refutation Procedure (MKRP), a first-order resolution-based automated theorem prover [93, 41]. In order to prove a theorem the MKRP system requires that the *axioms* (that are certain hypotheses) and a *theorem* are entered in an order-sorted *first*-order language. The system tries to show that the conjunction of these axioms entails the theorem. In 1984–1986 about a third of the theorems in a textbook on semi-groups and automata [40] was proved by the system. So far theorem provers have been used primarily to prove single theorems, that is, the formalization is input for one single problem only. This problem is solved and then everything is forgotten before the next problem is attempted. If we try to prove several theorems in the same domain, we expect a certain coherence of the representation in the problems. A primary goal of this research – in addition to testing the strengths and drawbacks of MKRP – has been to get an intuition for the new problems that may occur when proving a large set of interdependent problems. In this sense proving the textbook [40] was an important testbed and source of ideas and the following is a listing of the problems that occurred most often:

– The representation of the mathematical concepts in the sorted first-order input language of MKRP was *ad hoc.* Since the constructs in [40] are mostly higher-order, they had to be translated into the MKRP first-order input language. This was done for each problem anew. Moreover this representation was neither based on an axiomatic set theory nor on a systematic translation of higher-order constructs into first-order logic. So it was not always obvious what the MKRP-proved theorems had to do with the textbook theorems and hence what was proved, even in the case of a positive session.

- Definitions and already proved theorems had to be duplicated, because the theorem prover can only prove first-order tautologies. This means, the definitions and theorems that are used as preconditions for the actual theorem have to be entered again. Not only is this rather boring, it is also a source of error. The user is responsible for the correctness of the preconditions. He might use (slightly) different formulations in different contexts, with the result that the correctness of the whole procedure is no longer ensured. This faulty procedure is known in logic as *ignoratio elenchi*. Furthermore the user can use lemmata that have not been proved. This is comparable to the procedure of *ignotum per ignotum*. Discipline may be helpful, but as practice shows, system support is imperative.

- The preconditions had to be selected by the user. But even if this important job was done optimally, that is the user entered only a *minimal* set of preconditions, normally the proofs were too difficult to be found without any lemmata. The user had to split the proof into different parts, prove certain lemmata with MKRP and enter them later on as preconditions for the actual theorem. The system did not provide any support for such a procedure.

- There was no way of explicitly reasoning *about* proofs. All structuring and every proof plan is hand-crafted and hence only subconsciously in the head of the user. So it requires a lot of practice in proving theorems and automation is not possible at this level.

These and other problems indicate that considerable additional support from a system is necessary in order to have a usable tool for developing and finding proofs.

Although automated theorem provers have solved even difficult mathematical problems (see e.g. [132, Chapter 9]), these problems are relatively non-standard and not of general interest. In particular the difficulty of the theorems, which can be solved fully automatically, is limited. We believe the future is with systems that strongly interact with the user. The user can guide the system as he wants, but the system can show essential parts by its own. Neither proof checkers, where the user has to input the proof at calculus level, nor automated theorem provers, which have limited abilities, will become general accepted tools. Instead a proof development system like Automath or Nuprl should be combined with an *automatic* theorem prover. This is the aim of a project in which the MKRP is to be extended to a new system, called Ω-MKRP [121]. In this context it is particularly important to know how to represent mathematical concepts (in an enriched higher-order language) and how to translate the formulae from higher-order into first-order logic in a sound and complete way so that a first-order theorem prover like the MKRP system can be integrated as one tool in a larger system. This work is concerned with these two questions. Unfortunately such a translation will not make *higher-order* theorem proving obsolete, because additional particularly awkward axioms are necessary in order to prove *really* higher-order theorems by first-order proof procedures. But for many problems a first-order theorem prover will show better results than a higher-order one.

Overview

In the next chapter we will give a short historical overview of the development of mathematical logic and its mechanization and we will introduce some related works. In chapter 3 we will present the logical systems that are the heart of the representation, namely unsorted higher-order logic, many-sorted first-order logic, and sorted higher-order logic. We do not use any λ-expressions. In chapter 4 we present a frame-based approach for the representation of axioms, definitions and theorems. Up to this point we are dealing with the representation of mathematical knowledge. In chapter 5 we will show how this information can be translated into first-order logic and hence can be made available for the reasoning of theorem provers based on first-order logic like the MKRP prover. In chapter 6 we give some examples of how theorems can be translated and proved with the help of the MKRP prover, and provide some intuition for whether such a translation is good or bad.

CHAPTER 2

A Short Historical Overview

> Und immer sind da Spuren, und immer ist einer dagewesen, und immer ist einer noch höher geklettert als du es je gekonnt hast, noch viel höher. Das darf dich nicht entmutigen. Klettere, steige, steige. Aber es gibt keine Spitze. Und es gibt keinen Neuschnee.
>
> *Kurt Tucholsky*

We shall now present a short overview of the history of the representation and formalization of mathematical concepts in logic and set theory, on how computer systems are used for mathematics, and on knowledge representation facilities in artificial intelligence. Of course many an important work must be omitted here.

2.1 Logic and Foundations of Mathematics

Logic has a long tradition. The Ancient Greeks formalized already parts of it by introducing calculation rules. ARISTOTLE (384–322 BC) developed the theory of syllogisms and created the first comprehensive system of logic in his Organon (especially in the "Analytica priora" and "Analytica posteriora") [5]. RAIMUNDUS LULLUS (~1233–1316) developed ideas of the mechanization of logic in the "Ars magna" (compare [21]). In "Regulae ad directionem ingenii" RENÉ DESCARTES (1596–1650) introduced a notion by which all problems should be translated into mathematical problems. These problems are transformed into a system of equations, the system into one equation, and this is then solved [39]. (A discussion of the relevance of this procedure for nowadays problem solving is given by GEORGE PÓLYA in [115].)

Taking up these works, GOTTFRIED WILHELM LEIBNIZ (1646–1716) formulated with a new accuracy the idea of a *lingua characteristica universalis* in which everything should be expressible and should be mechanized by a universal calculus, the *calculus ratiocinator* based on numbers. Thus every dispute between people should be settled by some sort of calculation ("calculemus") [81, 80]. Thereby LEIBNIZ anticipated the idea of the mechanization of human thought, a dream that has now – three hundred years later – been revived with a new twist in the field of artificial intelligence.

In "The Mathematical Analysis of Logic" (1847) GEORGE BOOLE (1815–1864) developed a first usable calculus for propositional logic [14]. In his famous "Begriffsschrift" [45] of 1879 GOTTLOB FREGE (1848–1925) restricted LEIBNIZ'S universal approach by comparing the relationship of his formal language to everyday language with the relationship of a microscope to the eye: the first is very accurate and the latter universally applicable. For the restricted area of mathematics, he wanted to build LEIBNIZ'S lingua characteristica and a calculus ratiocinator. In the "Begriffsschrift" FREGE developed the first predicate logic by analyzing quantified assertions. He clearly understood the distinction between syntax and semantics. GEORG CANTOR (1845–1918) invented (naive) set theory around the same time. In the beginning of our century antinomies were found which could be expressed in these formal systems. This lead to the development of the ramified theory of types by ALFRED NORTH WHITEHEAD (1861–1947) and BERTRAND RUSSELL (1872–1970). They used their formalism to found much of mathematics logically in "Principia Mathematica" [131]. On the other hand set theory was axiomatically founded – in order to avoid RUSSELL'S antinomies – in the set theories of ERNST ZERMELO (1871–1953) [133] and ADOLF ABRAHAM FRAENKEL (1891–1965) [44] or JOHN VON NEUMANN (1903–1957) [102], KURT GÖDEL (1906–1978) [53], and PAUL BERNAYS (1888–1977) [8, 9]. The set theory of ZERMELO was not yet based on first-order logic, but relied on higher-order constructs. Later on in long discussions about what mathematics requires it has become the general custom to restrict the underlying logic to first-order. For a discussion of the interplay between mathematical logic and set theory see [98].

DAVID HILBERT (1862–1943) articulated the idea of formalizing mathematics in what is now called HILBERT program [62]. He had the idea of axiomatizing all classical mathematics. Every theorem should be derived in finitely many steps from the axioms. For the field of geometry he had already worked out these ideas by 1899 [61]. GÖDEL'S completeness result for the first-order predicate calculus (1930) [51] was positive in this sense and seemed to amplify this direction. But then different negative results have been discovered: GÖDEL proved in 1931 [52] that every system that formalizes arithmetic cannot have a complete calculus. In 1936 ALONZO CHURCH (1903–) [32] and ALAN TURING (1912–1954) [125] proved that first-order predicate calculus is undecidable and the whole enterprise of HILBERT'S program seemed to be in jeopardy. This bad result is relativized by the theorem of Jacques Herbrand (1908–1931) [59], that for every predicate logical formula holds: it is a tautology if and only if it is possible to construct a propositional formula out of the original one, which can be shown to be a tautology. Therefore it is possible to give a enumeration of the first-order tautologies. In other words first-order logic is semi-decidable. Herbrand's theorem is of essential importance for the whole field of automated reasoning, because all proof procedures rely on it.

Another approach to mathematics, in which the axiomatic approach is strictly rejected is constructive mathematics, which is closely related to intuitionistic mathematics. The main difference from classical mathematics can be seen in their rejection of the principle of the excluded middle. Every mathematical object must be constructed explicitly. For instance an existentially quantified formula can be proved only by giving a witness for the variable. The position of constructivism was propagated by Luitzen Egbertus Jan Brouwer (1881–1966) [22, 23] and Arend Heyting (1898–1980) [60].

Until the beginning of this century there was no distinction between first-order and higher-order logic, only in 1915 Leopold Löwenheim (1878–1957) was the first to distinguish between them [91]. First-order logic is not categorical, that is, it is not possible to characterize infinite models up to isomorphy as stated in the theorem of Löwenheim-Skolem [91, 122]. In mathematics first-order logic dominates the foundations whereas in mathematical practice higher-order logic is used without a second thought. In 1940 Church formulated higher-order logic basing it on his simple theory of types and the λ-calculus [33]. Ten years later Leon Henkin (1921–) extended the semantic notion of Alfred Tarski (1901–) [124] to the concept of general models, in which he could give a complete calculus for Church's logic [57].

Andrzej Mostowski (1913–1975) could prove the isomorphy of higher-order formulations to set-theoretical formulations based on first-order logic relative to a general model semantics [100]. Herbert B. Enderton introduced in [42, p.281–289] a sound and complete translation of second-order logic into many-sorted first-order logic. Lawrence J. Henschen describes extensions of first-order theorem provers for handling arbitrary higher-order theorems [58] by introducing sorts and a special treatment for the so-called comprehension axioms. In [6] Johan van Benthem and Kees Doets give a translation for higher-order logic into first-order logic that is sound and complete relative to Henkin's semantics.

Through these developments a new rigour of proof (unfortunately along with a "loss of certainty", about the relationship between mathematics and reality; see [74]) was achieved. But these advances in logic have not had great influence on the daily practice of mathematicians when proving theorems, so that until today for (human) theorem proving one can often say "paper won't blush": The formalization of reasoning in mathematics was more than ripe for a mechanization on computers. In the next section we describe the corresponding systems briefly.

2.2 Computer Systems for Mathematics

There are different areas for the application of computers in proving theorems: there are general theorem proving systems, for example based on resolution and paramodulation which can find certain proofs automatically. In another area the computer is used to check given proofs or the user develops proofs with the aid of such a system.

Automated Theorem Proving Systems

The focus of research in the field of automated reasoning has changed considerably since the early days of the formation of artificial intelligence. At the beginning, the main interest was devoted to "theorem proving", which at that time was concerned with showing that certain logical formulae are tautologies. So it is not surprising that the **Logic Theorist** of Allen Newell, Cliff Shaw and Herbert Simon [104], which was used for proving parts of Principia Mathematica, was among the earliest AI-Systems. In 1954 Martin Davis implemented a decision procedure for Presburger arithmetic [37]. For an historical overview of these early days of automated theorem proving see [38, 120, 118].

Current deduction systems rely on different basic techniques. A very important one is *resolution*, which was invented in 1965 by John Alan Robinson [116]. In a normalization process, the whole problem, formulated in a first-order language, is transformed into a conjunction of disjunctions, where all existentially quantified variables are eliminated (Skolemization). The disjunctions consist only of atomic formulae or negated atomic formulae, so-called literals. The disjunction is called a clause and represented as a set of the contained literals. This transformation is refutation correct and complete, that is, the clause form is unsatisfiable if and only if the original problem is. Proving is done by generating new clauses (resolvents) out of two (binary resolution or factorization) or more parent clauses. If the empty clause is obtained a contradiction is found and the problem is solved.

Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle [132] have built a succession of resolution-based theorem provers that have been the strongest systems of the field and that finally resulted in the **Otter** system [96]. Otter has a strong equality handling component using demodulation and paramodulation. Integer arithmetic is built-in.

In the **Markgraf Karl Refutation Procedure (MKRP)** [93, 41], Jörg H. Siekmann and his group realized a clause graph based resolution theorem prover. Graph based resolution was invented by Robert Kowalski [77], and the idea is that literals, on which a (binary) resolution step may be executed, are connected by a link that is labeled by the corresponding unifier. The links of the resolvents can be computed by the links of the literals of the parent clauses. Clauses with a pure literal, that is, a literal without any link, can be removed from the graph because they can never have the empty clause among their descendants. The input language incorporates equality which is treated by paramodulation. The input language is order-sorted first-order logic. For examples see chapter 6.

Further developments for strengthening the deductive power of resolution-based theorem provers consist of building-in syntactic heuristics and incorporating additional features such as theory unification (for an overview of unification theory see [119]), and theory resolution [123].

Robert S. Boyer and J Strother Moore [17] built the **Computational Logic Theorem Prover** which relies on mathematical induction. Many heuristics concerned with induction schemata are incorporated into this system. In the **Inka**-system [11] the handling of existentially quantified formulae and the generalization of the induction hypothesis is automatized also.

The *matrix* method is based on the representation of the formula set in a so-called matrix. The proof is done by showing that on all paths of this matrix there are contradictory formulae. The main difference from resolution is, that no new formulae are generated, but that the problem is solved by searching the initial set of formulae (albeit some duplication of the original matrix may be necessary). Wolfgang Bibel worked out this idea in [10]; Peter B. Andrews and his group developed a higher-order theorem prover, **TPS** [2, 4] based on matrices with connections.

While the above systems are essentially general purpose theorem provers, problem specific systems for example for geometry [46], set theory [24, 108], or analysis [20] have been built and used successfully. The idea of building and using a general purpose theorem prover is nevertheless still a vital and exciting task. Today it however generally agreed upon that special, *domain specific* knowledge is necessary in order to employ such a system successfully. In [28] and [13] such proposals are described. Many approaches have been advocated to overcome problems in automated theorem proving by using heuristic knowledge of the kind described by

George Pólya [113, 114, 115] in order to teach mathematics. Allen Newell gives an extended summary of Pólya's ideas and their relationship to AI in [103]. He argues that there is still a non-trivial gap between Pólya's ideas and the representational possibilities of todays AI-formalisms, but that this gap can and should be filled one day.

Proof Checkers

The **Automath** system of Nicolaas Govert de Bruijn [25, 26, 27] was one of the first proof checking systems, in the sense that a human user could develop his proof of a theorem with the help of the system and it would then guarantee that the proofs were actually correct. It was built in the late sixties and based on natural deduction and (different) richly typed languages, some incorporating typed λ-calculus. Its philosophy is to write "books" in the system, so that the user has almost as much flexibility as if he were writing on paper except that he cannot write down anything that is false. The books are written line by line and checked by the system. The empty book is correct and whenever a new line is written it must be an admissible expression with respect to the previous ones. The system as a very general tool is not expected to find anything in itself, but to improve publication standards, for instance. In 1975 L.S. van Benthem Jutting [70] was able to completely formulate and check Edmund Landau's book on the "Grundlagen der Analysis" [79] in the Automath system. The system – like all other comparable systems so far – did not reach any broad acceptance as a working instrument for mathematicians. One reason may be that there was a *loss factor* of 10 to 20 when using Automath. The loss factor expresses what one loses in shortness when translating ordinary mathematics into Automath. It is an important observation however that this loss factor is constant over the range of a book, that is, it does not increase if one goes further into the book [27, p.603]. So there is legitimate hope of cutting this factor down (for instance to one) by some automated theorem proving techniques ("gap filling").

Whereas in Automath every proof step must be encoded by hand, the **Nuprl** system of Robert L. Constable and his group [34] actually supports the user in finding proofs, because he can write so-called tactics. Elementary tactics correspond to the application of calculus rules. They can be combined using tacticals like tac_1 `THEN` tac_2 and `REPEAT` tac. Tactics are written in a meta-language, the functional polymorphic programming language ML from the **LCF** system [54] (LCF stands for "Logic for Computable Functions"), which strongly influenced the meta-component of Nuprl. The logic is based on the constructive typed lambda calculus of Per Martin-Löf [94], in the tradition of constructive mathematics. Definitions and theorems can be stored in libraries; proofs follow the proposition-as-type paradigm [36, 68]. If a constructive proof for the inhabitation of a proposition type is given, this proof has a computational content which realizes a program. For instance, the computational content of the Nuprl-proof that every natural number has a prime factorization can be used to prime factorize an arbitrary natural number. Different proofs correspond to different algorithms. Therefore Nuprl is especially interesting for constructive mathematics and the development of functional programs.

Lawrence C. Paulson's system **Isabelle** is a generic proof development environment for different formal systems [109]. It provides a framework for developing proof checkers for different logics, initiated by the need for different logics in artificial intelligence. In a fixed higher-order meta-logic, which is a fragment of Church's typed λ-calculus, it is possible to specify an object logic and a calculus for this logic. Isabelle then behaves like a proof checker for this special logic without any further adaptation. As in Nuprl it is possible to write tactics in Isabelle.

The **Ontic** system of David A. McAllester [95] is a semi-automated verification system based on classical Zermelo-Fraenkel set theory. The language contains a rich vocabulary of types including type constructors like (`OR-TYPE` τ_1 τ_2), (`AND-TYPE` τ_1 τ_2), or (`LAMBDA` ((x τ_1)) $\Phi(x)$). Predicates are eliminated in favour of these types and type generators. The inference process is guided by a user-specified set of focus objects by which Ontic finds and applies the information in a large lemma library. The implementation of this focus method relies on semantic network style inheritance.

Meta-Reasoning

Another aspect of theorem proving is considered in the **FOL** system of Richard Weyhrauch [130] and further developments such as those of Fausto Giunchiglia [50]. They are interested in a logical meta-level representation, so that they can use an amalgamated form of reasoning, where some parts of the argumentation take place at the meta-level and some parts at the object level. Meta-level and object level must be closely related, that is, they must observe the reflection principle: If φ is a theorem of the object theory, then *THEOREM*("φ") must hold on the meta-level and vice versa. Of course one wants to reflect only certain parts of the object level at the meta-level, because then proof planning can be done at the meta-level.

Alan Bundy [29, 30] and his group are working on the integration of heuristics into proof development environments like Nuprl by the use of *proof plans*. They are using a reimplementation of the Nuprl system, called Oyster [63], and want to build proof plans out of certain tactics. Because the tactics written in ML are not particularly well suited for reasoning about them, they are extended in the **Clam**-system [30] to so-called methods. These incorporate a high-level meta-logical description of the tactics including precondition, postcondition, and the effects of a tactic. On this high level a planning system develops a plan for solving the problem. So far the examples are such that most of the proof is already found when a plan is found. Probably the planning will become more powerful, when an automated theorem prover is connected to the Clam-system instead of a proof checker.

2.3 Representation Formalisms

In all sciences there is a tradition on how to represent its technical knowledge, in mathematics as one of the oldest sciences this tradition is distinct. So mathematicians generally spend a lot of energy on a precise and elegant nomenclature, the form of the theorems, the presentation of the proofs, and last but not least the structure of the final text books. These forms of knowledge and their representation are in general not objects of the investigation. Above all they are not the subject of publications in the field of mathematics. They have, to an extent, become an object of study in Hilbert's program [62], but many questions are left open in a mere *logical* description, which for instance does not distinguish between the importance of statements, whereas mathematicians distinguish between lemmata, theorems, main theorems, corollaries, auxiliary propositions, remarks, and so on.

The formal representation of the knowledge of a scientific discipline is normally not a research object of the discipline itself, however, it is a main research topic in the field of artificial intelligence, where many different forms of representing knowledge have been developed. Frames, first introduced by Marvin Minsky [97], are very popular among others, because of their clarity and expressive power. Minsky proposed them in order to represent and structure common-sense knowledge. "Attached to each frame are several kinds of information. Some of this information is about how to use the frame. ... Some is about what one can expect to happen next." Every frame consists of different slots, which are filled by the so-called slot-fillers. An advantage of frames is the possibility to fix the slots and the types of the admissible slot fillers in advance, that is, to define the semantic *primitives* for a certain field of application in this way. These primitives are essential for the use of frames and when applying such a system they have to be made explicit; this is to establish an *epistemological level* of knowledge representation (compare Ronald J. Brachman [18]). As with KL-ONE systems [19] one may also have a network of inheritance. Frames are for example successfully applied to the representation of large knowledge bases in the CYC-project of Douglas B. Lenat [85].

For all different forms of knowledge representation there has been a discussion of their semantics. So for a long time the semantics of semantic nets has been quite obscure. In order to avoid the procedural semantics of a system, it is usual to give a translation into some logic with a clearly understood declarative semantics. Especially in a mathematical context it is essential to give a clear semantics of the represented knowledge.

Chapter 3

Logic

Die Grenzen meiner Spache bedeuten
die Grenzen meiner Welt.

Ludwig Wittgenstein,
Tractatus logico-philosophicus 5.6

First-order logic is a powerful tool for expressing and proving mathematical facts. Nevertheless higher-order expressions are often better suited for the representation of mathematics and in fact almost all mathematical text books rely on some higher-order fragments for expressiveness. This fragment can be obtained by a higher-order logic or by “implementing” parts of it in first-order logic and building it up by a strong set theory. Throughout this dissertation we will follow the first approach and use a higher-order logic, which is introduced in the current chapter.

Mathematicians use a technical language, which is relatively informal compared to the formal approaches of logic or set theory. It is however much closer to higher-order logic augmented by “naive” set theory than to first-order logic. Mathematicians know about the antinomies and avoid them, for example by omission of expressions like “$\{x|x \notin x\}$”. They also know that there is (hopefully) a clean foundation of set theory, but how this is done in detail is in general not of much interest to a working mathematician (if he is not working on one of the aspects of the foundations of mathematics of course, e.g. on logic or set theory).

Formal set theory is of course a very strong tool, especially when higher concepts are introduced by abbreviations. Beginning with the binary relation “$\in$” one can (and this has actually been achieved by N. Bourbaki [15]) define the concepts subset, intersection, union, function, relation, power-set, and so on; then all of mathematics can be built up on these constructs. The definition of a function as a left-total, right-unique relation is rather complex and remote from the construct of a function symbol that is provided originally in logic in order to express functions. If the whole of mathematics is based on set theory it will probably be better to build special theorem provers for set theory rather than relying on general purpose theorem provers (as done in [16]).

The representation of concepts using functions can be more adequately done in a higher-order language. For instance in higher-order logic it is possible to write:
$\forall +\ \ associative(+) \iff \forall x, y, z\ \ (x+y)+z \equiv x+(y+z)$.
Here $+$ is a function variable, and *associative* is a predicate constant, which expects a function term as its argument. This cannot be written immediately in first-order logic, because we quantify over $+$, so $+$ would have to be a variable. On the other hand it must be a function because of the term $x+y$, hence a function variable, and this is not allowed in first-order logic. Nevertheless this definition is expressible in first-order logic whereas many other concepts cannot be axiomatized in first-order logic at all, for example the set $\mathbb{N}$ of natural numbers is not first-order characterizable. GIUSEPPE PEANO (1858–1932) used the following induction axiom in his axiomatization of the natural numbers [110], it is second-order:
$\forall P\ \ P(0) \wedge (\forall n^{nat}\ \ P(n) \Longrightarrow P(s(n))) \Longrightarrow (\forall n^{nat}\ \ P(n))$
Another example of the inadequacy of first-order logic comes from the theorem of LÖWENHEIM-SKOLEM [91, 122]: Every (countable) axiomatization of a set which has an infinite model also has a countable model. Therefore every first-order axiomatization of the real numbers $\mathbb{R}$ has a countable model.

Using higher-order logic it is possible to characterize these models up to isomorphy. For a discussion on categoricity see [35]. Unfortunately one has to pay a price, namely that the notions of *truth* and *provability* no longer coincide [52].

In the next section we formally define a higher-order logic and then we extend this to higher-order sorted logic.

3.1 Higher-Order Logic

Our higher-order logic is based on CHURCH'S simple theory of types [33], but unlike CHURCH we use no λ-expressions and no types of the form $(o \to o)$.* Much of the notation is taken from [3]. However, we shall write the types in a different way.**

The Syntax

We will introduce type symbols, terms and formulae for the logics $\mathcal{L}^\omega$. The n-th order predicate logics $\mathcal{L}^n$ are then defined as subsets of $\mathcal{L}^\omega$. First of all we define

*These restrictions will be motivated in remarks 3.3, 3.8, and 5.19.

**For example if P is a binary predicate symbol on individuals, we write its type as $(\iota \times \iota \to o)$ instead of $(o\iota\iota)$ for better readability. Apologies to all who are familiar with CHURCH'S original notation.

types. Every expression in higher-order logic must have a type; this is one of the known devices to avoid the notorious paradoxes.

3.1 Definition (Types of $\mathcal{L}^\omega$):

1. ι is a *type* of order 0 that denotes the type of the individuals.
2. o is a *type* of order 1. It denotes the type of the truth values.
3. If $\tau_1, \ldots, \tau_m$, and σ are types not equal to o (with $m \geq 1$), then $(\tau_1 \times \cdots \times \tau_m \to \sigma)$ is a *type* of order 1 + maximum of the orders of $\tau_1, \ldots, \tau_m, \sigma$. It denotes the type of m-ary functions with arguments of type $\tau_1, \ldots, \tau_m$, respectively, and value of type σ.
4. If $\tau_1, \ldots, \tau_m$ are types not equal to o (with $m \geq 1$), then $(\tau_1 \times \cdots \times \tau_m \to o)$ is a *type* of order 1 + maximum of the orders of $\tau_1, \ldots, \tau_m$. It denotes the type of m-ary predicates with arguments of type $\tau_1, \ldots, \tau_m$, respectively.

3.2 Remark: We distinguish types like $(\tau_1 \to (\tau_2 \to \tau_3))$ from $(\tau_1 \times \tau_2 \to \tau_3)$, that is, we do not assume CURRY-equality of types, although most of the following definitions can be done assuming that these types are equal. We do that, because otherwise we have to change our translations in chapter 5 in so far as additional axioms become necessary to identify the corresponding sorts in the target logic.

3.3 Remark: By definition 3.1 we exclude – unlike CHURCH and ANDREWS [33, 3] – types such as $(o \to o)$. These types give rise to special problems in the translation into first-order logic (see also remark 5.19), because they are essentially on the level of connectives. Therefore in our restricted language it is not possible to define the connectives $\neg$ and $\wedge$ and the quantifier $\forall$, and hence they must be introduced as primitives. In the same way it is not possible to define an "IF-THEN-ELSE" construct in the restricted languages. Nevertheless we conjecture that the languages $\mathcal{L}^n$ – defined below – are adequate for expressing most mathematical facts. For instance we can have predicates like *ordered_group*$(G, +, \leq)$ of type $((\iota \to o) \times (\iota \times \iota \to \iota) \times (\iota \times \iota \to o) \to o)$. In fact in all our examples from mathematical textbooks, this was no serious restriction.

3.4 Definition (Signature of $\mathcal{L}^\omega$): The *signature* of a logic in $\mathcal{L}^\omega$ is a set $\mathcal{S} = \bigcup_\tau \mathcal{S}_\tau^{const} \cup \bigcup_\tau \mathcal{S}_\tau^{var}$ where each set $\mathcal{S}_\tau^{const}$ is a (possibly empty) set of constant symbols of type τ and $\mathcal{S}_\tau^{var}$ a countable infinite set of variable symbols of type τ. We assume that the sets $\mathcal{S}_\tau$ are all disjoint, in addition we sometimes mark the elements of a set $\mathcal{S}_\tau$ by its type τ as index. A logic in $\mathcal{L}^\omega$ is defined by its signature $\mathcal{S}$ and is denoted $\mathcal{L}^\omega(\mathcal{S})$. If there is only one signature and no danger of confusion we shall also write $\mathcal{L}^\omega$ instead of $\mathcal{L}^\omega(\mathcal{S})$.

3.5 Definition (Terms of $\mathcal{L}^\omega$):

1. Every variable or constant of a type τ is a *term* of type τ.
2. If $f_{(\tau_1 \times \cdots \times \tau_m \to \sigma)}, t_{\tau_1}, \ldots, t_{\tau_m}$ are terms of the type indicated by their subscripts, then $f_{(\tau_1 \times \cdots \times \tau_m \to \sigma)}(t_{\tau_1}, \ldots, t_{\tau_m})$ is a *term* of type σ.

3.6 Definition (Formulae of $\mathcal{L}^\omega$):

1. Every term of type o is a *formula*.
2. If φ and ψ are formulae and x is a variable of any type, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, and $(\forall x \varphi)$ are *formulae*. As long as there is no danger of confusion we often omit parentheses.

3.7 Definition (Formulae of $\mathcal{L}^\omega_\equiv$):

1. Every term of type o is a *formula*.
2. If t_1 and t_2 are terms of type τ with $\tau \neq o$ then $(t_1 \equiv_{(\tau \times \tau \to o)} t_2)$ is a *formula*.
3. If φ and ψ are formulae and x is a variable of any type, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, and $(\forall x \varphi)$ are *formulae*.

Of course we have to add $\equiv_{(\tau \times \tau \to o)}$ to $\mathcal{S}^{const}_{(\tau \times \tau \to o)}$. We use the symbol "$\equiv$" for syntactic equality and for equality in sets. It has the usual semantics. "$=$" is used as the equality symbol at the meta-level. Since the type of "$\equiv_{(\tau \times \tau \to o)}$" is normally fixed by the context, we often shorten "$\equiv_{(\tau \times \tau \to o)}$" to "$\equiv$".

3.8 Remark: As usual one can define (on a meta-level) $\vee$, $\Longrightarrow$, $\Longleftrightarrow$, and $\exists$ in terms of $\neg$, $\wedge$, and $\forall$ and use formulae containing these symbols as abbreviations. We have excluded all λ-expressions in our logics. If they are included, translations into first-order logic end up in higher-order theorem proving with undecidable unification and related problems. They are not necessary for formulating mathematics, but are important for eleminating the so-called comprehension axioms in proving theorems (compare section 5.4).

3.9 Definition ($\mathcal{L}^n$, for $n \geq 1$): $\mathcal{L}^{2n}$ ($\mathcal{L}^{2n}_\equiv$) is that subset of $\mathcal{L}^\omega$ ($\mathcal{L}^\omega_\equiv$) such that every variable and every constant is of order less then or equal to n, $\mathcal{L}^{2n-1}$ ($\mathcal{L}^{2n-1}_\equiv$) is that subset of $\mathcal{L}^{2n}$ ($\mathcal{L}^{2n}_\equiv$) such that no variable of order n is quantified.

The Semantics

In this section we introduce a set-theoretical semantics for our higher-order logic which is due to TARSKI [124] and has been extended by HENKIN [57] to general model semantics.

We use the following notation: Let $A_1, \ldots, A_m$, and B be sets, then we denote by $\mathcal{F}(A_1, \ldots, A_m; B)$ the set of all functions from $A_1 \times \cdots \times A_m$ to B.

3.10 Definition (Frame): A *frame** (set of universes) is a collection $\{\mathcal{D}_\tau\}_\tau$ of non-empty sets $\mathcal{D}_\tau$, one for each type symbol τ, such that $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_{(\tau_1 \times \cdots \times \tau_m \to \sigma)} \subseteq \mathcal{F}(\mathcal{D}_{\tau_1}, \ldots, \mathcal{D}_{\tau_m}; \mathcal{D}_\sigma)$. The members of $\mathcal{D}_o$ are called *truth values* and the members of $\mathcal{D}_\iota$ are called *individuals*.

3.11 Definition (Interpretation): An *interpretation* $\langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ of $\mathcal{L}^\omega$ consists of a frame and a function $\mathcal{J}$ that maps each constant of type τ of $\mathcal{L}^\omega$ to an element of $\mathcal{D}_\tau$.

3.12 Definition (Assignment): An *assignment* into a frame $\{\mathcal{D}_\tau\}_\tau$ is a function ξ that maps each variable of type τ of $\mathcal{L}^\omega$ to an element of $\mathcal{D}_\tau$. An assignment into an interpretation is an assignment into the frame of the interpretation. In contexts where a particular interpretation is under discussion, it will be assumed that all assignments are into that interpretation unless otherwise indicated. Given an assignment ξ, a variable x_τ, and an element $d \in \mathcal{D}_\tau$, $\xi[x_\tau \leftarrow d]$ is defined as ξ except for x_τ where it is d.

3.13 Definition (Weak Interpretation): An interpretation $\mathcal{M} = \langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ is a *weak interpretation* (weak model, general model) for $\mathcal{L}^\omega$ ($\mathcal{L}^\omega_{\equiv}$) iff there is a binary function $\mathcal{V}^\mathcal{M}$ so that for every assignment ξ and term t of type τ, $\mathcal{V}^\mathcal{M}_\xi(t) \in \mathcal{D}_\tau$ and the following conditions hold:

1. for all variables x_τ, $\mathcal{V}^\mathcal{M}_\xi(x_\tau) = \xi(x_\tau)$
2. for all constants c_τ, $\mathcal{V}^\mathcal{M}_\xi(c_\tau) = \mathcal{J}(c_\tau)$
3. for composed terms $\mathcal{V}^\mathcal{M}_\xi(f_{(\tau_1 \times \cdots \times \tau_m \to \sigma)}(t_{\tau_1}, \ldots, t_{\tau_m})) =$
$$\mathcal{V}^\mathcal{M}_\xi(f_{(\tau_1 \times \cdots \times \tau_m \to \sigma)})(\mathcal{V}^\mathcal{M}_\xi(t_{\tau_1}), \ldots, \mathcal{V}^\mathcal{M}_\xi(t_{\tau_m}))$$
4. $\mathcal{V}^\mathcal{M}_\xi(\varphi \wedge \psi) = \mathcal{V}^\mathcal{M}_\xi(\varphi) \wedge \mathcal{V}^\mathcal{M}_\xi(\psi)$

*The notion of frame has here nothing to do with the notion of frame in knowledge representation.

5. $\mathcal{V}_\xi^{\mathcal{M}}(\neg\varphi) = \neg\mathcal{V}_\xi^{\mathcal{M}}(\varphi)$

6. $\mathcal{V}_\xi^{\mathcal{M}}(\forall x_\tau \varphi) = \forall d \in \mathcal{D}_\tau \mathcal{V}_{\xi[x_\tau \leftarrow d]}^{\mathcal{M}}(\varphi)$

7. for a model of $\mathcal{L}_\equiv^\omega$ we have additionally for all terms t_1, t_2 of type $\tau \neq o$, $\mathcal{V}_\xi^{\mathcal{M}}(t_1 \equiv t_2) = (\mathcal{V}_\xi^{\mathcal{M}}(t_1) \equiv_{\mathcal{D}_\tau} \mathcal{V}_\xi^{\mathcal{M}}(t_2))^*$

3.14 Remark: We use the connectives "$\wedge$" and "$\neg$" and the quantifier "$\forall$" in a naive way at the meta-level. We do that extensively when we prove that certain morphisms between two different logics are sound. Then we have connectives in the different logics and connectives at the meta-level. Because we use the same symbol for three different ones we get nice homomorphic properties (see page 67) and avoid a mess of symbols. So the reader has to identify by the context which one is meant.

3.15 Definition (Strong Interpretation): An interpretation $\mathcal{M} = \langle \{\mathcal{D}_\tau\}_\tau, \mathcal{J} \rangle$ is a *strong interpretation* (strong model, standard model) iff it is a weak interpretation and for all occurring types $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\mathcal{D}_\tau = \mathcal{F}(\mathcal{D}_{\tau_1}, \ldots, \mathcal{D}_{\tau_m}; \mathcal{D}_\sigma)$.

3.16 Remark:

– Every strong interpretation is by definition also a weak interpretation.

– In order to fix a strong interpretation we only have to fix $\mathcal{D}_\iota$ and $\mathcal{J}$.

3.17 Remark: Of course equality at level n (for odd n) can be defined at level $n+1$ by LEIBNIZ'S *identitas indiscernibilium*, that is, by the formula
$\forall x_\tau \forall y_\tau (x \equiv_{(\tau\times\tau\to o)} y) :\Longleftrightarrow (\forall P_{(\tau\to o)}\ P(x) \iff P(y))$.
But if the underlying semantics is weak by this definition we would get a non-standard semantics for the equality predicate: Suppose we have two constants a_ι and b_ι and the equality predicate $\equiv_{(\iota\times\iota\to o)}$ is defined by LEIBNIZ'S identitas indiscernibilium in the signature of a logic. Now suppose further we have the equality $a \equiv b$ as an axiom, we could interpret the formula by $\mathcal{D}_\iota = \{1, 2\}$, $\mathcal{J}(a) = 1$, $\mathcal{J}(b) = 2$, and $\mathcal{D}_{(\iota\to o)} = \{P | P(1) = P(2) = \mathtt{T}\}$. Then we have $\mathcal{V}_\xi^{\mathcal{M}}(a \equiv b) = \mathtt{T}$ although a and b are interpreted by different elements. In other words writing $a \equiv b$ does not force a and b to be equal in all models. In the logics $\mathcal{L}_\equiv$ we want the predicate constants $\equiv$ to be interpreted strongly. That is, if we have an equality like $a \equiv b$, then a and b must be mapped onto the same element in the corresponding universe.

*Here and in the following we use for all sets A, $\equiv_A$ as equality for elements in the set A.

3.18 Definition:

1. Let φ be a formula and $\mathcal{M}$ be a weak (strong) interpretation. $\mathcal{M}$ is a weak (strong) *model* of φ if for every assignment ξ into $\mathcal{M}$, $\mathcal{V}_\xi^{\mathcal{M}}(\varphi) = \mathtt{T}$
2. A model for a set Γ of formulae is a model of each formula of Γ.
3. If every weak (strong) model of a formula set Γ is also a weak (strong) model of a formula φ, we write $\Gamma \models \varphi$ ($\Gamma \models\!\!\!\equiv \varphi$, respectively).

3.19 Theorem: *In $\mathcal{L}^1$ ($\mathcal{L}^1_{\equiv}$) for every weak model of a formula set there is a strong model with the same interpretation function $\mathcal{J}$.*

Proof: Let Γ be a set of formulae in $\mathcal{L}^1$ and $\mathcal{M} = \langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ be a weak model of Γ. If we define $\overline{\mathcal{D}}_\iota$ as $\mathcal{D}_\iota$, $\overline{\mathcal{D}}_o$ as $\mathcal{D}_o$, and for all occurring types τ with $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\overline{\mathcal{D}}_\tau := \mathcal{F}(\overline{\mathcal{D}}_{\tau_1}, \ldots, \overline{\mathcal{D}}_{\tau_m}; \overline{\mathcal{D}}_\sigma)$, then $\overline{\mathcal{M}} = \langle\{\overline{\mathcal{D}}_\tau\}_\tau, \mathcal{J}\rangle$ is a strong model of Γ. We have $\mathcal{D}_\tau \subseteq \overline{\mathcal{D}}_\tau$ for all types τ. With $\mathcal{V}^{\overline{\mathcal{M}}} = \mathcal{V}^{\mathcal{M}}$ the definition 3.13 is fulfilled automatically (because the interpretation function $\mathcal{J}$ is the same) except for 3.13.6. But 3.13.6 is satisfied, since in $\mathcal{L}^1$ we can quantify only over variables of type ι and $\overline{\mathcal{D}}_\iota = \mathcal{D}_\iota$. Therefore $\overline{\mathcal{M}}$ is a strong model of Γ. ∎

3.20 Remark: By the introduced semantics we implicitly assume that so-called *extensionality axioms* Ξ are valid, that is, that the following formulae hold:

Ξ^f For all function symbols f, g of type $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\sigma \neq o$:
$\forall f\ \ \forall g\ \ (\forall x_{\tau_1} \ldots \forall x_{\tau_m}\ \ f(x_{\tau_1}, \ldots, x_{\tau_m}) \equiv g(x_{\tau_1}, \ldots, x_{\tau_m})) \Longrightarrow f \equiv g$

Ξ^p For all predicate symbols p, q of type $\tau = (\tau_1 \times \cdots \times \tau_m \to o)$:
$\forall p\ \ \forall q\ \ (\forall x_{\tau_1} \ldots \forall x_{\tau_m}\ \ p(x_{\tau_1}, \ldots, x_{\tau_m}) \iff q(x_{\tau_1}, \ldots, x_{\tau_m})) \Longrightarrow p \equiv q$

When we introduce a calculus for our logics by translating them into first-order logic, we have to add corresponding axioms (compare definition 5.18) in order to obtain completeness.

The following example shows that the weak semantics can be very remote from the intuition mathematicians have about their models.

3.21 Example: Let P be a constant of type $(\iota \to o)$ and a be an individual constant, that is, a constant of type ι. Then the formula $\varphi := (\forall f_{(\iota\to\iota)} P(f(a))) \wedge \neg P(a)$ is unsatisfiable in the standard interpretation, because it is possible to choose the identity function for f. But we can find a weak model $\mathcal{M}$. For instance with $\mathcal{D}_\iota = \{1,2\}$, $\mathcal{D}_{(\iota\to\iota)} = \{g|g(x) = 2\}$, $\mathcal{J}(P)(1) = \mathtt{F}$, $\mathcal{J}(P)(2) = \mathtt{T}$, and $\mathcal{J}(a) = 1$, we get $\mathcal{V}_\xi^{\mathcal{M}}(\varphi) = \mathtt{T}$ for all assignments ξ.

This is a reason to restrict the possible models by requiring that certain axioms are fulfilled, namely the so-called comprehension axioms, which rule out the previous example by guaranteeing the existence of certain objects. They are the means in the approach to approximate the strong semantics by a weak one. For the axioms compare [3, p.156].

3.22 Definition (Comprehension Axioms): The *comprehension axioms* Υ are the following formulae:

Υ^f For every term t of type $\tau \neq o$ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \;\; \exists f_{(\tau_1 \times \cdots \times \tau_m \to \tau)} \;\; \forall x_1 \ldots \forall x_m \;\; (f(x_1, \ldots, x_m) \equiv t)$.

Υ^p For every formula φ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \;\; \exists p_{(\tau_1 \times \cdots \times \tau_m \to o)} \;\; \forall x_1 \ldots \forall x_m \;\; (p(x_1, \ldots, x_m) \iff \varphi)$.

3.23 Remark: In practice the user of an automated theorem prover has to decide very carefully whether such a comprehension axiom (and which, if any) is necessary for a proof. We hope that for most theorems no comprehension axiom is necessary at all. This motivates the following definition. Compare also section 5.4.

3.24 Definition: Let Γ be a formula set in $\mathcal{L}^n$ and φ be an $\mathcal{L}^n$-formula that follows semantically from Γ. We say φ is an *essentially first-order theorem* of Γ iff $\Gamma \models \varphi$. We say φ is a *truly higher-order theorem* of Γ iff $\Gamma \not\models \varphi$ but $\Gamma \cup \Upsilon \models \varphi$.

3.25 Remark: The distinction between essentially first-order and truly higher-order theorems is essential, when choosing an appropriate system for proving a theorem. Since in general it is hard to find the corresponding comprehension axiom, we conjecture that for truly higher-order theorems, a higher-order theorem prover is preferable, because it does not need the corresponding axiom. In the case of essentially first-order theorems first-order theorem proving systems should be better, because the higher-order overhead is not necessary. Of course there can be exceptions in both directions. There might be cases where it is easy to see which comprehension axiom is necessary and so this problem might be appropriate for a first-order theorem prover. On the other hand there might be cases where a higher-order theorem prover is preferable, because it is not known whether a comprehension axiom is necessary or not. Some further discussion of proving theorems by a first-order proof procedure can be found in chapter 6.

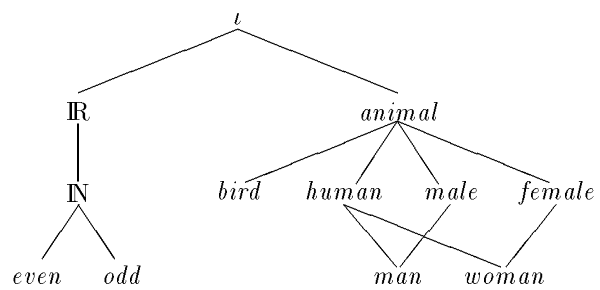
3.2 Sorted Higher-Order Logics

Now we extend the logics introduced so far to sorted higher-order logics $\mathcal{L}^n_\Sigma$. To that end, we adopt the notion of sorts for the first-order case of MANFRED SCHMIDT-SCHAUSS [117]. Many of the following definitions are analogous to those in [117]. The logics are similar to those in [43]. We will follow the concepts developed by MICHAEL KOHLHASE [75] in order to extend higher-order theorem proving to the sorted case.

The Syntax

The syntax is similar to the syntax of the logics without sorts, but now each type may have a substructure. Therefore we will introduce the notion of a sort and motivate the definition by an example.

3.26 Example: We want to structure our domain of discourse ι by subsorts $\mathbb{N}$, $\mathbb{R}$, *even*, *odd*, *human*, *male*, *female*, *animal*, *bird*, *man*, *woman* with the subsort relation $\sqsubseteq$ as denoted by the following diagram:



All these relations are introduced by *subsort declarations* of the form $\mathbb{R}\sqsubseteq\iota$.

Let us say that this are all *sorts of type* ι. Let $\dot\sqsubseteq$ be the transitive and reflexive closure of $\sqsubseteq$. Now we want to state the subsort relations between the sorts of type $(\iota \to \iota)$. First we state those relations that cannot be declared but are consequences of the corresponding relation of type ι.

We have relations $(\mathbb{R} \to \mathbb{N})\dot\sqsubseteq(\mathbb{R} \to \mathbb{R})\dot\sqsubseteq(\mathbb{R} \to \iota)$. Nota bene: A function from $(\mathbb{N} \to \mathbb{N})$ is *not* a special function from $(\mathbb{R} \to \mathbb{N})$ or vice versa. Hence we do *not* have relations like $(\mathbb{N} \to \mathbb{N})\dot\sqsubseteq(\mathbb{R} \to \mathbb{N})$ and $(\mathbb{R} \to \mathbb{N})\dot\sqsubseteq(\mathbb{N} \to \mathbb{N})$. The relation $(\kappa \to \mu)\dot\sqsubseteq(\kappa' \to \mu')$ is fulfilled iff $\kappa = \kappa'$ and $\mu\dot\sqsubseteq\mu'$.

There is no sort bigger than $(\mathbb{R} \to \iota)$ with respect to $\dot\sqsubseteq$. Hence we call this sort a *top sort* of type $(\iota \to \iota)$.

In addition to these relations it is possible to declare new subsort relations like $C\sqsubseteq(\mathbb{R} \to \mathbb{R})$, where C denotes for instance the sort of unary real continuous functions. C is called a *simple* sort, because it is not composed by an $\rightarrow$. In any subsort declaration $\kappa\sqsubseteq\mu$, κ has to be a simple sort. In the following definitions we introduce these concepts formally.

3.27 Definition (Sort): Σ is the set of *sorts*. Σ contains the type symbols ι and o. Whenever $\kappa_1, \ldots, \kappa_m$, and μ are in Σ then $(\kappa_1 \times \cdots \times \kappa_m \rightarrow \mu)$ is in Σ. We denote sorts by κ, μ, and ν.

3.28 Definition (Simple Sort): A *simple sort* is a sort that does not contain an arrow "$\rightarrow$". All other sorts are called *composed*.

3.29 Definition (Subsort Declaration): Between any sorts μ and ν it is possible to make a *subsort declaration* $\mu\sqsubseteq\nu$. The *subsort relation* $\dot{\sqsubseteq}$ is the reflexive, transitive closure of the relation $\sqsubseteq$, in addition, we have the covariance in the range sort, that is, for all composed sorts the relations: $(\kappa_1 \times \cdots \times \kappa_m \rightarrow \mu)\dot{\sqsubseteq}(\kappa'_1 \times \cdots \times \kappa'_m \rightarrow \mu')$ iff $\kappa_1 = \kappa'_1, \ldots, \kappa_m = \kappa'_m$ and $\mu\dot{\sqsubseteq}\mu'$.

3.30 Definition (Top Sort): A sort μ is called a *top sort* iff for all sorts ν with $\mu\dot{\sqsubseteq}\nu$, ν is equal to μ.

3.31 Definition (Type of a Sort): The *type of a sort* is defined inductively by:

- $\mathrm{type}(\iota) := \iota$ and $\mathrm{type}(o) := o$
- for $\mu\dot{\sqsubseteq}\nu$, $\mathrm{type}(\mu) := \mathrm{type}(\nu)$
- for $\nu = (\kappa_1 \times \cdots \times \kappa_m \rightarrow \mu)$, $\mathrm{type}(\nu) := (\mathrm{type}(\kappa_1) \times \cdots \times \mathrm{type}(\kappa_m) \rightarrow \mathrm{type}(\mu))$

3.32 Definition (Admissible Subsort Declaration): A subsort declaration is called *admissible* iff the following conditions are fulfilled:

- if $\mu\sqsubseteq\nu$ then μ is not equal to ι or o and ν is not equal to o,
- for every simple sort μ (not equal to ι or o) there is at least one subsort declaration $\mu\sqsubseteq\nu$, so that μ is subsort of a composed sort or of a type symbol (i.e. of ι or o),
- in every subsort declaration $\mu\sqsubseteq\nu$, μ must be a simple sort and $\mathrm{type}(\mu) = \mathrm{type}(\nu)$, and
- there are only finitely many subsort declarations.

3.33 Remark: In the following we assume that all subsort declarations are admissible. Therefore we have in particular that every sort has a type and that this type is unique. Furthermore we have that every top sort is either ι or o or it is a composed sort $(\kappa_1 \times \cdots \times \kappa_m \to \mu)$ with top sort μ.

3.34 Example: The next concept we want to introduce is that of a *term declaration*. For instance, we want to declare that a binary function constant $+$ is of sort $(\mathbb{N} \times \mathbb{N} \to \mathbb{N})$. In addition we might want to express that the sum of two equal numbers is even. We can do it by the term declaration: $+(x_{\mathbb{N}}, x_{\mathbb{N}}) : even$.

3.35 Remark: For the following we use the definitions of the previous section, especially the definition of types and well-typed terms.

3.36 Definition (Term Declaration): A *term declaration* is a pair (t, κ) usually denoted as $(t : \kappa)$, where t is a well-typed term that is not a variable and κ a sort symbol of the same type as t. We denote term declarations by δ. If δ is of the form $(c : \kappa)$ it is called a *constant declaration* and otherwise a *proper term declaration*. For every constant we allow at most one constant declaration. If there is no term declaration for a constant, we implicitly assume that its sort is equal to its type. Variables have fixed sorts, so they cannot be declared.

3.37 Remark: By this definition we exclude polymorphism, that is, we cannot declare $\mathbb{N} \sqsubseteq \mathbb{R}$ and then make the constant declarations $(+ : (\mathbb{N} \times \mathbb{N} \to \mathbb{N}))$ and $(+ : (\mathbb{R} \times \mathbb{R} \to \mathbb{R}))$. But it is possible to make the one constant declaration $(+ : (\mathbb{R} \times \mathbb{R} \to \mathbb{R}))$ and the term declaration $(+(x_{\mathbb{N}}, y_{\mathbb{N}}) : \mathbb{N})$, what is almost the same, with the exception that $+$ is not an instance for a function of sort $(\mathbb{N} \times \mathbb{N} \to \mathbb{N})$ (quasi-polymorphism).

3.38 Definition (Terms of $\mathcal{L}^{\omega}_{\Sigma}$):

1. Every variable x of sort κ is a *term* of sort κ.

2. Every term t with term declaration $(t : \kappa)$ and every instantiation of t is a *term* of sort κ.

3. If f is a term of sort $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$ and $t_{\kappa_1}, \ldots, t_{\kappa_m}$ are terms of the sorts indicated by their subscripts, then $f_\kappa(t_{\kappa_1}, \ldots, t_{\kappa_m})$ is a *term* of sort μ.

4. If t is a term of sort ν and $\nu \dot{\sqsubseteq} \mu$, then t is a *term* of sort μ.

3.39 Remark: The sort of a term need not be unique in general, only the top sort of a term is unique. For instance if we have in example 3.26 a function of sort $(\mathbb{R} \to \mathbb{N})$, then it has also the sorts $(\mathbb{R} \to \mathbb{R})$ and $(\mathbb{R} \to \iota)$. The top sort $(\mathbb{R} \to \iota)$ is unique.

3.40 Definition (Signature of $\mathcal{L}^{\omega}_{\Sigma}$): A *sorted signature* $\mathcal{S}_{\Sigma} = (\mathcal{S}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$ of a logic in $\mathcal{L}^{\omega}_{\Sigma}$ consist of

1. an unsorted signature $\mathcal{S}$,
2. a set Σ of sorts,
3. a function $\mathfrak{s} : \mathcal{S}^{var} \to \Sigma$, such that for every sort $\kappa \in \Sigma$, there exist countably infinitely many variables $x \in \mathcal{S}^{var}$ with $\mathfrak{s}(x) = \kappa$*,
4. a (finite) set of subsort declarations, and
5. a set of term declarations $\boldsymbol{\delta}$.

3.41 Definition (Admissible Sorted Signature): A sorted signature $\mathcal{S}_{\Sigma}$ is *admissible*, iff each subterm t_i of every well-sorted term $f(t_1, \ldots, t_m)$ is also a well-sorted term. Proper term declarations can only restrict the domain sort of a term in the sort hierarchy. In the following we will assume that all signatures are admissible.

3.42 Definition (Formulae of $\mathcal{L}^{\omega}_{\Sigma}$): Formulae are defined analogously to 3.6.

1. Every term of type o is a *formula*.
2. If φ and ψ are formulae and x is a variable of an arbitrary sort κ, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, and $(\forall x_{\kappa}\ \varphi)$ are *formulae*.

3.43 Definition (Formulae of $\mathcal{L}^{\omega}_{\Sigma,\equiv}$):

1. Every term of type o is a *formula*.
2. If t_1 and t_2 are terms of sort κ_1, κ_2 with the same top sort μ not equal to o then $(t_1 \equiv_{(\mu \times \mu \to o)} t_2)$ is a *formula*. As usual we drop the subscript of $\equiv$.

*To indicate that $\mathfrak{s}(x) = \kappa$ we often write the variable x in the form x_{κ}, we say x has sort κ. Instead of $\forall x_{\kappa}$ we often write $\forall x : \kappa$.

3. If φ and ψ are formulae and x_κ is a variable of an arbitrary sort κ, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, and $(\forall x_\kappa\ \varphi)$ are *formulae*.

3.44 Definition ($\mathcal{L}^n_\Sigma$, for $n \geq 1$): $\mathcal{L}^{2n}_\Sigma$ ($\mathcal{L}^{2n}_{\Sigma,\equiv}$) is the subset of $\mathcal{L}^\omega_\Sigma$ ($\mathcal{L}^\omega_{\Sigma,\equiv}$) such that every variable, every constant and every sort declaration is of order less or equal to n, $\mathcal{L}^{2n-1}$ ($\mathcal{L}^{2n-1}_\equiv$) is the subset of $\mathcal{L}^{2n}$ ($\mathcal{L}^{2n}_\equiv$) such that no variable of order n is quantified on.

3.45 Definition: A signature $\mathcal{S}_\Sigma$ is called *unsorted* iff Σ consists of just the type symbols, it is called *many-sorted* iff it has more sorts and the subsort relations are all of the form $\mu \dot{\sqsubseteq} \nu$ with top sort ν. Otherwise it is called *order-sorted*. We denote many-sorted logics by $\mathcal{L}^n_\Lambda$, order-sorted logics are written as $\mathcal{L}^n_\Sigma$.

3.46 Remark: An unsorted logic $\mathcal{L}^n_\Sigma(\mathcal{S}_\Sigma)$ can easily be mapped to a logic $\mathcal{L}^n(\mathcal{S})$, because in the unsorted case $\mathcal{S}_\Sigma$ has the form $(\mathcal{S}, \{\tau | \tau \text{ type}\}, x \mapsto \text{type}(x), \emptyset, \emptyset)$, that means, it does not contain any further information than that of $\mathcal{S}$.

The Semantics

As in section 3.1 we define a set-theoretical semantics for the formulae.

3.47 Definition (Frame): Let $\mathcal{S}_\Sigma$ be the sorted signature of a logic $\mathcal{L}_\Sigma$. A *frame* corresponding to $\mathcal{S}_\Sigma = (\mathcal{S}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$ is a collection $\{\mathcal{D}_\kappa\}_\kappa$ of nonempty sets $\mathcal{D}_\kappa$, one for each sort symbol $\kappa \in \Sigma$, such that $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_{(\kappa_1 \times \cdots \times \kappa_m \to \mu)} \subseteq \mathcal{F}(\mathcal{D}_{\kappa_1}, \ldots, \mathcal{D}_{\kappa_m}; \mathcal{D}_\mu)$ for all sorts κ_i, μ as well as $\mathcal{D}_\kappa \subseteq \mathcal{D}_\mu$ for all sorts κ, μ with $\kappa \dot{\sqsubseteq} \mu$. The members of $\mathcal{D}_o$ are called *truth values* and the members of $\mathcal{D}_\iota$ are called *individuals*.

The definitions of assignment, and interpretation are analogous to the corresponding definitions of the unsorted case, that is, definitions 3.11 and 3.12:

3.48 Definition (Interpretation): An *interpretation* $\langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J} \rangle$ of $\mathcal{L}^\omega_\Sigma$ consists of a frame and a function $\mathcal{J}$ that maps each constant of sort κ of $\mathcal{L}^\omega_\Sigma(\mathcal{S}_\Sigma)$ to an element of $\mathcal{D}_\kappa$, that is, for every constant declaration $(c : \kappa)$ we have $\mathcal{J}(c) \in \mathcal{D}_\kappa$.

3.49 Definition (Assignment): An *assignment* into a frame $\{\mathcal{D}_\kappa\}_\kappa$ is a function ξ that maps each variable of sort κ, that is, each variable x with $\mathfrak{s}(x) = \kappa$ of $\mathcal{L}^\omega_\Sigma$ to an element of $\mathcal{D}_\kappa$.

3.50 Definition (Weak Interpretation): An interpretation $\mathcal{M} = \langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J} \rangle$ is a *weak interpretation* (weak model, general model) for $\mathcal{L}^\omega_\Sigma(\mathcal{S}_\Sigma)$ (or $\mathcal{L}^\omega_{\Sigma,\equiv}(\mathcal{S}_\Sigma)$) with

$\mathcal{S}_\Sigma = (\mathcal{S}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$ iff there is a binary function $\mathcal{V}^{\mathcal{M}}$ so that for every assignment ξ and for every term t of sort κ, $\mathcal{V}^{\mathcal{M}}_\xi(t) \in \mathcal{D}_\kappa$, that is, in particular for every term declaration $(t : \kappa)$ in $\boldsymbol{\delta}$, $\mathcal{V}^{\mathcal{M}}_\xi(t : \kappa) = (\mathcal{V}^{\mathcal{M}}_\xi(t) \in \mathcal{D}_\kappa)$ and the following conditions hold:

1. for all variables x_κ, $\mathcal{V}^{\mathcal{M}}_\xi(x_\kappa) = \xi(x_\kappa)$
2. for all constants c_κ, $\mathcal{V}^{\mathcal{M}}_\xi(c_\kappa) = \mathcal{J}(c_\kappa)$
3. for composed terms $\mathcal{V}^{\mathcal{M}}_\xi(f_{(\kappa_1 \times \cdots \times \kappa_m \to \mu)}(t_{\kappa_1}, \ldots, t_{\kappa_m})) =$
   $\mathcal{V}^{\mathcal{M}}_\xi(f_{(\kappa_1 \times \cdots \times \kappa_m \to \mu)})(\mathcal{V}^{\mathcal{M}}_\xi(t_{\kappa_1}), \ldots, \mathcal{V}^{\mathcal{M}}_\xi(t_{\kappa_m}))$
4. $\mathcal{V}^{\mathcal{M}}_\xi(\varphi \wedge \psi) = \mathcal{V}^{\mathcal{M}}_\xi(\varphi) \wedge \mathcal{V}^{\mathcal{M}}_\xi(\psi)$
5. $\mathcal{V}^{\mathcal{M}}_\xi(\neg\varphi) = \neg\mathcal{V}^{\mathcal{M}}_\xi(\varphi)$
6. $\mathcal{V}^{\mathcal{M}}_\xi(\forall x_\kappa \varphi) = \forall d \in \mathcal{D}_\kappa \mathcal{V}^{\mathcal{M}}_{\xi[x_\kappa \leftarrow d]}(\varphi)$
7. for a model of $\mathcal{L}^{\omega}_{\Sigma,\equiv}$ we have in addition for all terms t_1 and t_2 that are terms of sort κ_1, κ_2 with the same top sort μ unequal to o:
   $\mathcal{V}^{\mathcal{M}}_\xi(t_1 \equiv_{(\mu \times \mu \to o)} t_2) = (\mathcal{V}^{\mathcal{M}}_\xi(t_1) \equiv_{\mathcal{D}_\mu} \mathcal{V}^{\mathcal{M}}_\xi(t_2))$

3.51 Definition (Strong Interpretation): An interpretation $\mathcal{M} = \langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J} \rangle$ is a *strong interpretation* (strong model, standard model) iff it is a weak interpretation and for all occurring sorts κ with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$, $\mathcal{D}_\kappa = \mathcal{F}(\mathcal{D}_{\kappa_1}, \ldots, \mathcal{D}_{\kappa_m}; \mathcal{D}_\mu)$.

3.52 Definition: Let φ be a formula of $\mathcal{L}^n_\Sigma(\mathcal{S}_\Sigma)$, and $\mathcal{M}$ be a weak interpretation of $\mathcal{L}^n_\Sigma(\mathcal{S}_\Sigma)$. $\mathcal{M}$ is a weak *model* of φ iff for every assignment ξ into $\mathcal{M}$, $\mathcal{V}^{\mathcal{M}}_\xi(\varphi) = \mathtt{T}$. We write $\mathcal{M} \models \varphi$. (Analogously for strong models.)

3.53 Remark: In chapter 5 we will give translations of the *unsorted* higher-order logics $\mathcal{L}^n$ into the *many-sorted* first-order logics $\mathcal{L}^1_\Lambda$ and of the general *order-sorted* higher-order logics $\mathcal{L}^n_\Sigma$ into the *order-sorted* logics $\mathcal{L}^1_\Sigma$. That is, in all cases the order-sorted logics of Arnold Oberschelp [106], which have been operationalized by Christoph Walther [129] and Manfred Schmidt-Schauss [117], serve as target logics. Hence in all cases we can use theorem provers like the Markgraf Karl Refutation Procedure [93] for proving theorems in these logics.

3.54 Remark: The extensionality axioms Ξ_Σ now have the following form:

Ξ^f_Σ For all function symbols f, g of sort κ, κ' of the same top sort $(\kappa_1 \times \cdots \times \kappa_m \to \mu)$, $\mu \neq o$:
$\forall f \ \forall g \ (\forall x_{\kappa_1} \ldots \forall x_{\kappa_m} \ f(x_{\kappa_1}, \ldots, x_{\kappa_m}) \equiv g(x_{\kappa_1}, \ldots, x_{\kappa_m})) \Longrightarrow f \equiv g$

Ξ^p_Σ For all predicate symbols p, q of sort κ, κ' of the same top sort $(\kappa_1 \times \cdots \times \kappa_m \to o)$:
$\forall p \ \forall q \ (\forall x_{\kappa_1} \ldots \forall x_{\kappa_m} \ p(x_{\kappa_1}, \ldots, x_{\kappa_m}) \iff q(x_{\kappa_1}, \ldots, x_{\kappa_m})) \Longrightarrow p \equiv q$

The comprehension axioms Υ_Σ now are the set of formulae:

Υ^f_Σ For every term t of sort $\kappa \neq o$ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of sort $\kappa_1, \ldots, \kappa_m, \mu_1, \ldots, \mu_k$:
$\forall y_1 \ldots \forall y_k \ \exists f_{(\kappa_1 \times \cdots \times \kappa_m \to \kappa)} \ \forall x_1 \ldots \forall x_m \ (f(x_1, \ldots, x_m) \equiv t)$.

Υ^p_Σ For every formula φ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of sort $\kappa_1, \ldots, \kappa_m, \mu_1, \ldots, \mu_k$:
$\forall y_1 \ldots \forall y_k \ \exists p_{(\kappa_1 \times \cdots \times \kappa_m \to o)} \ \forall x_1 \ldots \forall x_m \ (p(x_1, \ldots, x_m) \iff \varphi)$.

3.3 Extensions

Later on we shall need three extensions of the logics as introduced above. One concerns the arity, which we do not like to be fixed, so as to let room for optional parameters, the second concerns the variability of sorts and types, and the last is a special treatment for inductively defined concepts.

Optional Parameters

Let us have a look at optional parameters by considering the definition of a group. Mathematicians often use the following formulation "*Let* $\langle G, + \rangle$ *be a group then* ..." or "*Let* $\langle G, + \rangle$ *be a group with neutral element* 0 *and inverse function* $-$ *then* ...". In order to transfer these informal parlor into the presented formal language we introduce optional parameters.

In order to define a group we need indeed only two parameters: a set and a binary operation on that set. If these two parameters are given, it is well defined whether or not the pair $\langle G, + \rangle$ is a group, provided we have the corresponding definitions. We do not need to know the name of the neutral element or the name of the inverse function. But these names may become important later on. For example, if we define the set of rational numbers $\mathbb{Q}$ and the common operations on $\mathbb{Q}$ and if we want to check that $\langle \mathbb{Q}, +, \cdot \rangle$ is a field with additive neutral element

"0" and multiplicative neutral element "1", we must be able to express that the already defined "0" of $\mathbb{Q}$ is the neutral element in the group $\langle \mathbb{Q}, + \rangle$, and that the already defined "1" is the neutral element in the group $\langle \mathbb{Q}\backslash\{0\}, \cdot \rangle$. We can do that by writing $group(\mathbb{Q}, +, 0) \wedge group(\mathbb{Q}\backslash\{0\}, \cdot, 1)$. So optional parameters are used not only to give names to objects, but also to formulate stronger properties. If we did not allow optional parameters, in the example we would have to define three different predicates $group2$, $group3$, and $group4$ and thereby we would introduce unnecessary redundancies.

Optional parameters are allowed only in predicate constants. We allow them by an explicit declaration of the arity of a predicate symbol $l \leq \mathtt{arity}(P) \leq m$ (with l, m explicit natural numbers). For instance: $2 \leq \mathtt{arity}(group) \leq 4$ or $2 \leq \mathtt{arity}(ex\_left\_neutral\_element) \leq 3$.

For the definition of a predicate constant P with arity $l \leq \mathtt{arity}(P) \leq m$ every formula φ containing $P(t_1, \ldots, t_k)$ with $l \leq k \leq m$ is an abbreviation for φ with $P(t_1, \ldots, t_k)$ replaced by $(\exists z_{k+1}, \ldots, z_m \;\; P(t_1, \ldots, t_k, z_{k+1}, \ldots, z_m))$.

Variable Sorts

For the representation of certain concepts it is useful to have variable sorts. For instance if we want to define the concept group, we want to say "$\forall G : (\iota \rightarrow o) \;\; \forall + : (G \times G \rightarrow G) \;\; group(G, +) \iff \ldots$", that is, we want to use the predicate variable G as a sort symbol. The general treatment of these sorts is difficult, especially it may be difficult to give a clear semantics for it (as in parametrized types for programming languages). We allow these expressions only as an abbreviation for the corresponding relativization, that is, $\forall G : (\iota \rightarrow o) \;\; \forall + : (G \times G \rightarrow G) \;\; group(G, +) \iff \ldots$ is the abbreviation for $\forall G : (\iota \rightarrow o) \;\; \forall + : (\iota \times \iota \rightarrow \iota) \;\; function(+, G \times G \rightarrow G) \Longrightarrow (group(G, +) \iff \ldots)$. We write $\forall x : C \;\; P(x)$ as abbreviation for $\forall x \;\; C(x) \Longrightarrow P(x)$. In general we write unary predicates in a sort like manner, but we always use them only as abbreviation for the corresponding relativization. For a general discussion of relativization see section 5.3.1. Nevertheless it would be interesting to have a general theory for variable sorts and to treat them in a special manner and not to translate via relativization. By *function* we mean the following predicate:

$$function(f, X_1 \times \cdots \times X_m \rightarrow Y) \iff$$
$$(\forall x_1 \;\; X_1(x_1) \Longrightarrow (\cdots \forall x_m \;\; X_m(x_m) \Longrightarrow Y(f(x_1, \ldots, x_m)) \cdots)).$$

In the example above, $function(f, G \times G \rightarrow G)$ means
$\forall x : \iota \;\; G(x) \Longrightarrow (\forall y : \iota \;\; G(y) \Longrightarrow G(x + y))$.

Term declarations $(0 : G)$ are synonymous to $G(0)$.

Constructors

A common way of introducing concepts is via an inductive definition. Our higher-order logics are rich enough to cover this situation, but because of the particular importance of this, we will provide special facilities for defining concepts by mathematical induction. Let us give a standard example, namely that of the natural numbers: The PEANO axioms are [110] (written in our sorted higher-order language):

1. $\mathbb{N}\sqsubseteq\iota$
2. $(0:\mathbb{N})$
3. $(s:(\mathbb{N}\to\mathbb{N}))$
4. $\neg\exists n{:}\mathbb{N}\;\; 0=s(n)$
5. $\forall n,m{:}\mathbb{N}\;\; s(n)=s(m)\Longrightarrow n=m$
6. $\forall P{:}(\mathbb{N}\to o)\;\; P(0)\land(\forall n{:}\mathbb{N}\;\; P(n)\Longrightarrow P(s(n)))\Longrightarrow(\forall n{:}\mathbb{N}\;\; P(n))$

In the field of inductive theorem proving (compare e.g. [69]) this is often abbreviated to the data structure $\mathbb{N}$:

data structure		$\mathbb{N}$
constructors	base	$(0:\mathbb{N})$
	step	$(s:(\mathbb{N}\to\mathbb{N}))$,

where the semantics of this data structure is given by the formula set above.
In general we can introduce a data structure by:

- $P:(\kappa\to o)$ (or $P\sqsubseteq\kappa$) with $\kappa\neq o$
- $(c_j:P)$ for $1\leq j\leq n$, the c_j are called constructor constants.
- $(f_i:(P\times\kappa_1^i\times\cdots\times\kappa_{k_i}^i\to P))$ for $1\leq i\leq m$, the f_i are called constructor functions.

This data structure abbreviates the following set of higher-order formulae:

- there is a constant $P:(\kappa\to o)$ (or a sort $P\sqsubseteq\kappa$).
- for all constructor constants c_j, there are constants of sort P and for all constructor functions f_i, there are constants of sort $(P\times\kappa_1^i\times\cdots\times\kappa_{k_i}^i\to P))$.

- for all constructors c_j and all constructors f_i, there is a formula $\forall x{:}P\ \ \forall x_1 : \kappa^i_1 \ldots \forall x_{k_i} : \kappa^i_{k_i}\ \ c_j \not\equiv f_i(x, x_1, \ldots, x_{k_i})$, that is, no constructor constant is in the range of any constructor function.

- for all constructors f_i $\forall x, x'{:}P\ \ \forall x_1, x'_1 : \kappa^i_1 \ldots \forall x_{k_i}, x'_{k_i} : \kappa^i_{k_i}$
  $f_i(x, x_1, \ldots, x_{k_i}) \equiv f_i(x', x'_1, \ldots, x'_{k_i}) \Longrightarrow x \equiv x' \wedge x_1 \equiv x'_1 \wedge \ldots \wedge x_{k_i} \equiv x'_{k_i}$,
  that is, all constructor functions are injective.

- for all constructors f_i, f_l $i \neq l$, $\forall x : P, \forall x_1 : \kappa^i_1 \ldots \forall x_{k_i} : \kappa^i_{k_i}\ \ \forall x' : P, \forall x'_1 : \kappa^l_1 \ldots \forall x'_{k_l} : \kappa^l_{k_l}\ \ f_i(x, x_1, \ldots, x_{k_i}) \not\equiv f_l(x', x'_1, \ldots, x'_{k_l})$, that is, the ranges of the constructor functions are disjoint.

- $\forall Q{:}(P \rightarrow o)$
  $$\begin{array}{ll}
  (Q(c_1) \wedge \ldots \wedge Q(c_n) & \wedge \\
  (\forall x{:}P\ \ \forall x_1 : \kappa^1_1 \ldots \forall x_{k_1}{:}\kappa^1_{k_1}\ \ Q(x) \Longrightarrow Q(f_1(x, x_1, \ldots, x_{k_1}))) & \wedge \\
  \vdots & \wedge \\
  (\forall x{:}P\ \ \forall x_1 : \kappa^m_1 \ldots \forall x_{k_m}{:}\kappa^m_{k_m}\ \ Q(x) \Longrightarrow Q(f_m(x, x_1, \ldots, x_{k_m})))) & \\
  \Longrightarrow \forall x{:}P\ \ Q(x) &
  \end{array}$$
  that is, the induction principle holds.

The intention of this definition is not only to abbreviate certain formulae, but above all, if we formulate a concept with the help of such a data structure, a system can know that it is likely that certain properties have to be shown by complete induction. Although *in principle* the general framework is sufficient, *in practice* a system may only be successful if the induction hypothesis is given explicitly or a special proof procedure is used.

Chapter 4

Representation of Mathematical Knowledge

> Das wird nächstens schon besser gehen,
> Wenn Ihr lernt alles reduzieren
> Und gehörig klassifizieren.
>
> *Johann Wolfgang Goethe, Faust I*

In this chapter we are going to describe how to represent mathematical "factual" knowledge for automated theorem proving. Guideline is knowledge like that of a mathematical dictionary. In particular we are (in this chapter) not interested in heuristic knowledge as described in [113, 114, 115, 55, 126]. The main means for our representation is the logic introduced in the previous chapter. Indeed one may ask why logic is not sufficient for the description of the factual knowledge of mathematics, because it has been developed in the last hundred years for that purpose. The answer is that it is possible to find many extra-logical features in the presentation of mathematics, in mathematical text books for instance, and that these features are essential for mathematical activities like theorem proving.

What are the shortcomings of logic that make a mathematical knowledge representation necessary? One main point is, that the basic notion of logic is that of a formula, but that in mathematics different kinds of formulae are distinguished, namely *axioms*, *definitions*, and *theorems* and we will subdivide these kinds even more. Furthermore in mere logic a knowledge base consists of an unstructured set of formulae, whereas text books are well-structured and mathematicians spend a lot of time in the final presentation of the mathematical content. In addition there are constraints – which are not present in logic – in the procedure of stating theorems or defining concepts. The most important constraint for definitions is, that all concepts which are used in the definition – with the exception of the definiendum of course – must already be known. Analogously all concepts in theorems must be known. But even if all concepts are known, a definition has to fulfil further extra-logical requirements, for example, it is normally given in a form as abstract as possible.

Perhaps the main difference between logic and mathematics can be seen in the *conceptual* representation in mathematics. The standard schema of this procedure in mathematical text books is: "definition", "example", "theorem", "proof". When we look closer at this procedure, we see that the introduction of a concept is not terminated by giving a definition, but that examples, counter-examples, and lemmata about the introduced concept immediately belong to the concept. They do that in such an extent that it is possible to say: you have not understood the concept if you know only the definition, but you have not seen any examples and you do not know the simple properties of it.

Another great difference between logic and mathematics is that in mathematics it is always assumed – even if it cannot be proved – that a knowledge base is consistent, whereas by logic certain formulae are related, but it does not matter whether the preconditions are fulfilled or not. (This mathematical assumption is also the main reason for the completeness of the set-of-support strategy in resolution theorem proving.)

Which requirements should a knowledge representation formalism for mathematics satisfy? There are the following properties we would like to see:

- The knowledge base should be consistent and the representation formalism should support to keep it consistent.
- The representation formalism should reflect the different types of knowledge, that is, axioms, definitions, and theorems should be distinguished.
- The knowledge base should be redundancy free.
- It should not be possible to use unknown concepts.
- It should be possible that the knowledge can be represented in a conceptual, structured way.
- The representation formalism should be powerful enough in order to represent the knowledge easily.
- The formalism should have a clear semantics.

Now we discuss to what extent we can realize the requirements above.

The consistency of a knowledge base is of particular interest, because otherwise it is possible to derive anything of it. Unfortunately it cannot be shown in general because of GÖDEL'S incompleteness result, when a representation language such as first-order or higher-order logic is used. But we can restrict the possibilities where inconsistencies may be imported: definitions and theorems should not lead to any inconsistencies, because definitions form conservative extensions and theorems are proved to be consequences. So only axioms can cause any trouble.

Of course we cannot guarantee that the definitions of concepts are *correct*, that is, in accordance with the general use of them. We can define the concept "group" as something quite different from the general use, but because we cannot import contradictions by a definition, other parts of the knowledge base not using this concept cannot be concerned with such a non-standard definition. (The importance of this fact for automated theorem proving is already noted by Robert S. Boyer and J Strother Moore in [17, p.13].)

The distinction of the three different kinds of knowledge is very important for consistency: If we guarantee that definitions are really definitions and that theorems are proved, we have to be careful only with the axioms. Therefore we will use in the following three different basic knowledge units, one for axioms, one for definitions, and one for theorems.

The redundancy freeness can partially be guaranteed by preventing that concepts are defined twice, but to some extent it will be left to the user of a system. In particular we do not exclude that the user can define the same concept twice by giving different names to it.

If we have a knowledge base Δ and we want to add a new knowledge unit ϑ, a check of the signatures of the knowledge base and the newly introduced unit can guarantee that all concepts used in ϑ, with the exception of a newly defined one, are already in Δ.

When introducing concepts we do not want to spread the knowledge about these concepts all over the whole knowledge base. Hence a concept should not just consist of its actual logical definition, but simple consequences, examples, or alternative definitions should immediately be associated with this concept. Therefore we introduce in the following a formalism to represent mathematical concepts, such that the knowledge associated with an object is representationally attached to that object.

Now we present a frame-based representation of mathematical knowledge – using the higher-order logics introduced in chapter 3. We introduce the frame approach by examples as we go along. Then we give a formal definition of the frame language and its semantics, discuss the properties of corresponding knowledge bases and finish this chapter by some considerations on the advantages and disadvantages of the chosen formalism.

4.1 The Representation Language

In this section we introduce our representational formalism. We use the *frame* representation of Marvin Minsky [97]. The original idea is to represent knowledge in an object-oriented way and to simulate thereby the knowledge organisation as it is presumably organized in the mind (of a mathematician). The frames should structure the knowledge and contain in particular the information how to use this knowledge. Although introduced in opposition to the logicistic wing of the AI community, a frame can (more or less) be viewed as a certain way of arranging predicate logical facts [56], [105, chap.7]. Since our facts are predicate logical expressions, in our case this corresponds to meta-logical facts. A frame consists of a *name*, *slots*, and *fillers*. Slots correspond to certain meta-predicates and the fillers are arguments of these predicates. The syntactical surface is that of a box, as in figure 4.1.

We begin our examples with the definition of the concept "associative". This definition could be given in our extended sorted higher-order logic as:

$$\forall C{:}(\iota \to o)\ \ \forall f{:}(C \times C \to C)\ \ associative(C, f) \iff \forall x, y, z{:}C\ \ f(f(x, y), z) \equiv f(x, f(y, z)).$$

and this is now represented in a frame in figure 4.1 below. Every frame belongs to one of the three kinds: definition, axiom, or theorem. The kind is here indicated by the keyword "**Definition:**" The name of the introduced concept follows after a colon. In this case it is "**associative**". In the upper right corner we give the type of the definition. The entry "property" means that the whole concept models a property; a standard translation of a "property definition" into predicate logic can be done by mapping it into a predicate symbol. A "property definition" represents the relationship between its parameters. (The other type of a concept definition is that of a "mapping concept". In this case a new object is created, of which an example is given in figure 4.4 below.)

Now the slots and the slot-fillers of the frames are introduced:

Definition:	**associative**		property
parameters:	C	$:(\iota \to o)$	
	f	$:(C \times C \to C)$	
definition:	$\forall x, y, z{:}C\ \ f(f(x, y), z) \equiv f(x, f(y, z))$		
context:	basic algebra		

Figure 4.1

The argument of the binary property **associative** is given in the slot "parameters". The number of parameters corresponds to the arity of the predicate symbol defined in the frame. In the slot "parameters" the *formal* parameters and their sorts are written and when using the defined object elsewhere they are then bound to the *actual* parameters.

In the slot "definition" we find a (higher-order) logical definition of the concept. In the case of a property definition, the entry consists of the part of the definition that follows the equivalence sign "$\Longleftrightarrow$".

The slot "context" is provided for structuring the whole knowledge base into different modules. If we introduce a partial order among these contexts, we have the usual taxonomical hierarchy. (Here more details may become necessary, for example, which information may be used by other modules.) For instance we might structure a large knowledge base as in figure 4.2. This example is taken from the algebra text-book of Bartel L. van der Waerden [127, p.xi]. Such a structure is standard in all fields of mathematics, therefore we have to provide a possibility for representing it. How this information can be used is discussed below.

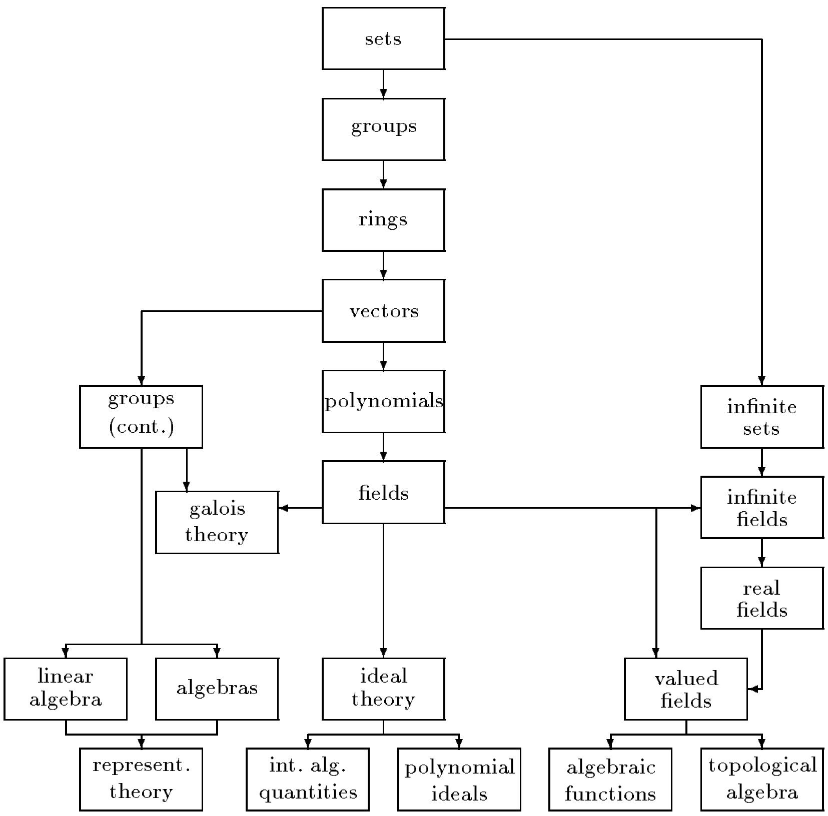


Figure 4.2

Now we shall introduce the second kind of frame, namely that of an axiom frame. In order to do so we take the first four axioms of Gödel's set axioms [53].

Axiom: set		
axioms:	$\forall x\!:\!Set\;\; Class(x)$	
	$\forall X,Y\!:\!Class\;\; X \in Y \Longrightarrow Set(X)$	
	$\forall X,Y\!:\!Class\;\; (\forall u\!:\!Set\;\; u \in X \iff u \in Y) \Longrightarrow X \equiv Y$	
	$\forall x,y\!:\!Set\;\; \exists z\!:\!Set\;\; (\forall u\!:\!Set\;\; u \in z \iff (u \equiv x \vee u \equiv y))^*$	
consequences:	1) $\forall x,y\!:\!Set\;\; y \in \{x\} \iff x \equiv y$	<proof-set-cons-1>
signature_ext:	$Class \sqsubseteq \iota$	
	$Set \sqsubseteq Class$	
	$\in\!: (Set \times Class \to o)$	
	$\{.,.\} : (Set \times Set \to Set)$	
context:	sets	

Figure 4.3

While definitions are always of the form "$\forall$*parameters definiendum*(*parameters*) $\iff$ *formula*", the slot "axioms" in contrast can contain arbitrary formulae, stating the properties of one or more function and/or predicate constants. If these constants are newly introduced by the frame, they have to be explicitly summarized in the slot "signature_ext". The slot "consequences" contains lemmata about the concepts introduced by the axioms. Such consequences must be proved; in <proof-set-cons-1> a pointer to such a proof is stored.

The logics $\mathcal{L}$ are expressible enough to give any of the standard axiomatizations of set theory such as ZFC, but for most cases it is sufficient to use sets of a certain sort κ as an abbreviation for a predicate of the sort $(\kappa \to o)$. If one follows this simple notion of a set, it is not possible to have elements of different types in one single set.

Now we give an example for a "mapping concept" and define a "pair".

Definition: pair			mapping
parameters:	x	$:Set$	
	y	$:Set$	
definition:	$\{x, \{x,y\}\}$		
sort:	Set		<proof-pair-sort>
main_property:	$\forall x,y,u,v\!:\!Set\;\; \langle x,y\rangle \equiv \langle u,v\rangle \iff x \equiv u \wedge y \equiv v$		<proof-pair-main_prop>
context:	sets		

Figure 4.4

We write $\langle x,y\rangle$ instead of $pair(x,y)$. The concept "pair" is an example where the entry in the definition slot itself is less important and indeed it is never used

*The existence of the variable z of sort Set can be written by the Skolem function $\{x,y\}$. As usual $\{x,x\}$ is abbreviated to $\{x\}$.

again. The definition is only given in order to have a set theoretical foundation of the concept as $\langle a, b\rangle :\equiv \{a, \{a, b\}\}$. The concept is closely related to a "main_property"; almost the only thing one has to know about the concept "pair" is: they are equal if and only if they agree on all arguments. In order to model main properties we introduce a corresponding new slot with this statement as filler. Another view of a main property is that this is the intrinsic meaning of the concept and that the definition is only an implementation of the concept in logic.

In the slot "sort" the sort of a "mapping"-concept is stored. Because we use sorts in a very general way, this sort information must be proved and hence in the slot we have to add a pointer to this proof.

In the next concept we use an "optional" parameter, which corresponds to the optional parameters of predicates in section 3.3. The name "*neutral_element*" is a selector of the tuple $(C, f, 0)$.

Definition:	**ex_left_neutral_element**		property
parameters:	C	$:(\iota \to o)$	
	f	$:(C \times C \to C)$	
(optional)	0	$:C$	(called *neutral_element*)
definition:	$\forall x{:}C\ \ f(0, x) \equiv x$		
context:	basic algebra		

Figure 4.5

Together with similarly definable concepts "ex_neutral_element" and "ex_inverse" it is now possible to introduce the concept "group".

Definition:		**group**		property
parameters:		G	$:(\iota \to o)$	(called *carrier*)
		$+$	$:(G \times G \to G)$	(called *operation*)
(optional)		0	$:G$	(called *neutral_element*)
(optional)		$-$	$:(G \to G)$	(called *inverse*)
definition:		$TRUE$		
superconcepts:	1)	$associative(G, +)$		
	2)	$ex\_neutral\_element(G, +, 0)$		
	3)	$ex\_inverse(G, +, 0, -)$		
equivalences:	1)	$associative(G, +) \wedge$		
		$ex\_left\_neutral\_element(G, +, 0) \wedge$		
		$ex\_left\_inverse(G, +, 0, -)$		<proof-group-equ-1>
	2)	$associative(G, +) \wedge$		
		$ex\_right\_neutral\_element(G, +, 0) \wedge$		
		$ex\_right\_inverse(G, +, 0, -)$		<proof-group-equ-2>
examples:	1)	$(Int, +, 0, -)$	model Integers	
	2)	$(Rat, +, 0, -)$	model Rationals	
	3)	$(Rat\backslash\{0\}, \cdot, 1, ^{-1})$	model Rationals	
context:		basic algebra		

Figure 4.6

The slot "superconcepts" expects slot-fillers that are generalizations of the actual concept. For instance every group is especially an associative structure. *TRUE* in the definition slot means that the concept is fully defined by the conjunction of its superconcepts.

The slot "equivalences" contains logical expressions that are necessary and sufficient to define the concept, that is, they are logically equivalent to the conjunction of the "definition" slot-filler and the "superconcepts" slot-fillers and form an alternative definition of the concept. In order to guarantee that the filler is really equivalent to the definition of the concept there must be a proof, which can be found in <proof-group-equ-1>. Although the different definitions of a concept are equivalent, they can be of different uses under different circumstances. For example, if one wants to prove that a certain object is a group, it is easier to use one of the two equivalent formulations, because then one has less to prove (e.g. $ex\_left\_neutral\_element(G,+,0)$ instead of $ex\_neutral\_element(G,+,0)$). The definition itself may be preferable if it is used as a premis, because then one can immediately use the property $ex\_right\_neutral\_element(G,+,0)$ without proving it first. In principle the frame approach is flexible enough to store such meta knowledge. But that is not integrated, in particular it is necessary to have a formal language to express meta knowledge. We can see an important difference between our epistemological term "property" and the logical term "predicate": The conceptual representation allows to make some assertions *about* the concepts (as for instance to use a certain variant of the definition in some situation). This would be difficult if we had mere predicates and distributed the knowledge in a whole set of formulae. Whether and how this can be used for actually guiding a theorem prover is a difficult question and not yet investigated deeply.

In the slot "examples" we can find a reference to a model of the corresponding concept. How examples can be represented and how it can be proved that an example is really an example is not investigated in this thesis. (See also the summary, chapter 7.)

We could have also given another definition for the concept "group" by writing the following formulae into the "definition" slot:

$$\begin{aligned}
&\forall x,y,z{:}G\ \ (x+y)+z \equiv x+(y+z) \wedge \\
&\exists x{:}G\ \ \forall y{:}G\ \ x+y \equiv y \equiv y+x\ (\wedge\ x \equiv 0) \wedge \\
&\forall x{:}G\ \ \exists y{:}G\ \ x+y \equiv 0 \equiv y+x\ (\wedge\ y \equiv -x)
\end{aligned}$$

But this is not as appropriate as the above formulation, because it is less structured. We want to formulate the concepts as abstract as possible for several reasons. If we formulate abstractly, we have the chance to find proofs also at a more high, abstract level. There is also the chance to find analogies or to use some special purpose algorithms that are defined for a special concept. For instance when the concept "group" is given, one might want to have a special treatment for the "+" as an associative operation. That could be done by not expanding the definition of associativity but by using a special equality reasoning procedure for associative function symbols, for instance, an A-unification algorithm.

So far we have given only examples for *simply* defined or axiomatized concepts. The next example is about *inductively* defined concepts. To this end we use the constructors introduced in the last part of section 3.3. We show how one can axiomatize the natural numbers with the constructors 0 and s.

Axiom:	**Nat**	
axiom:	base	$(0:\mathbb{N})$
	step	$(s:(\mathbb{N}\rightarrow\mathbb{N}))$
signature_ext:		$\mathbb{N}\sqsubseteq\iota$
		$0:\mathbb{N}$
		$s:(\mathbb{N}\rightarrow\mathbb{N})$
context:		numbers

Figure 4.7

The key words "base" and "step" correspond to the induction base and the induction step, respectively. In this example the induction base says that 0 is a natural number. The step case means that all natural numbers are constructed from 0 by the constructor function s.

Of course we could give second-order formulae in order to introduce the natural numbers, but because of its particular importance, we provide special facilities for inductively defined or axiomatized concepts.

Now we can define the function + for natural numbers.

Definition:	$+_{\mathbb{N}}$			mapping
parameters:		n	$:\mathbb{N}$	
		m	$:\mathbb{N}$	
sort:		$\mathbb{N}$		<proof-$+_{\mathbb{N}}$-sort>
definition:	base	$\forall n{:}\mathbb{N}\;\; n +_{\mathbb{N}} 0 \equiv n$		
	step	$\forall n,m{:}\mathbb{N}\;\; n +_{\mathbb{N}} s(m) \equiv s(n +_{\mathbb{N}} m)$		<proof-$+_{\mathbb{N}}$-def>
context:		numbers		

Figure 4.8

In this definition frame "base" and "step" correspond as in the axiom frame to the induction base and the induction step. For an inductive definition it must be shown that it is a definition indeed, that is, that all cases are covered and that the step case is well-founded (see e.g. [69]).

The next example shows an *implicit* definition. It is the general case of a definition and subsumes all previous cases as special cases. An implicit definition consists of an arbitrary formula set that uniquely characterizes the concept.

Definition:	**exp**		mapping
parameters:	x	$:\mathbb{R}$	
sort:	$\mathbb{R}$		<proof-exp-sort>
definition:	$\forall x, y{:}\mathbb{R}\ \ \exp(x+y) \equiv \exp(x)\cdot\exp(y)$		
	$\exp(1) \equiv e$		
	$\mathit{continous}(\exp, \mathbb{R})$		<proof-exp-def>
context:	real functions		

Figure 4.9

In order to make sure that it is a valid definition, the existence and uniqueness of the concept must be proved in <proof-exp-def>.

Partial functions can be defined in the following form:

Definition:	**reciprocal**		mapping
parameters:	x	$:\mathbb{R}$	
preconditions:	$x \not\equiv 0$		
sort:	$\mathbb{R}$		<proof-reciprocal-sort>
definition:	$\mathit{reciprocal}(x)\cdot x \equiv 1$		<proof-reciprocal-def>
context:	real functions		

Figure 4.10

The slot "preconditions" contains formulae, which must be satisfied, for the definition to make sense. A correct treatment of the preconditions requires *partial* functions. However we do not consider them in the following, because they require an essential extension of the logics $\mathcal{L}^n$, as for instance a transition from two-valued to three-valued logics. In particular we cannot translate them adequately into first-order (two-valued) logic as we do it with the other logics in chapter 5. For an overview of partial logics see [12, 78].

The concepts introduced so far are ordered by the fillers of the slot "super-concepts", which induce a transitive network of concepts with inheritance. That is, every concept inherits all definitions, consequences, equivalences, and counter-examples of its superconcepts. A superconcept itself inherits all examples of its subconcepts. A small section of this net can be seen in the following figure:

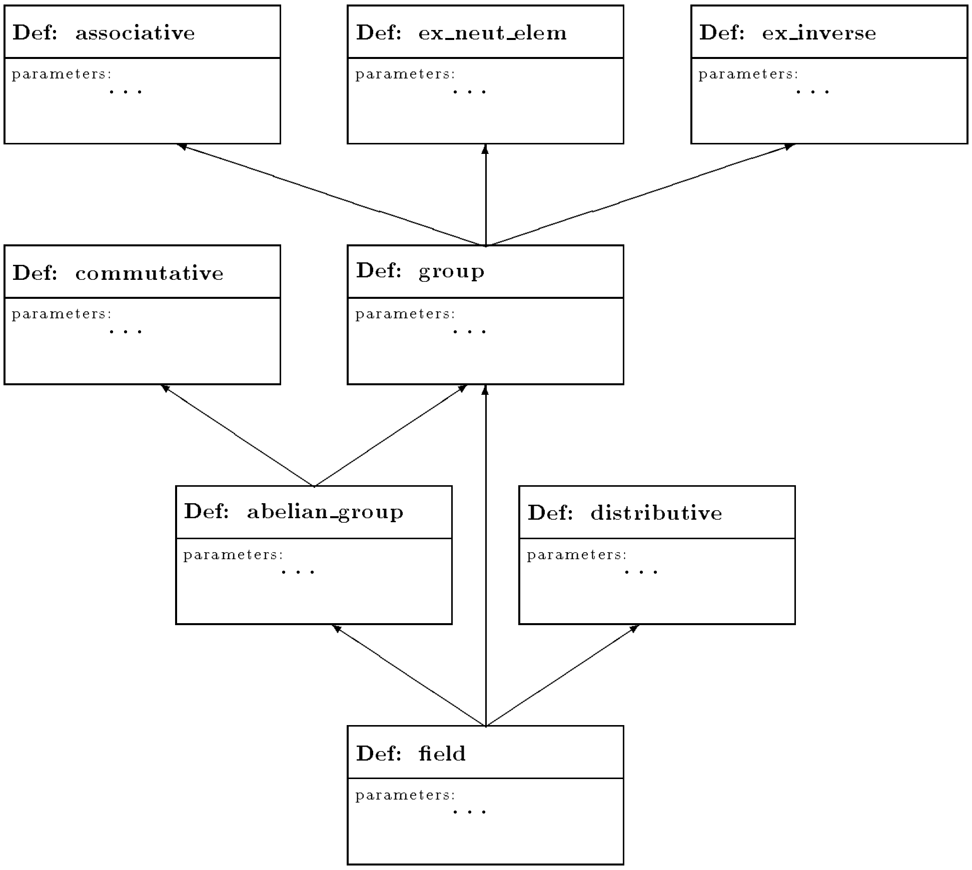


Figure 4.11

The edges in the network are labeled with the corresponding parameters in the "superconcepts" slot. In figure 4.11 we have omitted the labels for simplicity reasons. For instance, the edge from "field" to "abelian_group" expresses that a field $(F, +, 0, -, \cdot, 1, ^{-1})$ is an abelian group with respect to the *addition*, that is, $(F, +, 0, -)$ is an abelian group; whereas it is only a group with respect to the *multiplication*, that is, $(F \setminus \{0\}, \cdot, 1, ^{-1})$ is a group.

Finally we give an example for a theorem frame. We use CANTOR'S theorem that the power-set of a set has greater cardinality than the set itself in the formulation of ANDREWS [3, p.184].

Theorem: Cantor	theorem
theorem:	$\forall s : (\iota \to o)\ \neg\exists g : (\iota \to (\iota \to o))\ \forall f : (\iota \to o)$ $f \subseteq s \Longrightarrow (\exists j : \iota\ \ s(j) \wedge g(j) = f)$
status:	"proved"
proof:	<proof-Cantor>
context:	sets

Figure 4.12

The filler of the slot "theorem" is an arbitrary closed formula of $\mathcal{L}^n$ or $\mathcal{L}^n_\Sigma$.

The "status"-slot records whether the theorem is "proved", "conjectured", or "rejected".

In "proof" a pointer to a proof is given, if the status of the theorem is "proved" or "rejected".

After this informal introduction to our frame-based representation language, we shall now give a formal definition in the next section.

4.2 Formal Treatment

Now we define a general syntax of the three different kinds of frames. Their semantics is then given via translations into the underlying higher-order logic. The following frame shows all the different slots and possible fillers in a definition frame.

```
Definition:  <Name>                                              property | mapping
-------------------------------------------------------------------------------------
[ parameters:            { <variable_symbol>  :<sort_symbol>  [ (called <name>) ] }
[ { (optional)             <variable_symbol>  :<sort_symbol>  [ (called <name>) ] } ] ]
[ preconditions:         { <formula> } ]
definition:              { <formula> } |
                         { <term> }  |
                    base { <formula> }
                    step { <formula> }         <proof-<Name>-def>  |
                         { <formula> }         <proof-<Name>-def>
[ defined_symbols:       { <Name> } ]
[ superconcepts:   { #)  <concept>({ <variable_symbol> }) } ]
[ sort:                  <sort_symbol>         <proof-<Name>-sort> ]
[ main_property:         <formula>             <proof-<Name>-main_prop> ]
[ consequences:    { #)  <formula>             <proof-<Name>-cons-#> } ]
[ equivalences:    { #)  <formula>             <proof-<Name>-equ-#> } ]
[ examples:        { #)  <pointer_to_model>    <proof-<Name>-ex-#> } ]
[ counter_ex:      { #)  <pointer_to_model>    <proof-<Name>-counter_ex-#> } ]
[ used_in:               { <Name> } ]
[ subconcepts:           { <concept> } ]
context:                 <ContextName>
```

Figure 4.13

We use the extended BACKUS-NAUR form (EBNF) in the following way:
[.] stands for no or one occurrence, { . } for one or more repetitions, <.> for non-terminal symbols, and | for "or".

4.1 Definition (Definition Frame): A *definition frame* (written ϑ) is a list of the following elements:

1. "parameters" is of a list of variable_symbols with sorts and optionally selector names.

2. "preconditions" is a list of $\mathcal{L}$-formulae.

3. "definition" is either a simple definition, an inductive definition, or an implicit definition. If it is a simple definition it consists of an $\mathcal{L}$-formula or an $\mathcal{L}$-term corresponding to the type of the frame (property or mapping). The defined concept must not occur in that formula or term.

   In case of an inductive definition two slots must be filled, one for the base case and another for the step case, both with $\mathcal{L}$-formulae. For every constructor constant c_j the base case contains a formula of the form:
   $\forall x_1, \ldots, x_m$ <**Name**>$(c_j, x_1, \ldots, x_m) \iff \psi_j$, where the defined concept must not occur in ψ.
   For every constructor function f_i the step case contains a formula of the form $\forall x_1, \ldots, x_m \forall y, y_1, \ldots, y_{k_i}$ <**Name**>$(f_i(y, y_1, \ldots, y_{k_i}), x_1, \ldots, x_m) \iff \psi_i$ where in ψ_i no constructor function occurs. (In the case of a mapping concept "$\iff \psi$" is replaced by "$\equiv t$".) Inductive definitions must be shown to be definitions, that is, that all cases are covered and that the step case is well-founded. A pointer to a proof is stored in the slot.

   Implicit definitions consist of sets of formulae and a proof that the defined object exists and is unique.

4. "defined_symbols" is the list of symbols that are defined in an implicit definition, if more than one symbol is defined. In all other cases just the symbol <**Name**> is defined.

5. "superconcepts" is a list of atomic $\mathcal{L}$-formulae.

6. "sort" is a sort symbol that shows the sort of a concept of type "mapping". This slot must be filled when a definition of type "mapping" is given. The sort information has to be proved. A pointer to a prove is stored in the slot.

7. "main_property" is an $\mathcal{L}$-formula. The proof must show that the formula in this slot follows from the definition.

8. "consequences" and "equivalences" are lists of $\mathcal{L}$-formulae. The proofs show that the formulae are consequences or equivalences of the definition, respectively.

9. "examples" and "counter_ex" are lists of $\mathcal{L}$-structures, which are models or no models of the concept, respectively.

10. "used_in" is a list of axioms, definitions, and theorems. "subconcepts" is a list of axioms and definitions. These slots can be filled automatically, since the "used_in" slot is only a book-keeping slot and "subconcepts" is the inverse slot to the "superconcept" slot.

11. "context" is a name for a theory.

The frame must not contain any free variable except the variables of the slot "parameters". <**Name**> is a constant symbol of sort $(\kappa_1 \times \cdots \times \kappa_m \rightarrow \kappa)$, where κ_i is the sort of the i-th parameter and κ is the entry of the "sort" slot for a mapping frame and equal to o for a property frame.

4.2 Definition (Semantics of a Definition Frame): We fix the *semantics of a definition frame* by giving a logical translation. We begin with the translation of the parts 1 through 5 of definition 4.1. Let $x_1, \ldots, x_m$ be the variables of the parameter slot, $p_1, \ldots, p_l$ be the list of preconditions, φ be the entry of the definition slot and $s_1, \ldots, s_k$ be the entries of the superconcepts slot. The logical translation of a simple definition of type property is then:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (<\mathbf{Name}>(x_1, \ldots, x_m) \iff (s_1 \wedge \ldots \wedge s_m \wedge \varphi))$.
The $x_1, \ldots, x_m$ are the only free variable symbols in p_i, s_i, and φ. Optional parameters are treated in the same manner as in section 3.3. If the slot "preconditions" or "superconcepts" is unfilled it is translated by the same formula. Recall that an empty conjunction evaluates to *TRUE*.

In the case of a mapping frame the translation is:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (<\mathbf{Name}>(x_1, \ldots, x_m) \equiv s)$,
where s is an $\mathcal{L}$-term. The "sort" slot entry κ is translated as $(s : \kappa)$ or $\kappa(s)$.

In the case of an inductive or an implicit definition the translation is equivalent to:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow \varphi_1 \wedge \ldots \wedge \varphi_m$,
where the φ_i are the entries of the definition slot.

The other slots contain theorems, models, or structuring information.

The slot "consequences" contains theorems c_i that belong to the concept. The proved theorems are translated by the formulae itself, that is, by c_i for all i.*

*For instance if the union of sets is defined, we should write into this slot, that the union is associative, commutative, and idempotent.

The semantic status of an entry of the slot "main_property" is the same as that of an entry of the "consequences" slot.

Entries ψ of the slot "equivalences" are translated to:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (<\textbf{Name}>(x_1, \ldots, x_m) \iff \psi)$.

"subconcepts" is the inverse slot of "superconcepts". That means, whenever we make an entry in the "superconcepts" slot of concept A, that B is a superconcept of A, the slot "subconcepts" of B is automatically filled by the filler A. (Parameters are neglected in the "subconcepts" slot.) This slot – as well as the "used_in" and "context" slot – is not translated into logic, that is, it has no semantics and is only for pragmatic use.

The semantics of the slots "examples" and "counter_ex" is defined by the logical model relation, that is, $\mathcal{M}$ is an example iff $\mathcal{M} \models <\textbf{Name}>(x_1, \ldots, x_m)$, and $\mathcal{M}$ is a counter-example iff $\mathcal{M} \not\models <\textbf{Name}>(x_1, \ldots, x_m)$.

4.3 Remark: Superconcepts provide an inheritance relation, for instance "associative" is a superconcept for "group". So every consequence in the "associative"-frame is also a consequence for the "group"-frame. On the other hand "group" is a subconcept of "associative". So every example for a group is in particular an example for an associative structure.

Analogously one can define axiom frames and theorem frames. All frame types have a "consequence" and a "context" slot.

In the next figure all different slots of an axiom frame are shown:

Axiom:	**<Name>**		
axioms:		{ <formula> } |	
	base	{ <term_declaration> }	
	step	{ <term_declaration> }	
[signature_ext:		[{ <constant_symbol> : <sort_symbol> }] |	
		[{ <subsort_declaration> }]]	
[consequences:	{ #)	<formula>	<proof-<Name>-cons-#> }]
[examples:	{ #)	<pointer_to_model>	<proof-<Name>-ex-#> }]
[counter_ex:	{ #)	<pointer_to_model>	<proof-<Name>-counter_ex-#> }]
[used_in:		{ <name> }]	
context:		<ContextName>	

Figure 4.14

4.4 Definition (Axiom frame): An *axiom frame* (also written as ϑ) is a list of the following elements: "axioms", "signature_ext", "consequences", "examples", "counter_ex", "used_in", and "context" with:

1. The slot “axioms” contains either a set of formulae as simple axioms or new constructor symbols (compare page 35) are introduced. In the latter case the name of the axiom is a new predicate symbol (or sort symbol). In the “base”-subslot the constructor constants of this new sort are declared in the form of term declarations. In the “step”-subslot the constructor functions are declared in the same manner. These symbols have to occur also in the slot “signature_ext”.

2. “signature_ext” contains the axiomatized constants, if they are new.

3. All other slots are similar to the corresponding slots of a definition frame.

4.5 Definition (Semantics of an Axiom Frame): The *semantics of an axiom frame* is also given by the translation to $\mathcal{L}$. In the case of simple axioms it is translated into the conjunction of the formulae itself. In the case of inductive axioms they are translated into the corresponding data structure, which in turn can be translated as shown on pages 35 and following.

Finally we show what a theorem frame looks like in general:

Theorem: <**Name**>			[<theoremtype>]
[assumptions:		<closed formula>]	
theorem:		<closed formula>	
status:		“proved” | “conjectured” | “rejected”	
[proof:		<proof>]	
[consequences:	{ #)	<formula>	<proof-<Name>-cons-#> }]
context:		<ContextName>	

Figure 4.15

4.6 Definition (Theorem Frame): A *theorem frame* (also written as ϑ) is a list of: “assumptions”, “theorem”, “status”, “proof”, “consequences”, and “context” with:

1. “assumptions” is a list of formulae, which are preconditions for the theorem.

2. “theorem” is a formula.

3. “status” is either “proved”, “conjectured”, or “rejected”.

4. “proof” is a pointer to a proof[*] in the case that the status of the frame is “proved”. It is a pointer to a counterexample in the case that the status is “rejected”.

[*] So far we have not considered proofs at all. See chapter 5.

5. the other slots are just as above.

The <theoremtype> is used for the classification of the theorem as "lemma", "theorem", "main theorem", or "corollary".

4.7 Definition (Semantics of a Theorem Frame): The *semantics of a theorem frame* with entries $\varphi_1, \ldots, \varphi_m$ as assumptions and ψ as theorem is again given by a translation. In the case of the status "proved" it is: $\varphi_1 \wedge \ldots \wedge \varphi_m \Longrightarrow \psi$, otherwise, it is: *TRUE*.

4.8 Definition (Signature of a Frame): The *signature of a frame* ϑ is the set of all constants in the terms and formulae in ϑ. (Recall that we have no free variables in the frames, except the variables introduced in the "parameters"-slot.)

4.9 Definition (Extension of a Frame): A *frame-extension* ϑ' of a frame ϑ is a frame with the same name and classification, where all slot-fillers but the slots mentioned below are identical. The fillers of the slots consequences, equivalences, examples, counter_ex, and used_in can differ in the form, that the fillers of ϑ are sublists of the corresponding fillers of ϑ'; the slot main_property can differ, if it is not filled in ϑ. The status of a theorem frame may be changed from "conjectured" to "proved" or "rejected". In both cases the proof slot must be filled.

So far we have considered single frames, in the next section we discuss when many such frames form a knowledge base.

4.3 Building up a Knowledge Base

In this section we define the notion "knowledge base" and consider the consistency of knowledge bases. In particular we discuss the conditions when a definition forms a conservative extension and therefore does not endanger the consistency of a knowledge base. A knowledge base is defined inductively: to the empty knowledge base we can add frames under certain conditions.

4.10 Definition (Knowledge Base): A *knowledge base* Δ over $\mathcal{L}$ is defined inductively:

- as the empty knowledge base $\Delta_\emptyset = \emptyset$, or
- an immediate extension of a knowledge base.

Δ is called an *immediate extension* of a knowledge base Δ' iff

- $\Delta = \Delta' \cup \{\vartheta\}$ with axiom frame, definition frame, or theorem frame ϑ relative to the knowledge base Δ', or
- it is equal to a knowledge base Δ' for all but one entry and this entry is a frame-extension of the other. Formally $\Delta \backslash \{\vartheta\} = \Delta' \backslash \{\vartheta'\}$ and ϑ is a frame-extension of ϑ'.

The transitive closure of this relation is called an *extension*.

The *signature of a knowledge base* Δ is the union of all signatures of the frames contained in Δ. The logic with this signature is denoted by $\mathcal{L}(\Delta)$.

Now we define *frame relative to* a knowledge base:

- A theorem frame ϑ is called a *theorem frame relative to a knowledge base* Δ if it is a theorem frame, its signature is a subset of the signature of Δ. Furthermore if the status is "proved" or "rejected", the slot proof must contain a proof for "$\Delta \models$ (assumptions $\implies$ theorem)", respectively a counter-example for this relation. (For "proofs" see chapter 5.) The entries in the "consequences" slot must be proved too.
- An axiom frame ϑ is called an *axiom frame relative to a knowledge base* Δ if it is an axiom frame and all occurring constants in the frame are either in the signature of Δ or in the slot "signature_ext" of ϑ. The signature of Δ must be disjoint from the elements in "signature_ext". All lemmata in the frame – that is, all slots that must contain proofs – must fulfill the condition for a theorem frame above.
- A frame ϑ is a *definition frame relative to a knowledge base* Δ if it is a definition frame, the defined concept name(s) is (are) not in the signature of Δ, but all other constant symbols are. Lemmata in the frame must fulfill the condition for a theorem frame above.

For the following we need the notion of semantic consequence. A formula φ follows semantically (weakly/strongly) from a knowledge base Δ ($\Delta \models \varphi$ or $\Delta \models\!\!\equiv \varphi$), iff the translation of Δ into logic entails φ (weakly/strongly).

4.11 Definition (Consistency): A knowledge base Δ is called (weakly/strongly) *consistent* iff there is no formula φ so that $\Delta \models \varphi$ and $\Delta \models \neg\varphi$ (or $\Delta \models\!\!\equiv \varphi$ and $\Delta \models\!\!\equiv \neg\varphi$, respectively).

4.12 Definition (Conservativity): An extension Δ of a knowledge base Δ' is called (weakly/strongly) *conservative* iff for all formulae φ holds: If $\varphi \in \mathcal{L}(\Delta')$ then $\Delta' \models \varphi$ iff $\Delta \models \varphi$ (or with $\models\!\!\equiv$ instead of $\models$, respectively).

4.13 Remark: In particular by a conservative extension we cannot import any contradiction. If the knowledge base Δ' is consistent and Δ is a conservative extension of Δ', then Δ is also consistent, because otherwise we could deduce from Δ a formula φ and its negation $\neg\varphi$ and by this any formula in $\mathcal{L}(\Delta)$, and hence any in $\mathcal{L}(\Delta')$, so we would have $\Delta \models \varphi$ and $\Delta \models \neg\varphi$.

4.14 Definition (Definition-Conservativity): A knowledge base is called *definition-conservative* iff every definition is a conservative extension.

4.15 Remark: We would expect now that the logics $\mathcal{L}^n$ are definition-conservative. Unfortunately due to our definition of higher-order logics and their semantics this is not the case in general as can be seen in the next example.

4.16 Example: Let Δ consist of the following axioms:

- $a : \iota$, $b : \iota$, $R : (\iota \to o)$
- $\forall P{:}(\iota \times \iota \to o)\ \forall x, y{:}\iota\ \ P(x, y) \iff P(y, x)$
- $R(a) \wedge \neg R(b)$

These axioms are consistent since we can give a weak model: $\mathcal{D}_\iota = \{1, 2\}$, $\mathcal{D}_{(\iota\times\iota\to o)}$ consists only of the binary relations that map everything to $\mathtt{T}$. $\mathcal{J}(a) = 1$, $\mathcal{J}(b) = 2$ $\mathcal{J}(R)(1) = \mathtt{T}$, $\mathcal{J}(R)(2) = \mathtt{F}$. This is of course no longer a model if we "define" a new binary predicate Q by $\forall x, y\ Q(x, y) :\iff R(x) \wedge \neg R(y)$ and add this to our knowledge base. We have $Q(a, b)$ since $R(a)$ and $\neg R(b)$. On the other hand by the commutativity axiom we get $Q(b, a)$, hence $\neg R(a)$ and $R(b)$. That is, now we have a contradiction in our knowledge base. Hence $\mathcal{L}^n$ is not definition-conservative for $n > 1$. This cannot happen if we have all comprehension axioms in the knowledge base (compare definition 3.22).

4.17 Lemma: *Every knowledge base Δ is definition-conservative for implicit definitions.*

Proof: This lemma holds trivially because we have required for implicit definitions that they must contain a proof that the defined objects exist and are unique. ∎

4.18 Lemma: *$\mathcal{L}^1$ is definition-conservative.*

Proof: Inductive definitions are not possible in $\mathcal{L}^1$, hence we have to show the property only for simple definitions. Let Δ be given and ϑ, the extending frame of Δ, may be equivalent to $\forall x_1, \ldots, x_m$<**Name**>$(x_1, \ldots, x_m) \iff s_1 \wedge \ldots \wedge s_m \wedge \varphi$

where φ is the formula in the definition slot of the frame and the s_i are the entries of the superconcepts slot. So we can replace any instance of <**Name**>$(x_1,\dots,x_m)$ by the corresponding instance of $s_1 \wedge \dots \wedge s_m \wedge \varphi$. Since the two formulations are equivalent, the extension is conservative. In the case of a mapping definition the defined term is substituted instead of the formula. Of course we can make this replacement also, when preconditions are present. ■

4.19 Lemma: *If Δ contains the comprehension axioms Υ, then it is definition-conservative for simple definitions.*[*]

Proof: Let $\mathcal{S}'$ be the signature of Δ' and $\mathcal{S} = \mathcal{S}' \cup \{$<**Name**>$\}$ be the signature of $\Delta' \cup \{\vartheta\}$. ϑ is equivalent to $\forall x_1,\dots,x_m$ <**Name**>$(x_1,\dots,x_m) \iff s_1 \wedge \dots \wedge s_m \wedge \varphi$ where φ is the formula in the definition slot of the frame and the s_i are the entries of the superconcepts slot. By a comprehension axiom we get a predicate P so that $\forall x_1,\dots,x_m P(x_1,\dots,x_m) \iff s_1 \wedge \dots \wedge s_m \wedge \varphi$. So we can replace any instance of <**Name**>$(x_1,\dots,x_m)$ by the corresponding instance of $P(x_1,\dots,x_m)$.

For definitions of the type "mapping" we can proceed analogously. ■

4.20 Lemma: *If Δ contains the comprehension axioms Υ, then it is definition-conservative for inductive definitions.*

Proof: Let Δ be given and ϑ be an inductive definition. Let Q be the $m+1$-place predicate ($m \geq 0$) of sort $(\kappa \times \kappa_1 \times \dots \times \kappa_m \to o)$ that is defined per induction on the first argument, then the inductive definition is of the form:

base: $\forall x_1,\dots,x_m\ Q(c_j, x_1,\dots,x_m) \iff \psi_j$ for $j = 1,\dots,n$ where c_j are all constructor symbols of sort κ for the inductively introduced concept κ.

step: $\forall x_1,\dots,x_m \forall y, y_1,\dots,y_{k_i}\ Q(f_i(y, y_1,\dots,y_{k_i}), x_1,\dots,x_m) \iff \psi_i$ for all constructor functions f_i where in the ψ_i no constructor function can occur.

Now we have to show that there is exactly one predicate Q that satisfies this definition. By inductively applying the defining equivalences we get that for all constants d of sort κ, we can equivalently replace $Q(d, x_1,\dots,x_m)$ by an expression not containing Q. Consequently for all elements d of sort κ there is a simple definition of Q for this element as argument. Hence we can conclude that the general definition of Q forms a conservative extension. ■

[*] We do not need such axioms if we use for our higher-order logic the λ-calculus, since we get the corresponding constant simply by λ-abstraction. The comprehension axioms cannot explicitly be in our knowledge base, because we have not provided facilities for introducing axiom schemata, but it can be easily tested whether a formula is a comprehension axiom or not.

Summarizing we have:

4.21 Lemma: *If Δ contains the comprehension axioms Υ, then it is definition-conservative.*

4.22 Lemma: *If Δ is a consistent knowledge base and ϑ a theorem relative to Δ, then $\Delta \cup \{\vartheta\}$ is consistent.*

Proof: Because the deductive closures of Δ and $\Delta \cup \{\vartheta\}$ are the same, the theorem holds trivially. ∎

4.23 Lemma: *The empty knowledge base $\Delta_\emptyset$ is consistent.*

Proof: Since for all formulae φ with $\Delta_\emptyset \models \varphi$, φ is a tautology and φ and $\neg\varphi$ cannot be tautologies at once, $\Delta_\emptyset$ must be consistent. ∎

4.24 Theorem: *If Δ is a consistent knowledge base that contains the comprehension axioms Υ and ϑ is a concept definition relative to Δ or a theorem relative to Δ, then $\Delta \cup \{\vartheta\}$ is consistent.*

Proof: This follows immediately from the fact that concept definitions form conservative extensions and by conservative extension no contradiction can be imported, and by lemma 4.22. ∎

4.25 Remark: In our considerations we have neglected the "context" slot. It could be integrated by defining a partial order on the contexts and by sharpening the definition of a frame relative to a knowledge base (compare definition 4.10) in that way that the required properties have not to hold only for the whole knowledge base Δ and ϑ, but even for that subpart of the knowledge base that consists of those modules, which are in the reflexive, transitive closure of the module, to which ϑ belongs.

4.4 Critique of the Frame Approach

In this section we summarize the advantages of frames for the representation of mathematical concepts and state the main disadvantage.

The first reason why we use frames is the *concept oriented* way of representation. Frames enable us to represent the knowledge in such a way that all properties that belong immediately to a concept are represented together. In other words we can take into account that a concept does not only consist of its definition, but of a definition plus a set of important properties. So we can structure the definitions and some theorems and need not to distribute them over the whole knowledge base. These structuring facilities are not given in mere logic, but are provided by the frame language.

By the frame approach we can distinguish so-called *primitives*, that are, axioms, definitions, and theorems. Every primitive has its own status, in logic we only have formulae and do not distinguish between axioms, definitions, and theorems. These are meta-logical features of formulae. Here we have the possibility to require a special form for definitions, so that they are really definitions and cannot import any contradictions into a knowledge base. Theorems can be arbitrary formulae, but in contrast to axioms, they must be proved in order to be used in the proofs of other theorems. Furthermore we can guarantee that we use only defined concepts and exclude the case of "ignotum per ignotum". The consistency of knowledge bases is widely ensured.

Frames have the necessary *strength* to represent the required properties. For instance KL-ONE is not strong enough to represent a concept hierarchy, where an "abelian_group" is a superconcept of "field", since it is necessary to specify parameters, relative to which the hierarchic relation holds. Furthermore we can choose our slots, that is, primitives for describing concepts in an adequate way.

We can give (and have given) to the frames a *clear semantics*, what is important if one compares it to the situation of semantic networks, where years after building up large knowledge bases the untenability of the approach had to be stated, because it was impossible to give a clear semantics for the is-a hierarchy as long as "is-a" was used for $\in$ *and* $\subseteq$.

Another advantage compared to most other representation formalisms is the *flexibility* of frames, that is, it is easily possible to add new features to the frames. In particular it will be necessary to add meta-knowledge how to use all this knowledge. The usage of the knowledge in the frames is of course a very difficult problem, which will be left to the user for the beginning. Some heuristic information about the usage belongs to the concept. How this knowledge can be represented has to be cleared in the context of higher problem solving methods like proof planning and tactical theorem proving.

The main disadvantage of the proposed representation formalism is, that structuring, modularization, and classifications must be done by the user. Almost no automation can be expected in such a general approach.

Chapter 5

Translations

> Die Mathematik ist *nicht* nach ihrem Gegenstand (etwa: Raum und Zeit, Formen der inneren Anschauung, Lehre vom Zählen und Messen u. dergl.) zu charakterisieren, sondern, wenn man ihren ganzen Umfang erschöpfen will, allein durch ihr eigentümliches Verfahren, den Beweis.
>
> *Ernst Zermelo*

So far we presented the means to represent mathematical knowledge and the argumentation was essentially semantical. Now we want to introduce a calculus for theorem proving. In order to prove higher-order theorems mechanically there are two options: either to have a theorem prover for higher-order logic such as TPS (of Peter B. Andrews [4]) or to translate the higher-order constructs into corresponding first-order expressions and to use a first-order theorem prover. Even though the first approach is very important and may be the way of the future, we follow the second approach because strong first-order theorem provers are available today. For that purpose we present translations from different higher-order logics to sorted first-order logics, for which strong calculi and theorem provers exist. We begin with some general notions for these translations, then we present translations from the unsorted higher-order logic into many-sorted first-order logics with equality and give a sufficient criterion for the soundness of these translations. In addition, translations are introduced that are sound and complete with respect to Henkin's general model semantics. Finally we generalize these results to the sorted case.

5.1 Example: A common translation of the associativity formula (of page 20) into a first-order logic with equality is:

$$\begin{aligned}\forall + \quad associative(+) \iff \forall x, y, z \;\; & apply(+, apply(+, x, y), z) \equiv \\ & apply(+, x, apply(+, y, z))\end{aligned}$$

Here *apply* is a new function constant and $+$ an object variable. Although *apply* is interpreted freely it is intended that the interpretation of $apply(+, x, y)$ is exactly the same as the interpretation of the higher-order term $x + y$.

Another translation, without equality is:

$$\forall + \;\; associative(+) \iff \forall x, y, z, u, v, w \;\; apply(+, x, y, u) \;\wedge\; apply(+, u, z, w) \;\wedge \\ apply(+, y, z, v) \;\Longrightarrow apply(+, x, v, w)$$

Here *apply* is a predicate; again it is interpreted freely, although it is intended that z is the sum of x and y in $apply(+, x, y, z)$. In other words, different translations from higher-order to first-order logic are possible.

We have the following problems:

- What are the conditions that such a translation is correct? That is, if we translate a formula and we obtain a tautology, is the original formula a tautology too?
- In what sense can such a translation be called complete? That is, if we translate a tautology, do we always obtain a tautology?

5.2 Example: Consider the following tautology:

$\forall P, Q \;\; ((\forall x \;\; P(x) \Longrightarrow Q(x)) \wedge \forall x \;\; P(x) \Longrightarrow \forall x \;\; Q(x))$

This can be translated into the first-order formula

$\forall P, Q \;\; ((\forall x \;\; apply(P, x) \Longrightarrow apply(Q, x)) \wedge \forall x \;\; apply(P, x) \Longrightarrow \forall x \;\; apply(Q, x))$

This is obviously a tautology again, hence in this case the translation is sound and complete.

5.3 Example: Consider the following tautology with function constants f and g:

$\forall x \;\; f(x) \equiv g(x) \Longrightarrow f \equiv g$

This is a tautology because functions which have the same results for all arguments are equal (extensionality). It is translated to

$\forall x \;\; apply(f, x) \equiv apply(g, x) \Longrightarrow f \equiv g$

But this is not a tautology: by interpreting *apply* as the projection to the second component and f and g as different elements we obtain a counterexample. This translation is obviously not complete.

Since the expressiveness of higher-order logic is principally stronger than that of first-order, it is clear that if we find a translation from higher-order into first-order logic, it cannot be complete in general (especially because of the theorem of Löwenheim-Skolem and of Gödel's incompleteness result). In principle such a translation must be equivalent to some set theoretical formulation as stated in Mostowski's isomorphism theorem [100].*

*I would like to thank Heinrich Herre for introducing me to this work of Mostowski.

5.0.0.1 Related Work

ENDERTON [42] presented a sound and complete translation of unsorted second-order logic into many-sorted first-order logic. The types of the second-order expressions are translated to sorts of a many-sorted first-order logic. This will be generalized in this thesis to arbitrary higher-order logics. The completeness proof for second-order is easier than for the general case, because in second-order logic no extensionality axioms are necessary. Furthermore we will extend these translations to *sorted* higher-order logics.

HENSCHEN [58] developped a method to translate arbitrary higher-order (λ-) expressions into a many-sorted first-order logic. He presents a method to modify first-order theorem provers so that the comprehension axioms can be handled. His translation is not complete. In opposition to this, the translations in this thesis treat existing theorem provers as black boxes with the advantage that these can be used as they are, but with the drawback that the translation method is only well-suited for essentially first-order theorems.

VAN BENTHEM and DOETS [6] give a translation of a restricted higher-order logic without function symbols and without higher-order constants and identities into a standard first-order logic. They use the general idea of a translation, and show its soundness and completeness. The translation into an unsorted first-order logic leads to more complicated formulae than the translation into a sorted version, because of the need to relativize quantification with respect to the corresponding type.

The work of HANS JÜRGEN OHLBACH [107] has had a great influence on our translation techniques. He translates modal logics and other non-classical logics into a context logic, where contexts are restricted higher-order expressions. These contexts in turn are then translated into an order-sorted first-order logic.

Whereas in all these works one single translation from a source logic into a target logic is given, we will present in the following a whole class of sound translations from a higher-order logic $\mathcal{L}^n$ into first-order logic.

5.1 Logic Morphisms

Now we shall define the concepts that are necessary to describe the relationships between formalizations in different logics. The important concepts are: morphism, quasi-homomorphism, and soundness and completeness of a morphism.

5.4 Definition (Morphism of Logics): Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two logical systems ($\mathcal{L}^\omega$, $\mathcal{L}^\omega_\equiv$, $\mathcal{L}^n$, $\mathcal{L}^n_\equiv$, $\mathcal{L}^n_\Sigma$, $\mathcal{L}^n_\Lambda$, $\mathcal{L}^n_{\Sigma,\equiv}$, or $\mathcal{L}^n_{\Lambda,\equiv}$), then a *morphism* Θ is a mapping that maps the signature $\mathcal{S}_\Sigma$* of a logic in $\mathcal{F}_1$ to a signature of a logic $\Theta(\mathcal{S}_\Sigma)$ in $\mathcal{F}_2$ and that maps every formula set in $\mathcal{F}_1(\mathcal{S}_\Sigma)$ to a formula set in $\mathcal{F}_2(\Theta(\mathcal{S}_\Sigma))$.**

5.5 Definition (Soundness): Let Θ be a morphism from $\mathcal{F}_1$ to $\mathcal{F}_2$. Θ is called strongly (weakly) *sound* iff the following condition holds for every formula set Γ in $\mathcal{F}_1$:
if Γ has a strong (weak) model in $\mathcal{F}_1$ then there is a strong (weak) model of $\Theta(\Gamma)$ in $\mathcal{F}_2$.

5.6 Definition (Completeness): Let Θ be a morphism from $\mathcal{F}_1$ to $\mathcal{F}_2$. Θ is called strongly (weakly) *complete* iff the following condition holds for every formula set Γ in $\mathcal{F}_1$:
if $\Theta(\Gamma)$ has a strong (weak) model in $\mathcal{F}_2$ then there is a strong (weak) model of Γ in $\mathcal{F}_1$.

5.7 Definition (Quasi-Homomorphism): Let $\mathcal{F}_1(\mathcal{S}^1_\Sigma)$ and $\mathcal{F}_2(\mathcal{S}^2_\Sigma)$ be two logics. A morphism Θ from $\mathcal{F}_1$ to $\mathcal{F}_2$ is called a *quasi-homomorphism* from $\mathcal{F}_1(\mathcal{S}^1_\Sigma)$ to $\mathcal{F}_2(\mathcal{S}^2_\Sigma)$ iff the following conditions are satisfied:

1. The sorted signature $\mathcal{S}^1_\Sigma = (\mathcal{S}_1, \Sigma_1, \mathfrak{s}_1, \sqsubseteq_1, \boldsymbol{\delta}_1)$ is mapped to a signature $\mathcal{S}^2_\Sigma = (\mathcal{S}_2, \Sigma_2, \mathfrak{s}_2, \sqsubseteq_2, \boldsymbol{\delta}_2)$ so that,
   - 1.1 $\Theta(\mathcal{S}_1) \subseteq \mathcal{S}_2$, variables are mapped on variables and constants on constants by Θ,
   - 1.2 $\Theta(\Sigma_1) \subseteq \Sigma_2$, sort symbols are mapped on sort symbols by Θ,
   - 1.3 $\Theta(\mathfrak{s}_1(x)) = \Theta(\mathfrak{s}_1)(\Theta(x)) = \mathfrak{s}_2(x)$ for all variables x in $\mathcal{S}_1^{var}$,
   - 1.4 if $\kappa \mathrel{\dot\sqsubseteq_1} \mu$, then $\Theta(\kappa) \mathrel{\dot\sqsubseteq_2} \Theta(\mu)$, that is, $\Theta(\dot\sqsubseteq_1) \subseteq \dot\sqsubseteq_2$, and
   - 1.5 for every term declaration $(t : \kappa)$ in $\boldsymbol{\delta}_1$ we have that $\Theta(t)$ and every instantiation of $\Theta(t)$ has sort $\Theta(\kappa)$. Especially we have for every constant c of sort κ in $\mathcal{S}_1$ that $\Theta(c)$ has sort $\Theta(\kappa)$ in $\mathcal{S}_2$.
2. For all composed terms: if $f(t_1, \ldots, t_m)$ is a term of $\mathcal{F}_1(\mathcal{S}_1)$ of sort κ, then $\Theta\left(f(t_1, \ldots, t_m)\right) = \theta\left(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m)\right)$ is a term of sort $\Theta(\kappa)$ with
   $\theta(a, a_1, \ldots, a_m) = \begin{cases} a(a_1, \ldots, a_m) & \text{or} \\ \boldsymbol{\alpha}^a(a, a_1, \ldots, a_m) \end{cases}$

---

*As seen in remark 3.46 we obtain unsorted logics as special cases of sorted logics.

**A formula is considered as a formula set with one element. Especially we write $\Theta(\varphi)$ instead of $\Theta(\{\varphi\})$.

The $\boldsymbol{\alpha}$ have to be chosen appropriately out of $\mathcal{S}_2$: they have to be *new*, that is, there is no element e of $\mathcal{S}^1_\Sigma$ with $\boldsymbol{\alpha}^a = \Theta(e)$. The choice of the above depends only on the type of a, the symbol $\boldsymbol{\alpha}^a$ must respect the corresponding sorts of $a, a_1, \ldots, a_m$. ($\boldsymbol{\alpha}$ stands for *apply*.)

3. For all formulae φ_1, φ_2 and for all variables x:

   3.1 $\Theta(\varphi_1 \wedge \varphi_2) = \Theta(\varphi_1) \wedge \Theta(\varphi_2)$

   3.2 $\Theta(\neg\varphi) = \neg\Theta(\varphi)$

   3.3 $\Theta(\forall x \varphi) = \forall\Theta(x)\Theta(\varphi)$

5.8 Remark: The choice in θ provides a flexibility in translating from one into another logic. In particular we can choose different apply-functions although it would be possible to take the same. Thereby we can exclude certain instantiations in the target logic, but loose completeness in general. Which case we choose in θ depends on the *type* of the first element, not on the element itself.

5.9 Definition (Injectivity): A quasi-homomorphism Θ from $\mathcal{F}_1(\mathcal{S}^1_\Sigma)$ to $\mathcal{F}_2(\mathcal{S}^2_\Sigma)$ $\mathcal{S}^i_\Sigma = (\mathcal{S}_i, \Sigma_i, \mathfrak{s}_i, \sqsubseteq_i, \boldsymbol{\delta}_i)$ $(i = 1, 2)$ is called *injective* iff

1. for all elements e_1, e_2 of the signature $\mathcal{S}_1$ holds: if $\Theta(e_1) = \Theta(e_2)$, then $e_1 = e_2$,
2. for all sort symbols κ, μ in Σ_1 holds: if $\Theta(\kappa) = \Theta(\mu)$, then $\kappa = \mu$,
3. for all x in $\mathcal{S}^{var}_1$ holds: if $\mathfrak{s}_2(\Theta(x)) = \Theta(\kappa)$, then $\mathfrak{s}_1(x) = \kappa$,*
4. for all sorts κ, μ in $\mathcal{F}_1$ holds: if $\Theta(\kappa) \mathrel{\dot\sqsubseteq}_2 \Theta(\mu)$, then $\kappa \mathrel{\dot\sqsubseteq}_1 \mu$, and
5. for all terms t and all sorts κ in $\mathcal{F}_1$ holds: if $\Theta(t)$ is of sort $\Theta(\kappa)$, then t is of sort κ.

5.10 Remarks:

- We have excluded as quasi-homomorphism those mappings that map a formula like $P(a)$ on a formula $P(a, a)$. That is, arguments cannot be doubled. We could allow this without losing anything essential in the sequel, but the proofs would become more tedious, without gaining really in expressive power.

*This holds trivially by the definition of a quasi-homomorphism.

– If we translate into a first-order logic, the first case for $\theta(a, a_1, \ldots, a_m)$ can only be chosen if there is no quantifier on a.

In the following we presuppose the existence of a sound and complete calculus for order-sorted first-order logics as it has been developed for example by MANFRED SCHMIDT-SCHAUSS [117].

The following theorem elucidates our special interest in sound quasi-homomorphisms. It states the basic idea of the whole translation approach.

5.11 Theorem: *If Θ is a strongly (weakly) sound quasi-homomorphism from $\mathcal{L}^n$ to $\mathcal{L}^1_\Lambda$, $\vdash$ the derivability relation induced by a sound calculus of $\mathcal{L}^1_\Lambda$, Γ a formula set and φ a formula in $\mathcal{L}^n$ with $\Theta(\Gamma) \vdash \Theta(\varphi)$, then $\Gamma \models\!\!\!= \varphi$ (resp. $\Gamma \models \varphi$).*

Proof: Because of $\Theta(\Gamma) \vdash \Theta(\varphi)$ we have that $\Theta(\Gamma) \cup \{\neg\Theta(\varphi)\}$ is unsatisfiable. Because of homomorphy in $\neg$ there is no model of $\Theta(\Gamma \cup \{\neg\varphi\})$. Hence by soundness there is no model of $\Gamma \cup \{\neg\varphi\}$. In other words every model of Γ is a model of φ. Because of theorem 3.19 this conclusion holds for both strong and weak models. ■

5.2 Translations of Unsorted Higher-Order Logic

In the following we are interested in translations into $\mathcal{L}^1$ or $\mathcal{L}^1_\Lambda$, because there are well-known complete calculi and strong theorem provers for these calculi. If we find a sound translation the theorem above guarantees that proofs of a translated problem in $\mathcal{L}^1$ or $\mathcal{L}^1_\Lambda$ are also proofs for the original problem. Strong completeness of such a translation is not obtainable because of GÖDEL'S incompleteness result, but a priori nothing speaks against weak completeness, that is, there might be a morphism Θ from $\mathcal{L}^n$ to $\mathcal{L}^1$ such that, if $\Gamma \models \varphi$ then $\Theta(\Gamma) \vdash \Theta(\varphi)$ in $\mathcal{L}^1$. $\mathcal{L}^1$ is not really appropriate as the target logic for a translation, but a sorted version $\mathcal{L}^1_\Lambda$ is preferable, because in $\mathcal{L}^1_\Lambda$ it is not necessary to relativize the type information; hence these translations are structure conserving and the proofs can easily be mapped back. For a direct translation into $\mathcal{L}^1$ see BENTHEM and DOETS [6, p.316–320].

5.2.1 A Sufficient Criterion for Soundness

In this section we give a sufficient criterion for the soundness of translations of formulae of $\mathcal{L}^n$ into formulae of $\mathcal{L}^1_\Lambda$, which is strong enough to cover most requirements. In addition we give an example for such a sound translation.

5.12 Theorem: *If Θ is an injective quasi-homomorphism from $\mathcal{L}^n(\mathcal{S})$ to $\mathcal{L}^1_\Lambda(\mathcal{S}')$, then Θ is weakly sound.*

Proof: The proof consists of five steps. In the *first* step we introduce the basic notions, in particular a formula set Γ in $\mathcal{L}^n(\mathcal{S})$ and an arbitrary model $\mathcal{M} = \langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ of Γ. In the *second* step we construct out of the frame $\{\mathcal{D}_\tau\}_\tau$ a new one. The individuals of this frame consist of the sets $\widehat{\mathcal{D}}_{\Theta(\tau)}$ for $\tau \neq o$. We complete these sets to a frame by closing it as in theorem 3.19. The proper idea is to view the elements of $\mathcal{D}_\tau$ even for higher-order τ as individuals with additional properties. In the *third* step we define an interpretation function $\widehat{\mathcal{J}}$. For all constants c in $\mathcal{S}$ we define $\widehat{\mathcal{J}}(\Theta(c)) := \mathcal{J}(c)$. For the new constants $\boldsymbol{\alpha}^a$ we define $\widehat{\mathcal{J}}(\boldsymbol{\alpha}^a)$ as the application of the first argument to the rest. In the *fourth* step we show by induction on the construction of terms, that the quasi-homomorphism property is compatible with the model property. Formally we show $\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}} \circ \Theta = \mathcal{V}^{\mathcal{M}}_{\xi}$. In the *fifth* and last step we use this property to show that $\widehat{\mathcal{M}}$ is a model of $\Theta(\Gamma)$.

S t e p 1:

Let $\mathcal{M}$ be a weak model of a formula set Γ in $\mathcal{L}^n(\mathcal{S})$, then $\mathcal{M}$ is equal to $\langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ for a frame $\{\mathcal{D}_\tau\}_\tau$ and an interpretation function $\mathcal{J}$. $\mathcal{M}$ is a weak model for any φ out of Γ, that is, $\mathcal{V}^{\mathcal{M}}_{\xi}(\varphi) = \mathtt{T}$ for every assignment ξ. We are going to construct a model $\widehat{\mathcal{M}}$ out of $\mathcal{M}$ and show that $\widehat{\mathcal{M}}$ is a model of $\Theta(\varphi)$. In the sequel we abbreviate $\Theta(\tau)$ to $\tilde{\tau}$ for all types τ.

S t e p 2:

We define the sets $\widehat{\mathcal{D}}_{\tilde{\tau}} := \mathcal{D}_\tau$ for all types τ in $\mathcal{L}^n(\mathcal{S})$. Note that the τ are types and hence in particular they are sorts, which are mapped on sorts in $\mathcal{L}^1_\Lambda$. All the $\tilde{\tau}$ are sorts of type ι. For all sorts of $\mathcal{L}^1_\Lambda$, which are not in the image of Θ (if there are any), we can define the corresponding universe in an arbitrary way, since the corresponding symbols do not occur in $\Theta(\Gamma)$, for instance we can choose them all as $\widehat{\mathcal{D}}_\kappa = \{e\}$, the same singleton set.

The function universes and predicate universes are defined for all $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$ by $\widehat{\mathcal{D}}_\kappa = \mathcal{F}(\widehat{\mathcal{D}}_{\kappa_1}, \ldots, \widehat{\mathcal{D}}_{\kappa_m}; \widehat{\mathcal{D}}_\mu)$, where the κ_i are arbitrary sorts of type ι and μ is a sort of type ι or o. All these $\widehat{\mathcal{D}}_\kappa$ form our frame. Since we map to first-order, we have no sorts of second or higher order, so we have defined the frame $\{\widehat{\mathcal{D}}_\kappa\}_\kappa$ completely.

S t e p 3:

In this step we define the interpretation function $\widehat{\mathcal{J}}$ in order to complete the definition of an interpretation $\langle\{\widehat{\mathcal{D}}_\kappa\}_\kappa, \widehat{\mathcal{J}}\rangle$.

For all constants c of type τ in $\mathcal{S}$ we define $\widehat{\mathcal{J}}(\Theta(c)) := \mathcal{J}(c)$. This mapping is well-defined, because Θ is injective. In addition, we have that $\Theta(c)$ is of sort $\tilde{\tau}$, so that $\widehat{\mathcal{J}}(\Theta(c))$ must be an element of $\widehat{\mathcal{D}}_{\tilde{\tau}}$, which is so, because $\mathcal{J}(c) \in \mathcal{D}_\tau$ and $\widehat{\mathcal{D}}_{\tilde{\tau}} = \mathcal{D}_\tau$.

Then we define the interpretation function $\widehat{\mathcal{J}}$ for the new constants $\boldsymbol{\alpha}^a$. Let $a = \Theta(f)$ and f be of type $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, then $\boldsymbol{\alpha}^a$ must have the sort $\kappa = (\tilde{\tau} \times \tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma})$. Hence we must map it to an element of $\widehat{\mathcal{D}}_\kappa = \mathcal{F}(\widehat{\mathcal{D}}_{\tilde{\tau}}, \widehat{\mathcal{D}}_{\tilde{\tau}_1}, \ldots, \widehat{\mathcal{D}}_{\tilde{\tau}_m}; \widehat{\mathcal{D}}_{\tilde{\sigma}})$. We do that by defining $\widehat{\mathcal{J}}(\boldsymbol{\alpha}^a)(x, x_1, \ldots, x_m) = x(x_1, \ldots, x_m)$ for all $x \in \widehat{\mathcal{D}}_{\tilde{\tau}}$ and all $x_i \in \widehat{\mathcal{D}}_{\tilde{\tau}_i}$. This is well-defined since $\widehat{\mathcal{D}}_{\tilde{\tau}} = \mathcal{D}_\tau \subseteq \mathcal{F}(\mathcal{D}_{\tau_1}, \ldots, \mathcal{D}_{\tau_m}; \mathcal{D}_\sigma) = \mathcal{F}(\widehat{\mathcal{D}}_{\tilde{\tau}_1}, \ldots, \widehat{\mathcal{D}}_{\tilde{\tau}_m}; \widehat{\mathcal{D}}_{\tilde{\sigma}})$, hence x is a function of the required sort and the range sort is the range sort of $\boldsymbol{\alpha}^a$.

For all other constants c of sort κ we choose $\widehat{\mathcal{J}}(c)$ as an arbitrary element out of $\widehat{\mathcal{D}}_\kappa$. Since these elements do not occur in $\Theta(\Gamma)$, it does not matter how they are interpreted.

Step 4:

In this step we show that for every assignment $\widehat{\xi}$ in $\widehat{\mathcal{M}}$ there is an assignment ξ in $\mathcal{M}$, so that for all terms (and hence all formulae) t we have: $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}} \circ \Theta(t) = \mathcal{V}_{\xi}^{\mathcal{M}}(t)$.
Let $\widehat{\xi}$ be an arbitrary assignment in $\widehat{\mathcal{M}}$. We define ξ as $\xi(x) = \widehat{\xi}(\Theta(x))$. Now we prove $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}} \circ \Theta = \mathcal{V}_{\xi}^{\mathcal{M}}$ by induction on the construction of terms and formulae.
For terms we have:

- T1 For all variables x_τ, $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(x_\tau)) = \widehat{\xi}(\Theta(x_\tau)) \overset{\text{def } \xi}{=} \xi(x_\tau) = \mathcal{V}_{\xi}^{\mathcal{M}}(x_\tau)$.
- T2 For all constants c_τ, $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(c_\tau)) = \widehat{\mathcal{J}}(\Theta(c_\tau)) \overset{\text{def } \widehat{\mathcal{J}}}{=} \mathcal{J}(c_\tau) = \mathcal{V}_{\xi}^{\mathcal{M}}(c_\tau)$.
- T3 For all composed terms that start with an m-ary function term f with $a = \Theta(f)$ so that $\theta(a, \Theta(t_1), \ldots, \Theta(t_m))$ is defined as $a(\Theta(t_1), \ldots, \Theta(t_m))$ we have:
  $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(f(t_1, \ldots, t_m))) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\theta(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m))) =$
  $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(f)(\Theta(t_1), \ldots, \Theta(t_m))) \overset{\text{def}}{=} \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(f))(\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}\Theta(t_1), \ldots, \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}\Theta(t_m)) \overset{\text{Ind.hyp}}{=}$
  $\mathcal{V}_{\xi}^{\mathcal{M}}(f)(\mathcal{V}_{\xi}^{\mathcal{M}}(t_1), \ldots, \mathcal{V}_{\xi}^{\mathcal{M}}(t_m)) =$
  $\mathcal{V}_{\xi}^{\mathcal{M}}(f(t_1, \ldots, t_m))$.
- T4 For all composed terms that start with an m-ary function term f with $a = \Theta(f)$ so that $\theta(a, \Theta(t_1), \ldots, \Theta(t_m))$ is defined as $\boldsymbol{\alpha}^a(a, \Theta(t_1), \ldots, \Theta(t_m))$ we have:
  $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(f(t_1, \ldots, t_m))) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\theta(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m))) =$
  $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\boldsymbol{\alpha}^a(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m))) \overset{\text{def}}{=}$

$\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\boldsymbol{\alpha}^a)(\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}\Theta(f), \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}\Theta(t_1), \ldots, \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}\Theta(t_m)) \stackrel{\text{Ind.hyp}}{=}$
$\widehat{\mathcal{J}}(\boldsymbol{\alpha}^a)(\mathcal{V}_{\xi}^{\mathcal{M}}(f), \mathcal{V}_{\xi}^{\mathcal{M}}(t_1), \ldots, \mathcal{V}_{\xi}^{\mathcal{M}}(t_m)) \stackrel{\text{def}\,\widehat{\mathcal{J}}}{=} \mathcal{V}_{\xi}^{\mathcal{M}}(f)(\mathcal{V}_{\xi}^{\mathcal{M}}(t_1), \ldots, \mathcal{V}_{\xi}^{\mathcal{M}}(t_m)) =$
$\mathcal{V}_{\xi}^{\mathcal{M}}(f(t_1, \ldots, t_m))$.

For formulae we get:

F1 An atomic formula is a special term, hence we have already proved the required property above.

F2 For a conjunction we have:
$\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi_1 \wedge \varphi_2)) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi_1) \wedge \Theta(\varphi_2)) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi_1)) \wedge \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi_2)) \stackrel{\text{Ind.hyp}}{=}$
$\mathcal{V}_{\xi}^{\mathcal{M}}(\varphi_1) \wedge \mathcal{V}_{\xi}^{\mathcal{M}}(\varphi_2) = \mathcal{V}_{\xi}^{\mathcal{M}}(\varphi_1 \wedge \varphi_2)$.

F3 For a negation we have:
$\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\neg\varphi)) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\neg\Theta(\varphi)) = \neg\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi)) \stackrel{\text{Ind.hyp}}{=} \neg\mathcal{V}_{\xi}^{\mathcal{M}}(\varphi) = \mathcal{V}_{\xi}^{\mathcal{M}}(\neg\varphi)$.

F4 For a quantification we have:
$\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\forall x_\tau\varphi)) = \mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\forall x_{\hat{\tau}}\Theta(\varphi)) = \forall d \in \widehat{\mathcal{D}}_{\hat{\tau}}\mathcal{V}_{\widehat{\xi}[x_{\hat{\tau}} \leftarrow d]}^{\widehat{\mathcal{M}}}(\Theta(\varphi)) =$
$\forall d \in \mathcal{D}_\tau\mathcal{V}_{\widehat{\xi}[\Theta(x_\tau) \leftarrow d]}^{\widehat{\mathcal{M}}}(\Theta(\varphi)) \stackrel{\text{Ind.hyp}}{=} \forall d \in \mathcal{D}_\tau\mathcal{V}_{\xi[x_\tau \leftarrow d]}^{\mathcal{M}}(\varphi) = \mathcal{V}_{\xi}^{\mathcal{M}}(\forall x\varphi)$.

Here we use that $\widehat{\xi} \circ \Theta = \xi$ for all assignments and hence $\widehat{\xi}[\Theta(x_\tau) \leftarrow d] \circ \Theta = \xi[x_\tau \leftarrow d]$.

Hence we have shown that $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}} \circ \Theta = \mathcal{V}_{\xi}^{\mathcal{M}}$ holds for all formulae.

Step 5:

Now we use this property to show that $\widehat{\mathcal{M}}$ is a model of $\Theta(\Gamma)$. Let φ be an arbitrary formula in Γ. We have to show that $\widehat{\mathcal{M}}$ is a model of $\Theta(\varphi)$. Let $\widehat{\xi}$ be an arbitrary assignment in $\widehat{\mathcal{M}}$ and ξ' be defined as $\widehat{\xi} \circ \Theta$, then we can conclude if $\mathcal{V}_{\xi}^{\mathcal{M}}(\varphi) = \mathtt{T}$ for all assignments ξ in $\mathcal{M}$, then this relation holds especially for $\xi = \xi'$, so we get $\mathcal{V}_{\widehat{\xi}}^{\widehat{\mathcal{M}}}(\Theta(\varphi)) = \mathtt{T}$ for all assignments $\widehat{\xi}$ in $\widehat{\mathcal{M}}$, hence $\widehat{\mathcal{M}}$ is a model of $\Theta(\varphi)$. ∎

5.13 Theorem: *If Θ is an injective quasi-homomorphism from $\mathcal{L}^n(\mathcal{S})$ to $\mathcal{L}^1_\Lambda(\mathcal{S}')$, then Θ is strongly sound.*

Proof: If there is a strong model of a formula set Γ in $\mathcal{L}^n(\mathcal{S})$ then this model is also a weak model. By the previous theorem there is hence a weak model of $\Theta(\Gamma)$ in $\mathcal{L}^1_\Lambda(\mathcal{S}')$. By a sorted version of theorem 3.19 there is also a strong model of $\Theta(\Gamma)$. ∎

5.14 Example: Let us see how to translate the predicative definition of a group into first-order logic. We drop the type information for readability, *group* is of type $(\iota \times (\iota \times \iota \times \iota \to o) \to o)$, G of type ι, $+$ of type $(\iota \times \iota \times \iota \to o)$, $-$ of type $(\iota \times \iota \to o)$, and so on. (In the translation to first-order logic these types are transformed into the sorts ("ι", "$(\iota \times \iota \times \iota \to o)$") for *group*, and so on.) A group can be defined as follows*:

1. $\forall G, + \;\; group(G,+) \iff associative(G,+) \land$
   $\exists 0 \;\; (0 \in G \;\land\; neutral\_element(G,+,0) \land$
   $\exists - \;\; inverse(G,+,0,-))$

2. $\forall G, + \;\; associative(G,+) \iff$
   $\forall u,v,w,x,y,z \;\; u,v,w,x,y,z \in G \;\land\; +(x,y,u) \;\land\; +(u,z,w) \land$
   $+(y,z,v) \implies +(x,v,w)$

3. $\forall G,+,0 \;\; neutral\_element(G,+,0) \iff \forall x \;\; x \in G \implies$
   $+(x,0,x) \;\land\; +(0,x,x)$

4. $\forall G,+,0,- \;\; inverse(G,+,0,-) \iff \forall x,y \;\; x,y \in G \;\land\; -(x,y) \implies$
   $+(x,y,0) \;\land\; +(y,x,0)$

This formula set is a subset of $\mathcal{L}^3$. Now we give a translation into a formula set of $\mathcal{L}^1_\Lambda$. The signatures are obvious, hence omitted. The translation is sound, because it is an injective quasi-homomorphism. We choose the $\boldsymbol{\alpha}^a$ so that they depend only on the type of their arguments and write the sort of the first argument as a string of the corresponding type, that is, $\Theta(\iota \times \iota \to o)$ is written as "$(\iota \times \iota \to o)$".

1. $\forall G, + \;\; group(G,+) \iff associative(G,+) \land$
   $\exists 0 \;\; (0 \in G \;\land\; neutral\_element(G,+,0) \land$
   $\exists - \;\; inverse(G,+,0,-))$

2. $\forall G, + \;\; associative(G,+) \iff \forall u,v,w,x,y,z \;\; u,v,w,x,y,z \in G \land$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,x,y,u) \land$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,u,z,w) \land$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,y,z,v) \implies$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,x,v,w)$

3. $\forall G,+,0 \;\; neutral\_element(G,+,0) \iff \forall x \;\; x \in G \implies$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,x,0,x) \land$
   $\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,0,x,x)$

*We write in this example simply $x, y \in G$ instead of $x \in G \;\land\; y \in G$.

4. $\forall G,+,0,-\ \ inverse(G,+,0,-) \iff \forall x,y\ \ x,y\in G \wedge$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(-,x,y) \Longrightarrow$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,x,y,0) \wedge$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\times\iota\to o)"}(+,y,x,0)$$

This translation is clumsy, because we do not use equality; a translation with equality can be found under example 5.29 below.

5.15 Remark: Note that the formulae that are obtained by these translations are not essentially more complex than the original, as the structure of the formulae (number and position of quantifiers and junctors) is respected. In the image the terms are never more nested than in the original. The only thing that can change, is that the number of arguments in a term is increased by one. The following theorem is important for lifting back a the proof of a translated first-order theorem to a proof in higher-order logic.

5.16 Theorem: *If* Θ *is a quasi-homomorphism from* $\mathcal{L}^n(\mathcal{S})$ *to* $\mathcal{L}^1_\Lambda(\Theta(\mathcal{S})\cup\{\boldsymbol{\alpha}^a\})$*, then* Θ *is surjective.*

Proof: This follows immediately from the quasi-homomorphism property. ∎

5.17 Remark: As important as it is to find a proof as important it is to be able to communicate it. Of course it is very desirable to present the proof in the language, which has been input by the user, because he is familiar with it. If we have a first-order proof procedure, which produces a proof, for instance as a sequence of formulae, the final proof can be translated back into higher-order logic, because the mappings Θ are all bijective. In other words if we have a first-order calculus then this calculus provides a calculus for $\mathcal{L}^n$ by Θ^{-1}. Surely this cannot solve all problems of proof presentation (for some literature on proof presentation see page 108), but it shows that the translation method is well compatible with these methods.

5.2.2 The Standard Translation From Unsorted Higher-Order Logic to Sorted First-Order Logic

Now we want to define morphisms Φ_n from $\mathcal{L}^n$ to $\mathcal{L}^1_{\Lambda,\equiv}$ which are not only sound but also complete. We define the morphisms for odd n, for even n they are obtained as the restriction of the next higher odd n, that is $\Phi_{2n} := \Phi_{2n+1}\mid_{\mathcal{L}^{2n}}$. The morphisms Φ are defined for formulae as $\Phi(\varphi) = \Psi(\varphi)\cup\Xi_\alpha$, where Ψ is a quasi-homomorphism and Ξ_α are special extensionality axioms for the $\boldsymbol{\alpha}$ (compare remark 3.20). In the following we drop the index n. Again we abbreviate *apply* as $\boldsymbol{\alpha}$.

5.18 Definition (Standard Translation Φ_{2n-1}): Let $\mathcal{S}^{2n-1} = \bigcup_\tau \mathcal{S}_\tau = \bigcup_\tau \mathcal{S}_\tau^{const} \cup \bigcup_\tau \mathcal{S}_\tau^{var}$ be the signature of a logic in $\mathcal{L}^{2n-1}$. In order to define a morphism Φ to $\mathcal{L}^1_{\Lambda,\equiv}$, we have to define the signature $\widehat{\mathcal{S}}_\Sigma$ of the target logic and we have to fix how formulae are mapped.

Let $\widehat{\mathcal{S}}_\Sigma$ be equal to $(\widehat{\mathcal{S}}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$

1. At first we define $\widehat{\mathcal{S}}$. Let $\widehat{\mathcal{S}} = \bigcup_\tau \widehat{\mathcal{S}}_\tau^{const} \cup \bigcup_\tau \widehat{\mathcal{S}}_\tau^{var}$. $\widehat{\mathcal{S}}$ is the union of the following sets:

   (a) $\widehat{\mathcal{S}}_\iota^{const} = \bigcup\limits_{\mathrm{ord}(\tau)<n} \mathcal{S}_\tau^{const}$

   (b) $\widehat{\mathcal{S}}^{const}_{\underbrace{(\iota\times\cdots\times\iota}_{m}\to\iota)} = \{\boldsymbol{\alpha}^{\tilde{\tau}} \mid \mathrm{ord}(\tau) < n \wedge \tau = (\tau_1 \times \cdots \times \tau_m \to \sigma), \sigma \neq o\}$

   (c) $\widehat{\mathcal{S}}^{const}_{\underbrace{(\iota\times\cdots\times\iota}_{m}\to o)} = \bigcup\limits_{\substack{\mathrm{ord}(\tau)=n\\ \mathrm{arity}(\tau)=m}} \mathcal{S}_\tau^{const} \cup$
   $\{\boldsymbol{\alpha}^{\tilde{\tau}} \mid \mathrm{ord}(\tau) < n \;\wedge\; \tau = (\tau_1 \times \cdots \times \tau_m \to o)\}$
   with elements $\boldsymbol{\alpha}^{\tilde{\tau}}$ which are new, that is, which do not occur in $\mathcal{S}$. In addition, for $m = 2$ we have in $\widehat{\mathcal{S}}^{const}_{(\iota\times\iota\to o)}$ the equality sign $\equiv$.

   (d) $\widehat{\mathcal{S}}_\iota^{var} = \bigcup\limits_\tau \mathcal{S}_\tau^{var}$

2. Σ is defined as the set
   $\{\tilde{\tau} \mid \mathrm{ord}(\tau) < n\} \;\cup$
   $\{(\tilde{\tau} \times \tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma}) \mid \tau = (\tau_1 \times \cdots \times \tau_m \to \sigma) \;\wedge\; \mathrm{ord}(\tau) < n\} \;\cup$
   $\{(\tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{o}) \mid \tau = (\tau_1 \times \cdots \times \tau_m \to o) \;\wedge\; \mathrm{ord}(\tau) = n\}$
   The function $\tilde{}$ must map τ injectively to new names. We can realize this function for instance by mapping the type to the string consisting of the type, that means, $(\iota \times \iota \to \iota)$ is mapped to "$(\iota \times \iota \to \iota)$", what is viewed as a primitive and not as a composed object. Instead of $\tilde{o}$ we often shortly write o.

3. $\mathfrak{s}$ is defined for variables $\Phi(x_\tau)$ with $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$ as
   $$s(\Phi(x_\tau)) = \begin{cases} \tilde{\tau} & \text{for } \mathrm{ord}(\tau) < n \\ (\tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma}) & \text{for } \mathrm{ord}(\tau) = n \end{cases}$$

4. $\sqsubseteq$ is defined as

   (a) $\tilde{\tau} \sqsubseteq \iota$ for all τ with $\mathrm{ord}(\tau) < n$.

   (b) $(\tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma}) \sqsubseteq (\iota \times \cdots \times \iota \to \iota)$ for $\sigma \neq o$.

(c) $(\tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to o) \sqsubseteq (\iota \times \cdots \times \iota \to o)$

5. $\boldsymbol{\delta}$ is defined as the set of all term declarations

   (a) $(\boldsymbol{\alpha}^{\tilde{\tau}} : (\tilde{\tau} \times \tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma})$ for all $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma))$ with $\mathrm{ord}(\tau) < n$,

   (b) for all constants c of type τ of order less than n, $(\Theta(c) : \tilde{\tau})$,

   (c) for all constants c of type $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$ of order equal to n, $(\Theta(c) : (\tilde{\tau}_1 \times \cdots \times \tilde{\tau}_m \to \tilde{\sigma}))$, and

   (d) $(\equiv: (\tilde{\tau} \times \tilde{\tau} \to \tilde{o}))$ for all τ with $\mathrm{ord}(\tau) < n$ and $\tau \neq o$.

Now we are going to define how terms and formulae are mapped by the morphism Ψ, which behaves on the signature exactly like Φ, with the only exception that $\equiv$ is not in the image of Ψ.

For terms it is defined inductively by:

T1 For a term with an m-ary function term f of type τ as top expression we define
$\Psi(f(t_1, \ldots, t_m)) = \boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(f), \Psi(t_1), \ldots, \Psi(t_m))$

For formulae we define Ψ inductively by:

F1 For an atomic formula with predicate constant p of order n as top expression we define
$\Psi(p(t_1, \ldots, t_m)) = \Psi(p)(\Psi(t_1), \ldots, \Psi(t_m))$

F2 For a term with an m-ary predicate term p of type τ and order less than n as top expression we define
$\Psi(p(t_1, \ldots, t_m)) = \boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(p), \Psi(t_1), \ldots, \Psi(t_m))$

F3 For a conjunction we define
$\Psi(\varphi_1 \wedge \varphi_2) = \Psi(\varphi_1) \wedge \Psi(\varphi_2)$

F4 For a negation we define
$\Psi(\neg\varphi) = \neg\Psi(\varphi)$

F5 For a quantified formula we define
$\Psi(\forall x \varphi) = \forall \Psi(x) \Psi(\varphi)$

Ξ_α is the set consisting of the following formulae of $\mathcal{L}^1_{\Lambda,\equiv}$:

Ξ^f_α For every function constant $\boldsymbol{\alpha}^{\tilde{\tau}}$ with $\tau = (\tau_1 \times \cdots \times \tau_m \rightarrow \sigma)$, $\sigma \neq o$:
$$\forall f_{\tilde{\tau}} \forall g_{\tilde{\tau}} (\forall x^1_{\tilde{\tau}_1}, \ldots, \forall x^m_{\tilde{\tau}_m}$$
$$\boldsymbol{\alpha}^{\tilde{\tau}}(f, x^1, \ldots, x^m) \equiv \boldsymbol{\alpha}^{\tilde{\tau}}(g, x^1, \ldots, x^m)) \Longrightarrow f \equiv g$$

Ξ^p_α For every predicate constant $\boldsymbol{\alpha}^{\tilde{\tau}}$ with $\tau = (\tau_1 \times \cdots \times \tau_m \rightarrow o)$:
$$\forall p_{\tilde{\tau}} \forall q_{\tilde{\tau}} (\forall x^1_{\tilde{\tau}_1}, \ldots, \forall x^m_{\tilde{\tau}_m}$$
$$\boldsymbol{\alpha}^{\tilde{\tau}}(p, x^1, \ldots, x^m) \iff \boldsymbol{\alpha}^{\tilde{\tau}}(q, x^1, \ldots, x^m)) \Longrightarrow p \equiv q$$

We define $\Phi(\varphi) = \Psi(\varphi) \cup \Xi_\alpha$. Analogously for formula sets $\Phi(\Gamma) = \Psi(\Gamma) \cup \Xi_\alpha$.

5.19 Remark: It should become apparent, why we excluded types like $(o \rightarrow o)$: Let P be a predicate of this type, Q be a predicate of type $(\iota \rightarrow o)$, and c be a constant of type ι. Then $\Psi(P(Q(c)) \wedge Q(c))$ would be* $\boldsymbol{\alpha}^{"(o\rightarrow o)"}(P, \boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)) \wedge \boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)$ or $P(\boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)) \wedge \boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)$ which is not well-formed, because $\boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)$ has to be a formula and a term at once. Even worse in general a *uniform* (quasi-homomorphic) translation is not possible, because $Q(c)$ must be translated in the first case to a term and in the second to a formula. That is not allowed in first-order logic. This example is also a counterexample for the correctness of the translation given by BENTHEM and DOETS [6] for a language without function symbols.

A possible translation of the unrestricted typed higher-order logic also has to provide a translation of formulae of the kind $P(Q(c)) \wedge Q(c)$. This is possible by having only function symbols $\boldsymbol{\alpha}^{\tilde{\tau}}$ and translating all other symbols into object variables or object constants. Especially the junctor "$\wedge$" has also to be translated to a constant. A possible translation would be:
$\boldsymbol{\alpha}^{"(o\rightarrow o)"}(\wedge, \boldsymbol{\alpha}^{"(o\rightarrow o)"}(P, \boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)), \boldsymbol{\alpha}^{"(\iota\rightarrow o)"}(Q, c)) \equiv \mathit{TRUE}$. That is, the problem is completely encoded into an equality problem. In order to gain completeness it would be necessary to add axioms for the junctor "$\wedge$".

We have not defined translations of arbitrary formula sets of $\mathcal{L}^\omega$. For instance with the unary predicate symbols $P^1_{(\iota\rightarrow o)}$, $P^2_{((\iota\rightarrow o)\rightarrow o)}$, $P^3_{(((\iota\rightarrow o)\rightarrow o)\rightarrow o)}, \ldots$, the formula set $\Gamma = \bigcup_{n\geq 1}\{P^{n+1}(P^n)\}$ is not translatable. Of course our mappings Φ_n could be extended to a mapping Φ_ω in such a way that we have as predicates only the $\boldsymbol{\alpha}^{\tilde{\tau}}$. We have not done this, because in all practical cases only finitely many formulae are involved and so we can have a translation Φ_n. This gives a translation that preserves the property of being a predicate for as many symbols as possible.

5.20 Lemma: *Ψ is an injective quasi-homomorphism from $\mathcal{L}^{2n-1}(\mathcal{S})$ to $\mathcal{L}^1_\Lambda(\Psi(\mathcal{S}))$.*

*We write here for all types τ the corresponding sort $\tilde{\tau}$ as "τ".

Proof: In order to show that Ψ is a quasi-homomorphism, we can check the definition step by step. The required properties for the signatures hold by definition of $\Psi(\mathcal{S})$. For terms and formulae, we have defined the relation in such a way that it holds immediately. The injectivity follows from the fact, that different types are mapped on different sorts in $\mathcal{L}^1_\Lambda$. ■

5.21 Theorem: Φ *is weakly sound.*

Proof: Let $\mathcal{M} = \langle\{\mathcal{D}_\tau\}_\tau, \mathcal{J}\rangle$ be a weak model of a formula set Γ, hence $\mathcal{M}$ is a weak model for any φ out of Γ, that is, $\mathcal{V}^{\mathcal{M}}_\xi(\varphi) = \mathtt{T}$ for every assignment ξ. We are going to show that the $\widehat{\mathcal{M}}$ in the proof of theorem 5.12 is a model of $\Phi(\varphi)$. By theorem 5.12 and lemma 5.20 we have that Ψ is sound.

So it remains to be shown that the extensionality property holds for the $\boldsymbol{\alpha}$-functions. Formally:

$$\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}}\begin{pmatrix}\forall f_{\tilde{\tau}}\forall g_{\tilde{\tau}}(\forall x^1_{\tilde{\tau}_1},\ldots,\forall x^m_{\tilde{\tau}_m} \\ \boldsymbol{\alpha}^{\tilde{\tau}}(f,x_1,\ldots,x_m) \equiv \boldsymbol{\alpha}^{\tilde{\tau}}(g,x_1,\ldots,x_m)) \Longrightarrow f \equiv g\end{pmatrix} = \mathtt{T}.$$

Therefore it is necessary to prove that for all F, G in $\widehat{\mathcal{D}}_{\tilde{\tau}}$ holds: if for all $X_1 \in \widehat{\mathcal{D}}_{\tilde{\tau}_1}, \ldots, X_m \in \widehat{\mathcal{D}}_{\tilde{\tau}_m}$, $\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[f,g,x_1,\ldots,x_m \leftarrow F,G,X_1,\ldots,X_m]}(\boldsymbol{\alpha}^{\tilde{\tau}}(f,x_1,\ldots,x_m)) = \mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[f,g,x_1,\ldots,x_m \leftarrow F,G,X_1,\ldots,X_m]}(\boldsymbol{\alpha}^{\tilde{\tau}}(g,x_1,\ldots,x_m))$, then $\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[f,g\leftarrow F,G]}(f) = \mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[f,g\leftarrow F,G]}(g)$, that is, $F = G$.

By the definition of $\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[f,g,x_1,\ldots,x_m \leftarrow F,G,X_1,\ldots,X_m]}$ and the interpretation of $\boldsymbol{\alpha}^{\tilde{\tau}}$ the precondition is equivalent to $F(X_1,\ldots,X_m) = G(X_1,\ldots,X_m)$. Because two functions are the same, if they have the same values on all arguments, we get $F = G$.

The axioms for the predicates can be proved to be true analogously. ■

5.22 Remark: Φ is strongly sound, analogously to theorem 5.13.

We shall now show that Φ is weakly complete in the sense of definition 5.6.

5.23 Theorem: Φ *is weakly complete.*

Proof: The proof consists again of five steps. In the *first* step we introduce the basic notion, in particular a formula set Γ in $\mathcal{L}^n(\mathcal{S})$ and an arbitrary model $\mathcal{M}$ of $\Phi(\Gamma)$. In the *second* step we define a frame with the help of which we will define a model for Γ. This model is defined inductively with the induction base $\check{\mathcal{D}}_\iota := \mathcal{D}_{\tilde{\iota}}$ and $\check{\mathcal{D}}_o := \mathcal{D}_{\tilde{o}}$. For composed types $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$ we define $\check{\mathcal{D}}_\tau$ as a subset of $\mathcal{F}(\check{\mathcal{D}}_{\tau_1},\ldots,\check{\mathcal{D}}_{\tau_m};\check{\mathcal{D}}_\sigma)$. We cannot take the whole set, because then we would try to obtain strong completeness, which cannot be achieved in general. In order to construct $\check{\mathcal{D}}_\tau$ we make use of the interpretations of the $\boldsymbol{\alpha}$-functions, especially we construct injective functions $\natural$, which map $\mathcal{D}_{\tilde{\tau}}$ to $\check{\mathcal{D}}_\tau$. In a *third* step we define an interpretation function $\check{\mathcal{J}}$ for $\mathcal{L}^n$. In a *fourth* step we show by induction on the construction of terms and formulae that the quasi-homomorphism Ψ is compatible with the model relation. Formally we show $\natural \circ \mathcal{V}_\xi^{\mathcal{M}} \circ \Psi = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}$. In a *fifth* and last step we use this property to show that $\check{\mathcal{M}} = \langle \{\check{\mathcal{D}}_\tau\}_\tau, \check{\mathcal{J}} \rangle$ is a model of Γ.

Step 1:

Let Γ be a formula set in $\mathcal{L}^{2n-1}(\mathcal{S})$. Let $\mathcal{M}$ be a weak model of $\Phi(\Gamma)$. Then $\mathcal{M}$ is a model of $\Phi(\varphi)$ for every formula φ in Γ. Let $\mathcal{M}$ be $\langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J} \rangle$ and ξ be an arbitrary assignment. Then we have $\mathcal{V}_\xi^{\mathcal{M}}(\Phi(\varphi)) = \mathtt{T}$. We want to construct a model $\check{\mathcal{M}}$ of φ, so that for all assignments $\check{\xi}$ we have $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi) = \mathtt{T}$.

Step 2:

In this step we define a frame for $\mathcal{L}^{2n-1}(\mathcal{S})$. Therefore we define $\check{\mathcal{D}}_\iota := \mathcal{D}_{\tilde{\iota}}$ and $\check{\mathcal{D}}_o := \mathcal{D}_{\tilde{o}}$. For all other types τ with $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$ we have to define $\check{\mathcal{D}}_\tau \subseteq \mathcal{F}(\check{\mathcal{D}}_{\tau_1}, \ldots, \check{\mathcal{D}}_{\tau_m}; \check{\mathcal{D}}_\sigma)$. We do this by inductively defining injective functions $\natural_\tau$ from $\mathcal{D}_{\tilde{\tau}}$ to $\mathcal{F}(\check{\mathcal{D}}_{\tau_1}, \ldots, \check{\mathcal{D}}_{\tau_m}; \check{\mathcal{D}}_\sigma)$ and setting $\check{\mathcal{D}}_\tau := \natural_\tau(\mathcal{D}_{\tilde{\tau}})$. Hence $\natural_\tau$ is a bijective function from $\mathcal{D}_{\tilde{\tau}}$ to $\check{\mathcal{D}}_\tau$.* We define $\natural_\tau$ as bijective functions inductively:

1. $\natural_\iota : \mathcal{D}_{\tilde{\iota}} \to \check{\mathcal{D}}_\iota$ and $\natural_o : \mathcal{D}_{\tilde{o}} \to \check{\mathcal{D}}_o$ as the identity mappings (These functions are obviously bijective).

2. Let $\natural_{\tau_i}$ and $\natural_\sigma$ be defined for $\mathcal{D}_{\tilde{\tau}_1}, \ldots, \mathcal{D}_{\tilde{\tau}_m}$, and $\mathcal{D}_{\tilde{\sigma}}$. We are going to define a function $\natural_\tau$ with $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\sigma \neq o$, for $\mathcal{D}_{\tilde{\tau}}$. For all $x \in \mathcal{D}_{\tilde{\tau}}$ $\natural_\tau(x)$ is defined as $\natural_\tau(x)(\check{x}_1, \ldots, \check{x}_m) := \natural_\sigma(\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x, \natural_{\tau_1}^{-1}(\check{x}_1), \ldots, \natural_{\tau_m}^{-1}(\check{x}_m)))$ for all $\check{x}_1 \in \check{\mathcal{D}}_{\tau_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\tau_m}$

   The following diagram may help to see the involved mappings at a glance:

$$\begin{array}{ccccccc}
\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}}) : \mathcal{D}_{\tilde{\tau}} & \times & \mathcal{D}_{\tilde{\tau}_1} & \times \cdots \times & \mathcal{D}_{\tilde{\tau}_m} & \longrightarrow & \mathcal{D}_{\tilde{\sigma}} \\
! \downarrow \natural_\tau & & \uparrow \natural_{\tau_1}^{-1} & & \uparrow \natural_{\tau_m}^{-1} & & \downarrow \natural_\sigma \\
\check{\mathcal{D}}_\tau & \hookrightarrow \mathcal{F}(& \check{\mathcal{D}}_{\tau_1} & , \ldots , & \check{\mathcal{D}}_{\tau_m} & ; & \check{\mathcal{D}}_\sigma)
\end{array}$$

   In order to show the injectivity of $\natural_\tau$ we use that we have in Ξ_α^f the formula

$$\forall f_{\tilde{\tau}} \forall g_{\tilde{\tau}} (\forall x_{\tilde{\tau}_1}^1, \ldots, \forall x_{\tilde{\tau}_m}^m \; \boldsymbol{\alpha}^{\tilde{\tau}}(f, x^1, \ldots, x^m) \equiv \boldsymbol{\alpha}^{\tilde{\tau}}(g, x^1, \ldots, x^m)) \Longrightarrow f \equiv g$$

   Therefore we have in a model for all x, x' in $\mathcal{D}_{\tilde{\tau}}$

$$\forall y_1 \in \mathcal{D}_{\tilde{\tau}_1}, \ldots, \forall y_m \in \mathcal{D}_{\tilde{\tau}_m} \; \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x, y_1, \ldots, y_m) \equiv_{\mathcal{D}_{\tilde{\sigma}}} \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x', y_1, \ldots, y_m) \Longrightarrow x \equiv_{\mathcal{D}_{\tilde{\tau}}} x' \qquad (*)$$

*Since we cannot achieve bijectivity from $\mathcal{D}_{\tilde{\tau}}$ to $\mathcal{F}(\check{\mathcal{D}}_{\tau_1}, \ldots, \check{\mathcal{D}}_{\tau_m}; \check{\mathcal{D}}_\sigma)$ we do not get strong completeness. We have $\mathcal{D}_{\tilde{\tau}} \xrightarrow{\text{bij}} \check{\mathcal{D}}_\tau \xrightarrow{\text{inj}} \mathcal{F}(\check{\mathcal{D}}_{\tau_1}, \ldots, \check{\mathcal{D}}_{\tau_m}; \check{\mathcal{D}}_\sigma)$.

Let $\natural_\tau(x) \equiv_{\check{\mathcal{D}}_\tau} \natural_\tau(x')$ for arbitrary x and x' in $\mathcal{D}_{\tilde{\tau}}$. Then we have by definition for all $\check{x}_1 \in \check{\mathcal{D}}_{\tau_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\tau_m}$
$\natural_\sigma \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x, \natural_{\tau_1}^{-1}(\check{x}_1), \ldots, \natural_{\tau_m}^{-1}(\check{x}_m)) \equiv_{\check{\mathcal{D}}_\sigma} \natural_\sigma \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x', \natural_{\tau_1}^{-1}(\check{x}_1), \ldots, \natural_{\tau_m}^{-1}(\check{x}_m))$.
Since the mappings $\natural_{\tau_1}, \ldots, \natural_{\tau_m}, \natural_\sigma$ are all bijective, we get for all $y_1 \in \mathcal{D}_{\tilde{\tau}_1}, \ldots, y_m \in \mathcal{D}_{\tilde{\tau}_m}$: $\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x, y_1, \ldots, y_m) \equiv_{\mathcal{D}_{\tilde{\sigma}}} \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x', y_1, \ldots, y_m)$. Because of the relation $(*)$ $x \equiv_{\mathcal{D}_{\tilde{\tau}}} x'$, hence the injectivity is shown. Since the surjectivity is given by definition, we have proved that $\natural_\tau$ is bijective.

3. Let $\natural_{\tau_i}$ be defined for $\mathcal{D}_{\tilde{\tau}_1}, \ldots, \mathcal{D}_{\tilde{\tau}_m}$. We are going to define a function $\natural_\tau$ (for order of τ is less than n) with $\tau = (\tau_1 \times \cdots \times \tau_m \to o)$ for $\mathcal{D}_{\tilde{\tau}}$. For all $x \in \mathcal{D}_{\tilde{\tau}}$ $\natural_\tau(x)$ is defined as $\natural_\tau(x)(\check{x}_1, \ldots, \check{x}_m) := \natural_o \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(x, \natural_{\tau_1}^{-1}(\check{x}_1), \ldots, \natural_{\tau_m}^{-1}(\check{x}_m))$ for all $\check{x}_1 \in \check{\mathcal{D}}_{\tau_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\tau_m}$. Analogously to case 2 we get the bijectivity of $\natural_\tau$ by the corresponding formula in Ξ_α^p.

4. Let $\natural_{\tau_i}$ be defined for $\mathcal{D}_{\tilde{\tau}_1}, \ldots, \mathcal{D}_{\tilde{\tau}_m}$. We define a function $\natural_\tau$ (for order of τ is equal to n) with $\tau = (\tau_1 \times \cdots \times \tau_m \to o)$ for $\mathcal{D}_{\tilde{\tau}}$. For all $p \in \mathcal{D}_{\tilde{\tau}}$ $\natural_\tau(p)$ is defined as $\natural_\tau(p)(\check{x}_1, \ldots, \check{x}_m) := \natural_o p(\natural_{\tau_1}^{-1}(\check{x}_1), \ldots, \natural_{\tau_m}^{-1}(\check{x}_m))$ for all $\check{x}_1 \in \check{\mathcal{D}}_{\tau_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\tau_m}$. The bijectivity of $\natural_\tau$ follows trivially.

Hence we have defined a frame $\{\check{\mathcal{D}}_\tau\}_\tau$ for all types τ.

In the following we use $\natural$ as the polymorphic mapping defined by all the individual $\natural_\tau$.

Step 3:

In this step we define an interpretation mapping $\check{\mathcal{J}}$ in order to complete the definition of an interpretation $\langle \{\check{\mathcal{D}}_\tau\}_\tau, \check{\mathcal{J}} \rangle$. For all constants c we define $\check{\mathcal{J}}(c) := \natural \circ \mathcal{J} \circ \Psi(c)$.

Step 4:

In this step we show that for every assignment $\check{\xi}$ in $\check{\mathcal{M}}$ there is an assignment ξ in $\mathcal{M}$, so that for all terms (and hence all formulae) t we have: $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t) = \natural \circ \mathcal{V}_\xi^{\mathcal{M}} \circ \Psi(t)$. Let $\check{\xi}$ be an arbitrary assignment in $\check{\mathcal{M}}$. We define ξ as $\xi = \natural^{-1} \circ \check{\xi} \circ \Psi^{-1}$. Now we prove $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}} = \natural \circ \mathcal{V}_\xi^{\mathcal{M}} \circ \Psi$ by induction on the construction of terms and formulae. Let ξ be an arbitrary assignment in $\mathcal{M}$, we get:

- T1 For all variables x_τ, $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(x_\tau) = \check{\xi}(x_\tau) = \natural(\xi(\Psi(x_\tau))) = \natural \mathcal{V}_\xi^{\mathcal{M}} \Psi(x_\tau)$.
- T2 For all constants c_τ, $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(c_\tau) = \check{\mathcal{J}}(c_\tau) = \natural(\mathcal{J}(\Psi(c_\tau))) = \natural \mathcal{V}_\xi^{\mathcal{M}} \Psi(c_\tau)$.
- T3 For all composed terms beginning with an m-ary function term f we have: $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(f_\tau(t_1, \ldots, t_m)) = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(f)(\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_1), \ldots, \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_m)) \stackrel{\text{Ind.hyp}}{=}$

$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(f)(\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m)) \overset{\text{def }\natural\text{ case 2.}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(\mathcal{V}_\xi^{\mathcal{M}}\Psi(f),\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m))] \overset{\text{def }\mathcal{V}_\xi^{\mathcal{M}}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(f),\Psi(t_1),\ldots,\Psi(t_m)))] \overset{\text{def }\Psi}{=}$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(f(t_1,\ldots,t_m))$.

For formulae we get:

F1 For an atomic formula that starts with a predicate term p of order less than n we have:
$\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(p_\tau(t_1,\ldots,t_m)) = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(p)(\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_1),\ldots,\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_m)) \overset{\text{Ind.hyp}}{=}$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(p)(\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m)) \overset{\text{def }\natural\text{ case 3.}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}})(\mathcal{V}_\xi^{\mathcal{M}}\Psi(p),\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m))] \overset{\text{def }\mathcal{V}_\xi^{\mathcal{M}}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(p),\Psi(t_1),\ldots,\Psi(t_m)))] \overset{\text{def }\Psi}{=}$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(p(t_1,\ldots,t_m))$.

F2 For an atomic formula that starts with a predicate constant p of order n we have:
$\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(p_\tau(t_1,\ldots,t_m)) = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(p)(\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_1),\ldots,\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t_m)) \overset{\text{Ind.hyp}}{=}$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(p)(\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m)) \overset{\text{def }\natural\text{ case 4.}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}\Psi(p)(\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_1),\ldots,\natural^{-1}\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(t_m))] \overset{\text{def }\mathcal{V}_\xi^{\mathcal{M}}}{=}$
$\natural[\mathcal{V}_\xi^{\mathcal{M}}(\Psi(p)(\Psi(t_1),\ldots,\Psi(t_m)))] \overset{\text{def }\Psi}{=}$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(p(t_1,\ldots,t_m))$.

F3 For a conjunction we have:
$\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi_1\wedge\varphi_2) = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi_1)\wedge\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi_2) \overset{\text{Ind.hyp}}{=} \natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\varphi_1)\wedge\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\varphi_2) =$
$\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\varphi_1\wedge\varphi_2)$.

F4 For a negation we have:
$\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\neg\varphi) = \neg\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi) \overset{\text{Ind.hyp}}{=} \neg\natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\varphi) = \natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\neg\varphi)$.

F5 For a quantification we have:
$\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\forall x_\tau\varphi) = \forall d\in\check{\mathcal{D}}_\tau\,\mathcal{V}_{\check{\xi}[x\leftarrow d]}^{\check{\mathcal{M}}}(\varphi) \overset{\text{Ind.hyp}}{=} \forall d\in\check{\mathcal{D}}_\tau\ \natural\mathcal{V}_{\xi[\Psi(x)\leftarrow\natural^{-1}(d)]}^{\mathcal{M}}\Psi(\varphi) =$
$\forall d\in\mathcal{D}_{\tilde{\tau}}\ \natural\mathcal{V}_{\xi[\Psi(x)\leftarrow d]}^{\mathcal{M}}\Psi(\varphi) = \natural\mathcal{V}_\xi^{\mathcal{M}}\Psi(\forall x\varphi)$.
Here we use that $\check{\xi} = \natural\circ\xi\circ\Psi$ for all assignments and hence $\check{\xi}[x_\tau\leftarrow d] = \natural\circ\xi[\Psi(x_\tau)\leftarrow\natural^{-1}(d)]\circ\Psi$.

<u>Step 5:</u>

Now we show that if $\mathcal{M}$ is a model of $\Phi(\varphi)$, then $\check{M}$ is a model of φ. If $\mathcal{M}$ is model of $\Phi(\varphi)$, then $\mathcal{M}$ is a model of $\Psi(\varphi)$. Let $\check{\xi}$ be an arbitrary assignment and ξ be defined as above, then we have $\mathcal{V}_\xi^{\mathcal{M}}(\Psi(\varphi)) = \mathtt{T}$, because $\mathcal{M}$ is a model of $\Psi(\varphi)$. Hence we have $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi) = \natural(\mathcal{V}_\xi^{\mathcal{M}}(\Psi(\varphi))) = \mathtt{T}$. Recall that for truth values $\natural$ is the identity function. ∎

5.24 Remark: For $n > 1$ there is no sound morphism Θ from $\mathcal{L}^n$ to $\mathcal{L}^1_{\Lambda,\equiv}$ which is strongly complete. If there were such a morphism, it would provide a complete calculus for $\mathcal{L}^n$ which is impossible because of GÖDEL'S incompleteness theorem.

5.25 Remark: As already noticed in remark 5.17, Ψ^{-1} provides a calculus for $\mathcal{L}^n$. If we add rules that enforce that function symbols and predicate symbols are equal if they agree in all arguments, we can transform every sound and complete first-order calculus of $\mathcal{L}^1_{\Lambda,\equiv}$ by Φ to a sound and weakly complete calculus for $\mathcal{L}^n$. We can execute the proof in $\mathcal{L}^1_{\Lambda,\equiv}$ and then lift it to a proof in $\mathcal{L}^n$.

5.26 Remark: One might wonder why we proposed a sufficient criterion for the soundness of translations, when we have a translation that is sound and complete and hence could always be used. The reason is, that in a concrete situation it may be better not to translate into the full sound and complete formulae, because the search space for an automated theorem prover may become too big. In general it is not a good idea to add the extensionality axioms if they are not really needed. Furthermore we prevent instantiation if we translate certain constants not by an *apply*; also we may use different *apply* functions or predicates although we could use the same. The completeness result guarantees that we can find a translation; however, which one we choose may be very important for the actual performance of a (first-order) theorem prover. That is, the flexibility in translating is very important for *practical* purpose, although *in theory* it does not enlarge the power of the method.

5.2.3 Equality

In this section we discuss a possible extension of the soundness criterion and of the morphisms Φ_n to morphisms $\Phi_{\equiv,n}$, which are mappings form $\mathcal{L}^n_\equiv$ to $\mathcal{L}^1_{\Lambda,\equiv}$. As usual we fix n and drop the corresponding index. We show that $\Phi_\equiv$ is sound and weakly complete. In the following we write ς for $(\tau \times \tau \to o)$.

5.27 Definition (Equality Quasi-Homomorphism): We replace the part for composed terms in definition 5.7 by: if $f(t_1, \ldots, t_m)$ is a term of $\mathcal{F}_1(\mathcal{S}_1)$, then $\Theta\left(f(t_1, \ldots, t_m)\right) = \theta\left(\Theta(f), \Theta(t_1), \ldots, \Theta(t_m)\right)$ with

$$\theta(a, a_1, \ldots, a_m) = \begin{cases} a(a_1, \ldots, a_m) & \text{or} \\ \boldsymbol{\alpha}^a(a, a_1, \ldots, a_m) \end{cases}$$

The $\boldsymbol{\alpha}$ have to be chosen appropriately, they have to be *new*, that is, there must be no element e such that $\boldsymbol{\alpha}^a = \Theta(e)$. The choice of the above depends only on the type of a. The symbols $\boldsymbol{\alpha}^a$ must respect the corresponding sorts. *If the first case is chosen for equality, this must be mapped on equality. Furthermore we require that* <u>*only*</u> *equality is mapped on equality.*

5.28 Theorem: *If* Θ *is an injective equality quasi-homomorphism from* $\mathcal{L}^n_{\equiv}(\mathcal{S})$ *to* $\mathcal{L}^1_{\Lambda,\equiv}(\mathcal{S}')$, *then* Θ *is weakly sound.*

Proof: The proof is analogous to the proof of theorem 5.12. We only add the following cases to the proof in step 4 (analogously to the cases T3, T4):

- For an atomic formula with the equality symbol as top symbol that is mapped on the equality predicate:
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_1 \equiv t_2)) := \mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_1) \equiv \Theta(t_2)) =$
$(\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_1)) \equiv_{\mathcal{D}_{\hat{\tau}}} \mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_2))) \overset{\text{Ind.hyp}}{=} (\mathcal{V}^{\mathcal{M}}_{\xi}(t_1) \equiv_{\mathcal{D}_\tau} \mathcal{V}^{\mathcal{M}}_{\xi}(t_2)) = \mathcal{V}^{\mathcal{M}}_{\xi}(t_1 \equiv t_2)$
- For an atomic formula with an equality symbol as top symbol that is not mapped on the equality predicate we have:
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_1 \equiv t_2)) := \mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\boldsymbol{\alpha}^{\bar{\varsigma}}(\Theta(\equiv), \Theta(t_1), \Theta(t_2))) =$
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\boldsymbol{\alpha}^{\bar{\varsigma}})(\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(\equiv)), \mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_1)), \mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\Theta(t_2))) \overset{\text{Ind.hyp}}{=}$
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\widehat{\xi}}(\boldsymbol{\alpha}^{\bar{\varsigma}})(\mathcal{V}^{\mathcal{M}}_{\xi}(\equiv), \mathcal{V}^{\mathcal{M}}_{\xi}(t_1), \mathcal{V}^{\mathcal{M}}_{\xi}(t_2)) \overset{\text{def}}{=} (\mathcal{V}^{\mathcal{M}}_{\xi}(t_1) \equiv_{\mathcal{D}_\tau} \mathcal{V}^{\mathcal{M}}_{\xi}(t_2)) =$
$\mathcal{V}^{\mathcal{M}}_{\xi}(t_1 \equiv t_2)$ ■

5.29 Example: We shall use example 5.14, however in a formulation with equality and translate it then in the usual way. (In order to show that both representations are *equivalent* it would be necessary to show that there is a sound and complete morphism that maps them to one another.) We drop the type information for readability, *group* is of type $(\iota \times (\iota \times \iota \to \iota) \to o)$, G of type ι, $+$ of type $(\iota \times \iota \to \iota)$, $-$ of type $(\iota \to \iota)$, and so on. Also for readability we sometimes use infix notation. In the target the sorts are ("ι", "$(\iota \times \iota \to \iota)$") for *group*, and so on. A group can be defined as follows:

1. $\forall G, + \;\; group(G,+) \iff associative(G,+) \wedge$
$\qquad \exists 0 \;\; (0 \in G \;\wedge\; neutral\_element(G,+,0) \wedge$
$\qquad\quad \exists - \;\; inverse(G,+,0,-))$
2. $\forall G, + \;\; associative(G,+) \iff \forall x,y,z \;\; x,y,z \in G \Longrightarrow (x+y)+z \equiv x+(y+z)$
3. $\forall G,+,0 \;\; neutral\_element(G,+,0) \iff \forall x \;\; x \in G \Longrightarrow x{+}0 \equiv x \wedge 0{+}x \equiv x$

4. $\forall G, +, 0, -\ \ inverse(G, +, 0, -) \iff \forall x\ \ x \in G \Longrightarrow x + (-x) \equiv 0\ \wedge$
$(-x) + x \equiv 0$

This formula set is a subset of $\mathcal{L}^3_{\equiv}$. Now we give a translation into a formula set of $\mathcal{L}^1_{\Lambda,\equiv}$. The signatures are obvious, hence omitted. The translation is sound, because it is an injective equality quasi-homomorphism.

1. $\forall G, +\ \ group(G, +) \iff associative(G, +)\ \wedge$
$\exists 0\ \ (0 \in G\ \wedge\ neutral\_element(G, +, 0)\ \wedge$
$\exists -\ \ inverse(G, +, 0, -))$

2. $\forall G, +\ \ associative(G, +) \iff \forall x, y, z\ \ x, y, z \in G \Longrightarrow$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, \boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, x, y), z)\ \equiv$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, x, \boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, y, z))$

3. $\forall G, +, 0\ \ neutral\_element(G, +, 0) \iff \forall x\ \ x \in G \Longrightarrow$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, x, 0)\ \equiv\ x\ \wedge$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, 0, x)\ \equiv\ x$

4. $\forall G, +, 0, -\ \ inverse(G, +, 0, -) \iff \forall x\ \ x \in G \Longrightarrow$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, x, \boldsymbol{\alpha}^{"(\iota\to\iota)"}(-, x))\ \equiv\ 0\ \wedge$
$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+, \boldsymbol{\alpha}^{"(\iota\to\iota)"}(-, x), x)\ \equiv\ 0$

5.30 Definition (Standard Translation $\Phi_{\equiv}$):

- At first we define the mapping on the signature. We proceed as in definition 5.18, but add for each $\equiv_\varsigma$ in $\mathcal{S}_\varsigma$ of order less than n an object-constant symbol $\hat{\equiv}_{\tilde{\tau}}$ to $\mathcal{S}_{\tilde{\varsigma}}$. We cannot name it $\equiv_{(\tilde{\tau}\times\tilde{\tau}\to\tilde{o})}$ because this is already defined as a binary predicate symbol. In addition we have the term declarations ($\equiv_{\tilde{\varsigma}}$: $(\tilde{\tau} \times \tilde{\tau} \to \tilde{o})$) for every τ with order of ς is equal to n and ($\hat{\equiv}_{\tilde{\tau}} : \tilde{\varsigma}$) for every τ with order of ς less than n.
- The inductive definition of $\Psi_{\equiv}(\Gamma)$ is the definition of $\Psi(\Gamma)$ in definition 5.18 plus
  $\Psi_{\equiv}(\equiv_\varsigma) = \equiv$ for order of ς equal to n and $\Psi_{\equiv}(\equiv_\varsigma) = \hat{\equiv}_{\tilde{\varsigma}}$ for order of ς less than n. In addition we have:
  If t_1 and t_2 are terms of type τ with $\tau \neq o$ and order of ς is equal to n, then $\Psi_{\equiv}(t_1 \equiv t_2) = (\Psi_{\equiv}(t_1) \equiv \Psi_{\equiv}(t_2))$. This term is well-sorted, because $\Psi_{\equiv}(t_i)$ are both of sort $\tilde{\tau}$.
  If t_1 and t_2 are terms of type τ with $\tau \neq o$ and order of ς less than n, then $\Psi_{\equiv}(t_1 \equiv t_2) = \boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, \Psi_{\equiv}(t_1), \Psi_{\equiv}(t_2))$. This term is again well-sorted, because $\Psi_{\equiv}(t_i)$ are both of sort $\tilde{\tau}$ and $\hat{\equiv}_{\tilde{\varsigma}}$ is of type $\tilde{\varsigma}$.

- $\Xi_\equiv$ is defined as Ξ_α plus the set of all formulae (with order of ς less than n):
  $\forall x_{\tilde{\tau}} \forall y_{\tilde{\tau}} \;\; \boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y) \Longrightarrow x \equiv y$.

- Like above on formula sets we define $\Phi_\equiv(\Gamma) := \Psi_\equiv(\Gamma) \cup \Xi_\equiv$.

5.31 Theorem: $\Phi_\equiv$ *is weakly sound.*

Proof: As above we have that $\Psi_\equiv$ is sound, because it is an injective equality quasi-homomorphism. We still have to show that every formula in $\Xi_\equiv$ is satisfied, that is, it remains to be shown:
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}}(\forall x_{\tilde{\tau}} \forall y_{\tilde{\tau}} \boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y) \Longrightarrow x \equiv y) = \mathtt{T}$
Therefore it is sufficient to show that for all $X, Y \in \mathcal{D}_{\tilde{\tau}}$
$\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[x,y \leftarrow X,Y]}(\boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y) \Longrightarrow x \equiv y) = \mathtt{T}$.
By the definitions of $\boldsymbol{\alpha}^{\tilde{\varsigma}}$ and $\mathcal{V}^{\widehat{\mathcal{M}}}_{\hat{\xi}[x,y \leftarrow X,Y]}$ that is equivalent to
$X \equiv_{\mathcal{D}_\tau} Y \Longrightarrow X \equiv_{\mathcal{D}_\tau} Y$, which is obviously true. ∎

5.32 Theorem: $\Phi_\equiv$ *is weakly complete.*

Proof: In the completeness theorem 5.23 we have to add to the proof in step 4:

- For equalities of order equal to n:
  $\mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_1 \equiv t_2) := (\mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_1) \equiv_{\check{\mathcal{D}}_\tau} \mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_2)) \overset{\text{Ind.hyp}}{=} (\natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi(t_1) \equiv_{\check{\mathcal{D}}_\tau} \natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi(t_2)) =$
  $(\natural \mathcal{V}^{\mathcal{M}}_{\xi}(\Psi(t_1) \equiv \natural \Psi(t_2)) = \natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi(t_1 \equiv t_2)$.

- For the equalities of order less than n we use at first the additional axioms in $\Xi_\equiv$:
  $\forall x_{\tilde{\tau}} \forall y_{\tilde{\tau}} \boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y) \Longrightarrow x \equiv y$.
  Hence we have in a model for all X, Y in $\mathcal{D}_{\tilde{\tau}}$:
  $\mathcal{V}^{\mathcal{M}}_{\xi[x,y \leftarrow X,Y]}(\boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y)) \Longrightarrow X \equiv_{\mathcal{D}_{\tilde{\tau}}} Y$
  Since the direction $\Longleftarrow$ is trivially satisfied, we have:
  $\mathcal{V}^{\mathcal{M}}_{\xi[x,y \leftarrow X,Y]}(\boldsymbol{\alpha}^{\tilde{\tau}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y)) = (X \equiv_{\mathcal{D}_{\tilde{\tau}}} Y) \qquad (*)$
  Now we can prove:
  $\mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_1 \equiv t_2) = (\mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_1) \equiv_{\check{\mathcal{D}}_\tau} \mathcal{V}^{\check{\mathcal{M}}}_{\check{\xi}}(t_2)) \overset{\text{Ind.hyp}}{=}$
  $(\natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_1) \equiv_{\check{\mathcal{D}}_\tau} \natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_2)) \overset{\natural\text{ bij}}{=} (\mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_1) \equiv_{\mathcal{D}_{\tilde{\tau}}} \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_2)) \overset{(*)}{=}$
  $\mathcal{V}^{\mathcal{M}}_{\xi[x,y \leftarrow \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_1), \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_2)]}(\boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, x, y)) = \mathcal{V}^{\mathcal{M}}_{\xi}(\boldsymbol{\alpha}^{\tilde{\varsigma}}(\hat{\equiv}_{\tilde{\varsigma}}, \Psi_\equiv(t_1), \Psi_\equiv(t_2))) =$
  $\mathcal{V}^{\mathcal{M}}_{\xi}(\boldsymbol{\alpha}^{\tilde{\varsigma}}(\Psi_\equiv(\equiv_\varsigma), \Psi_\equiv(t_1), \Psi_\equiv(t_2))) \overset{\natural\text{ bij}}{=} \natural \mathcal{V}^{\mathcal{M}}_{\xi} \Psi_\equiv(t_1 \equiv t_2)$ ∎

5.33 Remark: We do not translate $\equiv_\varsigma$ immediately to $\equiv_{(\tilde{\tau},\tilde{\tau})}$, because then it could not become the argument of a higher-order predicate and we would also lose completeness. Consider the case of the following induction schema:

$$\forall P_{(\iota\times\iota\to o)}(P(0,0)\wedge$$
$$(\forall n\;\; P(n,0) \Longrightarrow P(s(n),0)) \wedge (\forall n,m\;\; P(n,m) \Longrightarrow P(n,s(m))) \Longrightarrow$$
$$\forall n,m\;\; P(n,m)),$$

where in addition we have the formulae $0 \equiv 0$, $\forall n \;\; n \equiv 0 \Longrightarrow s(n) \equiv 0$, and $\forall n,m \;\; n \equiv m \Longrightarrow n \equiv s(m)$. If we want to prove $\forall n,m \;\; n \equiv m$ we have to instantiate the predicate variable P in the induction schema by the equality predicate. But if we translate P by an object variable and $\equiv$ by a predicate constant we cannot instantiate P by $\equiv$ in the first-order target formulation.

5.34 Example: We translate the examples 5.14 and 5.29 from above. Using $\Psi_{\equiv,3}$ this is translated to:

1. $\forall G,+ \;\; \mathit{group}(G,+) \iff \mathit{associative}(G,+) \wedge$
$$\exists 0 \;\; (0 \in G \;\wedge\; \mathit{neutral\_element}(G,+,0) \wedge$$
$$\exists - \;\; \mathit{inverse}(G,+,0,-))$$

2. $\forall G,+ \;\; \mathit{associative}(G,+) \iff$
$$\forall x,y,z \;\; \boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\in,x,G) \wedge \;\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\in,y,G) \wedge \;\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\in,z,G)$$
$$\Longrightarrow \boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\hat{\equiv},\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,x,y),z),$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,x,\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,y,z)))$$

3. $\forall G,+,0 \;\; \mathit{neutral\_element}(G,+,0) \iff$
$$\forall x \;\; \boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\in,x,G) \Longrightarrow$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\hat{\equiv},\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,x,0),x)\wedge$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\hat{\equiv},\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,0,x),x)$$

4. $\forall G,+,0,- \;\; \mathit{inverse}(G,+,0,-) \iff$
$$\forall x \;\; \boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\in,x,G) \Longrightarrow$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\hat{\equiv},\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,x,\boldsymbol{\alpha}^{"(\iota\to\iota)"}(-,x)),0) \wedge$$
$$\boldsymbol{\alpha}^{"(\iota\times\iota\to o)"}(\hat{\equiv},\boldsymbol{\alpha}^{"(\iota\times\iota\to\iota)"}(+,\boldsymbol{\alpha}^{"(\iota\to\iota)"}(-,x),x),0)$$

Of course this translation is more complicated than that of example 5.29.

5.3 Translations of Higher-Order Sorted Logic

In this section we sketch how to extend the results of the previous section to *sorted* higher-order logics. This could be done by essentially copying the proofs of the unsorted case. But instead we are going to reduce the soundness theorems to the corresponding unsorted theorems by using relativizations, which can be done since the translations are structure conserving. The advantage of the proofs via relativizations is that they can be used for other kinds of sort structures too, especially they are easy to transfer to sorted logics where the semantics is defined by the relativization. At first we define relativizations and show them to be sound and complete morphisms. The completeness proof for the standard translation is not lifted, but worked out directly.

5.3.1 Relativizations and Partial Relativizations

For a formula set of sorted logics it is in general possible to state an equivalent formula set of an unsorted logic. In this section we will introduce relativizations and partial relativizations for the logics $\mathcal{L}^n_\Sigma$ to $\mathcal{L}^n$ and $\mathcal{L}^1_\Sigma$ to $\mathcal{L}^1_\Lambda$.

5.35 Definition (Relativization): The *relativization* $\Re$ from $\mathcal{L}^n_\Sigma$ to $\mathcal{L}^n$ is a morphism of the form that:

1. the signature $\mathcal{S}_\Sigma = (\mathcal{S}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$ is mapped to $(\mathcal{S} \cup \Sigma \backslash \{\tau | \tau \text{ is type}\}, \{\tau | \tau \text{ is type}\}, x \mapsto \text{type}(x), \emptyset, \emptyset)$, where the sort symbols κ of type τ are mapped onto unary predicate constants of type $(\tau \to o)$.
2. A formula φ is mapped to the formula set $\Re(\varphi)$ consisting of:
   $\{\widehat{\Re}(\varphi)\} \cup$
   $\{\forall x_\tau \;\; \kappa(x) \Longrightarrow \mu(x) \mid \kappa \dot{\sqsubseteq} \mu \text{ with } \text{type}(\kappa) = \text{type}(\mu) = \tau\} \cup$
   $\{\kappa(\widehat{\Re}(t)) \mid (t : \kappa) \in \boldsymbol{\delta}\}$, where

   $\widehat{\Re}$ is defined as:

   (a) For terms $\widehat{\Re}(t) = t$ and for atomic formulae: $\widehat{\Re}(\varphi) = \varphi$
   (b) For conjunctions and negations: $\widehat{\Re}(\varphi \wedge \psi) = \widehat{\Re}(\varphi) \wedge \widehat{\Re}(\psi)$ and $\widehat{\Re}(\neg\varphi) = \neg\widehat{\Re}(\varphi)$.
   (c) For a quantification over a variable x of sort κ with $\text{type}(\kappa) = \tau$, $\widehat{\Re}(\forall x_\kappa \; \varphi) = \forall x_\tau \; \kappa(x) \Longrightarrow \widehat{\Re}(\varphi)$.

For formula sets Γ we have as usual $\Re(\Gamma) = \bigcup_{\varphi \in \Gamma} \Re(\varphi)$.

5.36 Definition (Partial Relativization): The *partial relativization* $\partial\Re$ from $\mathcal{L}^1_\Sigma$, where every sort (except ι) has a unique upper sort (shortly called "uus") immediately below ι, to $\mathcal{L}^1_\Lambda$ is a morphism defined in the following form. We have for all sorts κ, $\kappa \dot{\sqsubseteq} \text{uus}(\kappa)$ (we define $\text{uus}(\iota) = \iota$).

- the signature $\mathcal{S}_\Sigma = (\mathcal{S}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$ is mapped to $(\partial\Re(\mathcal{S}), \partial\Re(\Sigma), \partial\Re(\mathfrak{s}), \partial\Re(\sqsubseteq), \partial\Re(\boldsymbol{\delta}))$ with:

  1. $\partial\Re(\mathcal{S}) = \mathcal{S} \cup \Sigma \backslash \{\tau \mid \tau = \mathrm{uus}(\tau)\}$ where these sort symbols are now new unary predicate constants of sort $(\iota \to o)$.
  2. $\partial\Re(\Sigma) = \{\tau \mid \tau = \mathrm{uus}(\tau)\}$
  3. $\partial\Re(\mathfrak{s}) = x \mapsto \mathrm{uus}(\mathfrak{s}(x))$
  4. $\partial\Re(\sqsubseteq) = \{\mathrm{uus}(\kappa) \sqsubseteq \iota \mid \kappa\}$
  5. $\partial\Re(\boldsymbol{\delta}) = \{(\partial\Re(t) : \mathrm{uus}(\kappa)) \mid (t : \kappa) \in \boldsymbol{\delta}\}$

- A formula φ is mapped to the formula set $\partial\Re(\varphi)$ consisting of
  $\{\partial\widehat{\Re}(\varphi)\} \cup$
  $\{\forall x_{\mathrm{uus}(\kappa_2)}\ \ \kappa_1(x) \Longrightarrow \kappa_2(x) \mid \kappa_1 \dot{\sqsubseteq} \kappa_2 \text{ with } \kappa_1 \neq \mathrm{uus}(\kappa_1) = \mathrm{uus}(\kappa_2)\} \cup$
  $\{\kappa(\partial\widehat{\Re}(t)) \mid (t : \kappa) \in \boldsymbol{\delta}\}$, where

  $\partial\widehat{\Re}$ is defined as:

  1. For terms $\partial\widehat{\Re}(t) = t$ and for atomic formulae: $\partial\widehat{\Re}(\varphi) = \varphi$
  2. For conjunctions and negations:
     $\partial\widehat{\Re}(\varphi \wedge \psi) = \partial\widehat{\Re}(\varphi) \wedge \partial\widehat{\Re}(\psi)$ and
     $\partial\widehat{\Re}(\neg\varphi) = \neg\partial\widehat{\Re}(\varphi)$.
  3. For a quantification over a variable x of sort κ:
     $\partial\widehat{\Re}(\forall x_\kappa \varphi) = (\forall x_{\mathrm{uus}(\kappa)} \kappa(x) \Longrightarrow \partial\widehat{\Re}(\varphi))$.

5.37 Theorem: *The relativizations $\Re$ form $\mathcal{L}^n_\Sigma$ to $\mathcal{L}^n$ are sound and complete.*

Proof: Let Γ be a formula set in $\mathcal{L}^n_\Sigma$. Analogously to the proofs above we can show that if there is a model of Γ we can construct a model of $\Re(\Gamma)$ and vice versa.

1. Soundness: Let us assume that there is a model $\mathcal{M} = \langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J}\rangle$ of Γ. We define an interpretation $\dot{\mathcal{M}} = \langle \{\dot{\mathcal{D}}_\tau\}_\tau, \dot{\mathcal{J}}\rangle$ of $\Re(\Gamma)$ by:

- $\dot{\mathcal{D}}_\tau := \bigcup_{\mathrm{type}(\kappa)=\tau} \mathcal{D}_\kappa$,
- $\dot{\mathcal{J}}(\widehat{\Re}(c)) := \mathcal{J}(c)$
- For the "new" constants $\widehat{\Re}(\kappa)$ of $\mathcal{L}^n$ we define the predicate $\dot{\mathcal{J}}(\widehat{\Re}(\kappa))$ by $\dot{\mathcal{J}}(\widehat{\Re}(\kappa))(x) := (x \in \mathcal{D}_\kappa)$ for all x in $\dot{\mathcal{D}}_\tau$.

The proof that $\dot{\mathcal{M}}$ is a model of $\Re(\Gamma)$ can be done by showing inductively on the construction of formulae that $\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}} \circ \widehat{\Re} = \mathcal{V}^{\mathcal{M}}_{\xi}$. The only interesting part is that of quantifications:

$\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\widehat{\Re}(\forall x_\kappa \varphi)) = \mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\forall x_\tau\ \ \kappa(x) \Longrightarrow \widehat{\Re}(\varphi)) =$
$\forall d \in \dot{\mathcal{D}}_\tau\ \ \mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}[x_\tau \leftarrow d]}(\kappa(x) \Longrightarrow \widehat{\Re}(\varphi)) =$
$\forall d \in \dot{\mathcal{D}}_\tau\ \ \dot{\mathcal{J}}(\widehat{\Re}(\kappa))(d) \Longrightarrow \mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}[x_\tau \leftarrow d]}(\Re(\varphi)) =$
$\forall d \in \dot{\mathcal{D}}_\tau\ \ d \in \mathcal{D}_\kappa \Longrightarrow \mathcal{V}^{\mathcal{M}}_{\xi[x_\kappa \leftarrow d]}(\varphi) =$
$\forall d \in \mathcal{D}_\kappa\ \ \mathcal{V}^{\mathcal{M}}_{\xi[x_\kappa \leftarrow d]}(\varphi) =$
$\mathcal{V}^{\mathcal{M}}_{\xi}(\forall x_\kappa \varphi)$.

Furthermore we have to show that for all sorts κ, μ of type τ with $\kappa \dot{\sqsubseteq} \mu$:
$\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\forall x_\tau\ \ \kappa(x) \Longrightarrow \mu(x)) = \mathtt{T}$. This holds since:
$\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\forall x_\tau\ \ \kappa(x) \Longrightarrow \mu(x)) = (\forall d \in \mathcal{D}_\tau\ \ \mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}[x \leftarrow d]}(\kappa(x) \Longrightarrow \mu(x))) =$
$(\forall d \in \mathcal{D}_\tau\ \ \dot{\mathcal{J}}(\widehat{\Re}(\kappa))(d) \Longrightarrow \dot{\mathcal{J}}(\widehat{\Re}(\mu))(d)) =$
$(\forall d \in \mathcal{D}_\tau\ \ d \in \mathcal{D}_\kappa \Longrightarrow d \in \mathcal{D}_\mu)$ and this holds since $\mathcal{D}_\kappa \subseteq \mathcal{D}_\mu$.

At last we have to show that for all term declarations $(t : \kappa) \in \boldsymbol{\delta}$: $\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\kappa(\widehat{\Re}(t))) = \mathtt{T}$. This holds since:
$\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\kappa(\widehat{\Re}(t))) = \mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}}(\kappa)(\mathcal{V}^{\dot{\mathcal{M}}}_{\dot{\xi}} \circ \widehat{\Re}(t)) =$
$\dot{\mathcal{J}}(\widehat{\Re}(\kappa))(\mathcal{V}^{\mathcal{M}}_{\xi}(t)) = (\mathcal{V}^{\mathcal{M}}_{\xi}(t) \in \mathcal{D}_\kappa)$, and this holds since $(t : \kappa)$.

2. Completeness: Let us assume that there is a model $\mathcal{M} = \langle \{D_\tau\}_\tau, \mathcal{J} \rangle$ of $\Re(\Gamma)$. We define a model $\breve{\mathcal{M}} = \langle \{\breve{\mathcal{D}}_\kappa\}_\kappa, \breve{\mathcal{J}} \rangle$ of Γ by:

- $\breve{\mathcal{D}}_\kappa := \{d \in \mathcal{D}_\tau | \mathcal{J}(\widehat{\Re}(\kappa))(d)\}$
- $\breve{\mathcal{J}}(c) := \mathcal{J}(\widehat{\Re}(c))$

The proof that $\breve{\mathcal{M}}$ is a model of Γ can be done by showing inductively on the construction of formulae that $\mathcal{V}^{\breve{\mathcal{M}}}_{\breve{\xi}} = \mathcal{V}^{\mathcal{M}}_{\xi} \circ \widehat{\Re}$. Again, the only interesting part is that of quantification:

$\mathcal{V}^{\breve{\mathcal{M}}}_{\breve{\xi}}(\forall x_\kappa \varphi) = \forall d \in \breve{\mathcal{D}}_\kappa\ \ \mathcal{V}^{\breve{\mathcal{M}}}_{\breve{\xi}[x \leftarrow d]}(\varphi) =$
$\forall d \in \mathcal{D}_\tau\ \ \mathcal{J}(\widehat{\Re}(\kappa))(d) \Longrightarrow \mathcal{V}^{\mathcal{M}}_{\xi[x \leftarrow d]}(\widehat{\Re}(\varphi)) =$
$\forall d \in \mathcal{D}_\tau\ \ \mathcal{V}^{\mathcal{M}}_{\xi[x \leftarrow d]}(\widehat{\Re}(\kappa)(x) \Longrightarrow \widehat{\Re}(\varphi)) =$
$\mathcal{V}^{\mathcal{M}}_{\xi}(\forall x_\tau \widehat{\Re}(\kappa)(x) \Longrightarrow \widehat{\Re}(\varphi)) =$
$\mathcal{V}^{\mathcal{M}}_{\xi}(\widehat{\Re}(\forall x_\kappa \varphi))$.

Furthermore we have to show that for all sorts κ, μ of type τ with $\kappa \dot{\sqsubseteq} \mu$: $\breve{\mathcal{D}}_\kappa \subseteq \breve{\mathcal{D}}_\mu$. This holds since it is equivalent to:
$(\forall d \in \mathcal{D}_\tau\ \ \mathcal{J}(\widehat{\Re}(\kappa))(d) \Longrightarrow \mathcal{J}(\widehat{\Re}(\mu))(d)) =$
$\mathcal{V}^{\mathcal{M}}_{\xi}(\forall x_\tau \kappa(x) \Longrightarrow \mu(x))$, what holds by definition of $\Re(\Gamma)$.

Analogously we can show that for all term declarations $(t : \kappa) \in \boldsymbol{\delta}$: $\mathcal{V}_\xi^{\breve{\mathcal{M}}}(t) \in \breve{\mathcal{D}}_\kappa$: Let $(t : \kappa) \in \boldsymbol{\delta}$, then since $\mathcal{M}$ is a model of $\Re(\Gamma)$ we have $\mathcal{V}_\xi^{\mathcal{M}}(\kappa(\widehat{\Re}(t)))$, what is equal to $\mathcal{J}(\widehat{\Re}(\kappa))(\mathcal{V}_\xi^{\mathcal{M}}(\widehat{\Re}(t)))$. Hence we have by definition of $\breve{\mathcal{D}}_\kappa$: $\mathcal{V}_\xi^{\mathcal{M}}(\widehat{\Re}(t)) \in \breve{\mathcal{D}}_\kappa$ and so finally $\mathcal{V}_\xi^{\breve{\mathcal{M}}}(t) \in \breve{\mathcal{D}}_\kappa$. ∎

5.38 Theorem: *The partial relativizations $\partial\Re$ from $\mathcal{L}^1_\Sigma$ to $\mathcal{L}^1_\Lambda$ are sound and complete.*

Proof: The proof is analogous to the proof of theorem 5.37. ∎

5.39 Remark: The relativizations $\Re$ are injective.

5.3.2 A Sufficient Criterion for Soundness

In this section we give a sufficient criterion for the soundness of translations of formulae of $\mathcal{L}^n_\Sigma$ onto formulae of $\mathcal{L}^1_\Sigma$, which is strong enough to cover most requirements.

5.40 Theorem: *If Θ is an injective quasi-homomorphism from $\mathcal{L}^n_\Sigma(\mathcal{S}_\Sigma)$ to $\mathcal{L}^1_\Sigma(\mathcal{S}'_\Sigma)$, then Θ is weakly sound.*

Proof: We show that there is a commutative diagram:

$$\begin{tikzcd} \mathcal{L}^n_\Sigma \arrow[r, "\Re"] \arrow[d, "\Theta"] \arrow[dr, phantom, "\#"] & \mathcal{L}^n \arrow[d, "\dot{\Theta}"] \\ \mathcal{L}^1_\Sigma \arrow[r, "\partial\Re"] & \mathcal{L}^1_\Lambda \end{tikzcd}$$

with an injective quasi-homomorphism $\dot{\Theta}$ from $\mathcal{L}^n$ to $\mathcal{L}^1_\Lambda$. Since $\partial\Re$ is complete, $\Re$ is sound and $\dot{\Theta}$ is sound by theorem 5.12, we can conclude that Θ is sound. We construct $\dot{\Theta}$ out of Θ by:

1. For all terms:

   (a) if x is a variable (or constant) of $\mathcal{L}^n$ then $\dot{\Theta}(x) = \partial\Re \circ \Theta \circ \Re^{-1}(x)$ is a variable (or constant) of $\mathcal{L}^1_\Lambda$.

   (b) if $f(t_1, \ldots, t_m)$ is a term of $\mathcal{L}^n$ then $\dot{\Theta}(f(t_1, \ldots, t_m)) = \theta(\dot{\Theta}(f), \Theta(t_1), \ldots, \Theta(t_m))$ with

$$\theta(a, a_1, \ldots, a_m) = \begin{cases} a(a_1, \ldots, a_m) & \text{or} \\ \boldsymbol{\alpha}^a(a, a_1, \ldots, a_m) \end{cases}$$ The $\boldsymbol{\alpha}$ have to be chosen appropriately, especially they have to be *new*, that is, there must be no element e so that $\boldsymbol{\alpha}^a = \dot{\Theta}(e)$. The case which is chosen depends on the case that is chosen for the translation of $\Theta \circ \Re^{-1}$.

2. All predicates $\Re(\kappa)$ in $\mathcal{L}^n$ are translated to $\partial\Re\circ\Theta(\kappa)$. Every formula resulting of a term declaration or a subsort relation is translated in the same manner.

3. $\dot{\Theta}$ is the homomorphic closure of the above relation.

Obviously $\dot{\Theta}$ is a quasi-homomorphism. ∎

5.41 Example: Let us take an example from [40, p.40], theorem (4.11) (for an MKRP proof using this translation see chapter 6). The different (constant) sets, especially a set S, are introduced. The interest is in considering binary relations on S, especially the subset relation between two such relations is defined. In our sorted higher-order logic this definition can be given by:

$$\begin{tikzcd}
\begin{array}{l} \forall f{:}(S \times S \to o)\ \forall g{:}(S \times S \to o) \\ \quad (subset(f,g) \iff \\ \quad (\forall x{:}S\ \forall y{:}S\ f(x,y) \Longrightarrow g(x,y))) \end{array} \arrow[r, "\Re"] \arrow[d, "\Theta"] & \begin{array}{l} \forall f{:}(\iota \times \iota \to o)\ \text{"}(S \times S \to o)\text{"}(f) \Longrightarrow \\ (\forall g{:}(\iota \times \iota \to o)\ \text{"}(S \times S \to o)\text{"}(g) \Longrightarrow \\ \quad (subset(f,g) \iff \\ \quad (\forall x{:}\iota\ \text{"}S\text{"}(x) \Longrightarrow (\forall y{:}\iota\ \text{"}S\text{"}(y) \Longrightarrow \\ \quad\quad (f(x,y) \Longrightarrow g(x,y)))))) \end{array} \arrow[d, "\dot{\Theta}"] \\
\begin{array}{l} \forall f{:}\text{"}(S \times S \to o)\text{"}\ \forall g{:}\text{"}(S \times S \to o)\text{"} \\ \quad (subset(f,g) \iff \\ \quad (\forall x{:}\text{"}S\text{"}\ \forall y{:}\text{"}S\text{"}\ \boldsymbol{\alpha}^{\text{"}(\iota\times\iota\to o\text{")}}(f,x,y) \\ \quad\quad \Longrightarrow \boldsymbol{\alpha}^{\text{"}(\iota\times\iota\to o)\text{"}}(g,x,y))) \end{array} \arrow[r, "\partial\Re"] & \begin{array}{l} \forall f{:}\text{"}(\iota \times \iota \to o)\text{"}\ \text{"}(S \times S \to o)\text{"}(f) \Longrightarrow \\ (\forall g{:}\text{"}(\iota \times \iota \to o)\text{"}\ \text{"}(S \times S \to o)\text{"}(g) \Longrightarrow \\ \quad (subset(f,g) \iff \\ \quad (\forall x{:}\text{"}\iota\text{"}\ \text{"}S\text{"}(x) \Longrightarrow \\ \quad (\forall y{:}\text{"}\iota\text{"}\ \text{"}S\text{"}(y) \Longrightarrow \\ \quad\quad (\boldsymbol{\alpha}^{\text{"}(\iota\times\iota\to o)\text{"}}(f,x,y) \\ \quad \Longrightarrow \boldsymbol{\alpha}^{\text{"}(\iota\times\iota\to o)\text{"}}(g,x,y)))))) \end{array}
\end{tikzcd}$$

5.42 Theorem: *If Θ is an injective quasi-homomorphism from $\mathcal{L}^n_\Sigma(\mathcal{S})$ to $\mathcal{L}^1_\Sigma(\mathcal{S}')$, then Θ is strongly sound.*

Proof: If there is a strong model of a formula set Γ in $\mathcal{L}^n_\Sigma(\mathcal{S})$ then this model is also a weak model. By the previous theorem there is hence a weak model of $\Theta(\Gamma)$ in $\mathcal{L}^1_\Sigma(\mathcal{S}')$. By a sorted version of theorem 3.19 there is also a strong model of $\Theta(\Gamma)$. ∎

5.43 Remark: In the sorted case we note again that the formulae that are obtained by these translations are not essentially more difficult then the original ones and that the proofs can easily be translated back, because the mappings Θ are injective.

5.3.3 The Standard Translation From Sorted Higher-Order Logic to Sorted First-Order Logic

Now we want to define morphisms Φ^n_Σ from $\mathcal{L}^n_\Sigma$ to $\mathcal{L}^1_\Sigma$ which are not only sound but also complete. As in section 5.2.2 we define the morphisms for odd n, for even n they are obtained as the restriction of the next higher odd n, that is $\Phi^{2n}_\Sigma := \Phi^{2n+1}_\Sigma \mid_{\mathcal{L}^{2n}_\Sigma}$. The morphisms Φ are defined as $\Phi(\varphi) = \Psi(\varphi) \cup \Xi^\Sigma_\alpha$, where $\Psi(\varphi)$ is a quasi-homomorphism and Ξ^Σ_α are special extensionality axioms for the $\boldsymbol{\alpha}$. In the following we drop the index n. Again we abbreviate *apply* as $\boldsymbol{\alpha}$.

5.44 Definition (Standard Translation Φ^Σ_{2n-1}): Let $\mathcal{S}^{2n-1}_\Sigma$ be the signature of a logic in $\mathcal{L}^{2n-1}_\Sigma$. In order to define a morphism Φ to $\mathcal{L}^1_{\Sigma,\equiv}$, we have to define the signature $\widehat{\mathcal{S}}_\Sigma$ of the target logic and we have to fix how formulae are mapped.
Let $\widehat{\mathcal{S}}_\Sigma$ be equal to $(\widehat{\mathcal{S}}, \Sigma, \mathfrak{s}, \sqsubseteq, \boldsymbol{\delta})$

1. $\widehat{\mathcal{S}}$ is defined as in the unsorted case (compare definition 5.18).

2. Σ is defined as the set
   $\{\tilde{\kappa} \mid \mathrm{ord}(\kappa) < n\} \ \cup$
   $\{(\tilde{\kappa} \times \tilde{\kappa}_1 \times \cdots \times \tilde{\kappa}_m \to \tilde{\mu}) \mid \kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu) \ \wedge \ \mathrm{ord}(\kappa) < n\} \ \cup$
   $\{(\tilde{\kappa}_1 \times \cdots \times \tilde{\kappa}_m \to \tilde{o}) \mid \kappa = (\kappa_1 \times \cdots \times \kappa_m \to o) \ \wedge \ \mathrm{ord}(\kappa) = n\}$
   The function $\tilde{}$ must map κ injectively to new names. Again we can realize this function by taking the strings. Often we abbreviate $\tilde{o}$ to o.

3. $\mathfrak{s}$ is defined for variables $\Phi(x_\tau)$, where x_τ is mapped to $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$ by the corresponding $\mathfrak{s}$-function in higher-order logic, as
   $$\mathfrak{s}(\Phi(x_\tau)) = \begin{cases} \tilde{\kappa} & \text{for } \mathrm{ord}(\kappa) < n \\ (\tilde{\kappa}_1 \times \cdots \times \tilde{\kappa}_m \to \tilde{\mu}) & \text{for } \mathrm{ord}(\kappa) = n \end{cases}$$

4. $\sqsubseteq$ is defined as

   (a) $\tilde{\kappa} \sqsubseteq \tilde{\mu}$ for all κ, μ with $\kappa \sqsubseteq \mu$.

   (b) $\tilde{\kappa} \sqsubseteq \iota$ for all κ top sort with $\mathrm{ord}(\kappa) < n$.

5. $\boldsymbol{\delta}$ is defined as the set of all term declarations

(a) $(\boldsymbol{\alpha}^{\tilde{\tau}}(x_{\tilde{\kappa}}, x_{\tilde{\kappa}_1}, \ldots, x_{\tilde{\kappa}_m}) : \tilde{\mu})$ for all $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu))$ with $\text{type}(\kappa) = \tau$ and $\text{ord}(\tau) < n$,*

(b) for all term declaration $(t : \kappa)$ of order less than n, $(\Theta(t) : \tilde{\kappa})$,

(c) for all constant declarations $(c : \kappa)$ with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$ of order equal to n, $(\Theta(c) : (\tilde{\kappa}_1 \times \cdots \times \tilde{\kappa}_m \to \tilde{\mu}))$, and

(d) $((x_{\tilde{\kappa}} \equiv x_{\tilde{\kappa}}) : \tilde{o})$ for all κ with $\text{ord}(\kappa) < n$ and $\kappa \neq o$.

Now we are going to define how terms and formulae are mapped by the morphism Ψ, which behaves on the signature exactly like Φ, with the only exception that $\equiv$ is not in the image of Ψ.

For terms it is defined inductively by:

T1 For a term with an m-ary function term f of sort κ of type τ as top expression we define
$\Psi(f(t_1, \ldots, t_m)) = \boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(f), \Psi(t_1), \ldots, \Psi(t_m))$

For formulae we define Ψ inductively by:

F1 For an atomic formula with predicate constant p of order n as top expression we define
$\Psi(p(t_1, \ldots, t_m)) = \Psi(p)(\Psi(t_1), \ldots, \Psi(t_m))$

F2 For a term with an m-ary predicate term p of sort κ of type τ and order less than n as top expression we define
$\Psi(p(t_1, \ldots, t_m)) = \boldsymbol{\alpha}^{\tilde{\tau}}(\Psi(p), \Psi(t_1), \ldots, \Psi(t_m))$

F3 For a conjunction we define
$\Psi(\varphi_1 \wedge \varphi_2) = \Psi(\varphi_1) \wedge \Psi(\varphi_2)$

F4 For a negation we define
$\Psi(\neg\varphi) = \neg\Psi(\varphi)$

F5 For a quantified formula we define
$\Psi(\forall x \varphi) = \forall \Psi(x) \Psi(\varphi)$

Ξ_α^Σ is the set consisting of the following formulae of $\mathcal{L}^1_{\Sigma,\equiv}$:

*That is, we use the functions and predicates $\boldsymbol{\alpha}$ quasi-polymorphic in the sense of remark 3.37.

$\Xi_\alpha^{\Sigma,f}$ For every function constant $\boldsymbol{\alpha}^{\tilde{\tau}}$ with $\tau = (\tau_1 \times \cdots \times \tau_m \rightarrow \sigma)$, $\sigma \neq o$ and for all sorts $\kappa = (\kappa_1 \times \cdots \times \kappa_m \rightarrow \mu)$ of type τ we have:

$$\forall f_{\tilde{\kappa}} \forall g_{\tilde{\kappa}} (\forall x^1_{\tilde{\kappa}_1}, \ldots, \forall x^m_{\tilde{\kappa}_m} \; \boldsymbol{\alpha}^{\tilde{\tau}}(f, x_1, \ldots, x_m) \equiv \boldsymbol{\alpha}^{\tilde{\tau}}(g, x_1, \ldots, x_m)) \Longrightarrow f \equiv g$$

$\Xi_\alpha^{\Sigma,p}$ For every predicate constant $\boldsymbol{\alpha}^{\tilde{\tau}}$ with $\tau = (\tau_1 \times \cdots \times \tau_m \rightarrow o)$ and for all sorts $\kappa = (\kappa_1 \times \cdots \times \kappa_m \rightarrow o)$ of type τ we have:

$$\forall p_{\tilde{\kappa}} \forall q_{\tilde{\kappa}} (\forall x^1_{\tilde{\kappa}_1}, \ldots, \forall x^m_{\tilde{\kappa}_m} \; \boldsymbol{\alpha}^{\tilde{\tau}}(p, x_1, \ldots, x_m) \iff \boldsymbol{\alpha}^{\tilde{\tau}}(q, x_1, \ldots, x_m)) \Longrightarrow p \equiv q$$

We define $\Phi(\varphi) = \Psi(\varphi) \cup \Xi_\alpha^\Sigma$. Analogously for formula sets $\Phi(\Gamma) = \Psi(\Gamma) \cup \Xi_\alpha^\Sigma$.

5.45 Remark: Analogously to 5.20 we have that Ψ_{2n-1} is an injective quasi-homomorphism from $\mathcal{L}_\Sigma^{2n-1}(\mathcal{S})$ to $\mathcal{L}_\Sigma^1(\Psi(\mathcal{S}))$, analogously to 5.21 and 5.42 Φ is weakly and strongly sound.

5.46 Theorem: Φ *is weakly complete.*

Proof: The proof is a reproduction of the proof of theorem 5.23. In the *first* step we introduce the basic notion, in particular a formula set Γ in $\mathcal{L}_\Sigma^n(\mathcal{S}_\Sigma)$ and an arbitrary model $\mathcal{M}$ of $\Phi(\Gamma)$. In the *second* step we define a frame with the help of which we will define a model for Γ. This model is defined inductively with the induction base $\check{\mathcal{D}}_\kappa := \mathcal{D}_{\tilde{\kappa}}$ for all κ of type ι and $\check{\mathcal{D}}_o := \mathcal{D}_{\tilde{o}}$. For top sorts $\kappa = (\kappa_1 \times \cdots \times \kappa_m \rightarrow \mu)$ we define $\check{\mathcal{D}}_\kappa$ as a subset of $\mathcal{F}(\check{\mathcal{D}}_{\kappa_1}, \ldots, \check{\mathcal{D}}_{\kappa_m}; \check{\mathcal{D}}_\mu)$. We cannot take the whole set, because then we would try to obtain strong completeness, which cannot be achieved in general. In order to construct $\check{\mathcal{D}}_\kappa$ we make use of the interpretations of the $\boldsymbol{\alpha}$-functions, especially we construct injective functions $\natural$, which maps $\mathcal{D}_{\tilde{\kappa}}$ to $\check{\mathcal{D}}_\kappa$. For the other sorts κ we define $\check{\mathcal{D}}_\kappa := \natural_{\mathrm{top\ sort}(\kappa)}(\mathcal{D}_{\tilde{\kappa}})$. In a *third* step we define an interpretation function $\check{\mathcal{J}}$ for $\mathcal{L}_\Sigma^n$ and show that the inclusion relations induced by the subsort relations hold. In a *fourth* step we show by induction on the construction of terms and formulae that the quasi-homomorphism Ψ is compatible with the model relation. Formally we show $\natural \circ \mathcal{V}_\xi^{\mathcal{M}} \circ \Psi = \mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}$. In a *fifth* and last step we use this property to show that $\check{\mathcal{M}} = \langle \{\check{\mathcal{D}}_\kappa\}_\kappa, \check{\mathcal{J}} \rangle$ is a model of Γ.

Step 1:

Let Γ be a formula set in $\mathcal{L}_\Sigma^{2n-1}(\mathcal{S}_\Sigma)$. Let $\mathcal{M}$ be a weak model of $\Phi(\Gamma)$. Then $\mathcal{M}$ is a model of $\Phi(\varphi)$ for every formula φ in Γ. Let $\mathcal{M}$ be $\langle \{\mathcal{D}_\kappa\}_\kappa, \mathcal{J} \rangle$ and ξ be an arbitrary assignment. Then we have $\mathcal{V}_\xi^{\mathcal{M}}(\Phi(\varphi)) = \mathtt{T}$. We want to construct a model $\check{\mathcal{M}}$ of φ, so that for all assignments $\check{\xi}$ we have $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi) = \mathtt{T}$.

<u>Step 2:</u>

In this step we define a frame for $\mathcal{L}_{\Sigma}^{2n-1}(\mathcal{S}_{\Sigma})$. Therefore we define $\check{\mathcal{D}}_{\iota} := \mathcal{D}_{\tilde{\iota}}$ and $\check{\mathcal{D}}_{o} := \mathcal{D}_{\tilde{o}}$. For all other top sorts κ with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$ we have to define $\check{\mathcal{D}}_{\kappa} \subseteq \mathcal{F}(\check{\mathcal{D}}_{\kappa_1}, \ldots, \check{\mathcal{D}}_{\kappa_m}; \check{\mathcal{D}}_{\mu})$. We do this by inductively defining injective functions $\natural_\kappa$ from $\mathcal{D}_{\tilde\kappa}$ to $\mathcal{F}(\check{\mathcal{D}}_{\kappa_1}, \ldots, \check{\mathcal{D}}_{\kappa_m}; \check{\mathcal{D}}_{\mu})$ and setting $\check{\mathcal{D}}_{\kappa} := \natural_\kappa(\mathcal{D}_{\tilde\kappa})$. Hence $\natural_\kappa$ is a bijective function from $\mathcal{D}_{\tilde\kappa}$ to $\check{\mathcal{D}}_{\kappa}$. For the other sorts κ with top sort μ we define: $\check{\mathcal{D}}_{\kappa} := \natural_\mu(\mathcal{D}_{\tilde\kappa})$.

We define $\natural_\kappa$ as bijective functions inductively:

1. $\natural_\iota : \mathcal{D}_{\tilde\iota} \to \check{\mathcal{D}}_{\iota}$ and $\natural_o : \mathcal{D}_{\tilde o} \to \check{\mathcal{D}}_{o}$ as the identity mappings (These functions are obviously bijective).

2. Let $\natural_{\kappa_i}$ and $\natural_\mu$ be defined for $\mathcal{D}_{\tilde\kappa_1}, \ldots, \mathcal{D}_{\tilde\kappa_m}$, and $\mathcal{D}_{\tilde\mu}$. We are going to define a function $\natural_\kappa$ with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to \mu)$, $\mu \neq o$, for $\mathcal{D}_{\tilde\kappa}$. For all $x \in \mathcal{D}_{\tilde\kappa}$ $\natural_\kappa(x)$ is defined as $\natural_\kappa(x)(\check{x}_1, \ldots, \check{x}_m) := \natural_\mu(\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x, \natural_{\kappa_1}^{-1}(\check{x}_1), \ldots, \natural_{\kappa_m}^{-1}(\check{x}_m)))$ for all $\check{x}_1 \in \check{\mathcal{D}}_{\kappa_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\kappa_m}$

   The following diagram may help to see the involved mappings at a glance:

$$\begin{array}{rcccccc}
\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau}) : & \mathcal{D}_{\tilde\kappa} & \times & \mathcal{D}_{\tilde\kappa_1} \times \cdots \times & \mathcal{D}_{\tilde\kappa_m} & \longrightarrow & \mathcal{D}_{\tilde\mu} \\
& !\downarrow\natural_\kappa & & \uparrow\natural_{\kappa_1}^{-1} & \uparrow\natural_{\kappa_m}^{-1} & & \downarrow\natural_\mu \\
& \check{\mathcal{D}}_\kappa \hookrightarrow & \mathcal{F}(& \check{\mathcal{D}}_{\kappa_1}, \ldots, & \check{\mathcal{D}}_{\kappa_m} & ; & \check{\mathcal{D}}_\mu)
\end{array}$$

   In order to show the injectivity of $\natural_\kappa$ we use that we have in $\Xi_\alpha^{\Sigma,f}$ the formula
   $$\forall f_{\tilde\kappa} \forall g_{\tilde\kappa} (\forall x_{\tilde\kappa_1}^1, \ldots, \forall x_{\tilde\kappa_m}^m \ \boldsymbol{\alpha}^{\tilde\tau}(f, x^1, \ldots, x^m) \equiv \boldsymbol{\alpha}^{\tilde\tau}(g, x^1, \ldots, x^m)) \Longrightarrow f \equiv g$$
   Therefore we have in a model for all x, x' in $\mathcal{D}_{\tilde\kappa}$
   $$\forall y_1 \in \mathcal{D}_{\tilde\kappa_1}, \ldots, \forall y_m \in \mathcal{D}_{\tilde\kappa_m} \ \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x, y_1, \ldots, y_m) \equiv_{\mathcal{D}_{\tilde\mu}} \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x', y_1, \ldots, y_m) \Longrightarrow x \equiv_{\mathcal{D}_{\tilde\kappa}} x' \qquad (*)$$
   Let $\natural_\kappa(x) \equiv_{\check{\mathcal{D}}_\kappa} \natural_\kappa(x')$ for arbitrary x and x' in $\mathcal{D}_{\tilde\kappa}$. Then we have by definition for all $\check{x}_1 \in \check{\mathcal{D}}_{\kappa_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\kappa_m}$
   $\natural_\mu \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x, \natural_{\kappa_1}^{-1}(\check{x}_1), \ldots, \natural_{\kappa_m}^{-1}(\check{x}_m)) \equiv_{\check{\mathcal{D}}_\mu} \natural_\mu \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x', \natural_{\kappa_1}^{-1}(\check{x}_1), \ldots, \natural_{\kappa_m}^{-1}(\check{x}_m))$.
   Since the mappings $\natural_{\kappa_1}, \ldots, \natural_{\kappa_m}, \natural_\mu$ are all bijective, we get for all $y_1 \in \mathcal{D}_{\tilde\kappa_1}, \ldots, y_m \in \mathcal{D}_{\tilde\kappa_m}$: $\mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x, y_1, \ldots, y_m) \equiv_{\mathcal{D}_{\tilde\mu}} \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x', y_1, \ldots, y_m)$. Because of the relation $(*)$ $x \equiv_{\mathcal{D}_{\tilde\kappa}} x'$, hence the injectivity is shown. Since the surjectivity is given by definition, we have proved that $\natural_\kappa$ is bijective.

3. Let $\natural_{\kappa_i}$ be defined for $\mathcal{D}_{\tilde\kappa_1}, \ldots, \mathcal{D}_{\tilde\kappa_m}$. We are going to define a function $\natural_\kappa$ (for order of κ is less than n) with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to o)$ for $\mathcal{D}_{\tilde\kappa}$. For all $x \in \mathcal{D}_{\tilde\kappa}$ $\natural_\kappa(x)$ is defined as $\natural_\kappa(x)(\check{x}_1, \ldots, \check{x}_m) := \natural_o \mathcal{V}_\xi^{\mathcal{M}}(\boldsymbol{\alpha}^{\tilde\tau})(x, \natural_{\kappa_1}^{-1}(\check{x}_1), \ldots, \natural_{\kappa_m}^{-1}(\check{x}_m))$

for all $\check{x}_1 \in \check{\mathcal{D}}_{\kappa_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\kappa_m}$. Analogously to case 2 we get the bijectivity of $\natural_\kappa$ by the corresponding formula in $\Xi_\alpha^{\Sigma,p}$.

4. Let $\natural_{\kappa_i}$ be defined for $\mathcal{D}_{\tilde{\kappa}_1}, \ldots, \mathcal{D}_{\tilde{\kappa}_m}$. We define a function $\natural_\kappa$ (for order of κ is equal to n) with $\kappa = (\kappa_1 \times \cdots \times \kappa_m \to o)$ for $\mathcal{D}_{\tilde{\kappa}}$. For all $p \in \mathcal{D}_{\tilde{\kappa}}$ $\natural_\kappa(p)$ is defined as $\natural_\kappa(p)(\check{x}_1, \ldots, \check{x}_m) := \natural_o p(\natural_{\kappa_1}^{-1}(\check{x}_1), \ldots, \natural_{\kappa_m}^{-1}(\check{x}_m))$ for all $\check{x}_1 \in \check{\mathcal{D}}_{\kappa_1}, \ldots, \check{x}_m \in \check{\mathcal{D}}_{\kappa_m}$. The bijectivity of $\natural_\kappa$ follows trivially.

Hence we have defined a frame $\{\check{\mathcal{D}}_\kappa\}_\kappa$ for all sorts κ.

Step 3:

In this step we define an interpretation mapping $\check{\mathcal{J}}$ in order to complete the definition of an interpretation $\langle\{\check{\mathcal{D}}_\kappa\}_\kappa, \check{\mathcal{J}}\rangle$. For all constants c we define $\check{\mathcal{J}}(c) := \natural \circ \mathcal{J} \circ \Psi(c)$. For all subsort relations $\kappa \sqsubseteq \mu$ (with same top sort ν) we have $\check{\mathcal{D}}_\kappa \subseteq \check{\mathcal{D}}_\mu$, because we have $\tilde{\kappa} \dot{\sqsubseteq} \tilde{\mu}$, thereby we get $\mathcal{D}_{\tilde{\kappa}} \subseteq \mathcal{D}_{\tilde{\mu}}$ and can conclude $\check{\mathcal{D}}_\kappa = \natural_\nu(\mathcal{D}_{\tilde{\kappa}}) \subseteq \natural_\nu(\mathcal{D}_{\tilde{\mu}}) = \check{\mathcal{D}}_\mu$.

Step 4:

In this step we have to show that for every assignment $\check{\xi}$ in $\check{\mathcal{M}}$ there is an assignment ξ in $\mathcal{M}$, so that for all terms (and hence all formulae) t we have: $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(t) = \natural \circ \mathcal{V}_\xi^{\mathcal{M}} \circ \Psi(t)$. Since the proof is analogous to the corresponding part or the proof of theorem 5.21 it is omitted here.

Step 5:

At first we notice that the term declarations are correctly interpreted. That is the case, because the corresponding term declarations hold in the translated case.

Now we are going to show that if $\mathcal{M}$ is a model of $\Phi(\varphi)$, then $\check{M}$ is a model of φ. If $\mathcal{M}$ is model of $\Phi(\varphi)$, then $\mathcal{M}$ is a model of $\Psi(\varphi)$. Let $\check{\xi}$ be an arbitrary assignment and ξ be defined as $\natural^{-1} \circ \check{\xi} \circ \Psi^{-1}$, then we have $\mathcal{V}_\xi^{\mathcal{M}}(\Psi(\varphi)) = \mathtt{T}$, because $\mathcal{M}$ is a model of $\Psi(\varphi)$. Hence we have $\mathcal{V}_{\check{\xi}}^{\check{\mathcal{M}}}(\varphi) = \natural(\mathcal{V}_\xi^{\mathcal{M}}(\Psi(\varphi))) = \mathtt{T}$. Recall that for truth values $\natural$ is the identity function. ■

5.47 Remark: Analogously to the unsorted case (see 5.25) Ψ^{-1} provides a calculus for $\mathcal{L}_\Sigma^n$. If we add rules that enforce that function symbols and predicate symbols are equal if they agree in all arguments, we can transform every sound and complete first-order calculus of $\mathcal{L}_\Sigma^1$ by Φ to a sound and weakly complete calculus for $\mathcal{L}_\Sigma^n$. We can execute the proof in $\mathcal{L}_\Sigma^1$ and then lift it to a proof in $\mathcal{L}_\Sigma^n$.

5.4 Relationship to Higher-Order Theorem Proving

An alternative approach to the translation techniques is to build a higher-order theorem provers in the first place. These systems are usually based on CHURCH'S λ-calculus [33] and the main advantage of this approach is that the additional axioms (namely the comprehension axioms Υ) are obsolete. The price one has to pay is that unification becomes undecidable and far more complex in general [67]. In TPS such a theorem prover for unsorted logic is realized. A sorted calculus based on the λ-calculus can be found in [75].

The following example should show the differences between the two approaches.

5.48 Example: Let us consider the formula set $\Gamma = \{\forall f_{(\iota\rightarrow\iota)} \forall x_\iota P(f(x)); \neg P(a)\}$ with object constant a of type ι and predicate constant P of type $(\iota \rightarrow o)$. In order to show that the formula set Γ is unsatisfiable, we have to add a comprehension axiom of the form $\exists f \forall x\ f(x) \equiv a$ or $\exists f \forall x\ f(x) \equiv x$ to Γ. Doing so we can show in our higher-order logic $\mathcal{L}^2$ that the set is unsatisfiable. In the λ-calculus one can find immediately some unifiers that make the two formulae of Γ complementary. For instance:
$\{f \leftarrow \lambda y.y; x \leftarrow a\}$ and $\{f \leftarrow \lambda y.a\}$. That is, in the λ-calculus the functions $\lambda y.y$ and $\lambda y.a$ are available as primitives without giving them names and introducing them explicitly as objects by some axioms.

The question which of these two methods is better, remains open, but there is evidence that for essentially first-order theorems (compare definition 3.24) the translation techniques are preferential, whereas for truly higher-order theorems it is better to search for the proof in the λ-calculus.

Chapter 6

Examples and Practical Considerations on Translations

Begriffe ohne Anschauungen sind leer und Anschauungen ohne Begriffe blind.

Immanuel Kant

In this chapter we present examples for the translation of theorems from higher-order logic into first-order logic and their proofs by the Markgraf Karl Refutation Procedure [93]. In the first section we present a proof of an essentially first-order theorem, in the second a proof of a truly higher-order theorem. In the third section the advantage of higher-order *sorted* language is shown by an example.

6.1 An Essentially First-Order Theorem

As an example for an essentially first-order theorem (for the definition of *essentially first-order* compare definition 3.24) we give a proof of theorem 4.3 of [40, p.34], that the composition of binary relations is associative. We will formulate the theorem in our higher-order logic.

Let ϱ and σ be two (binary) relations over a (fixed) set S. (Since this set S is fixed and no other is under consideration, we will take it as the universe of individuals.) The relation $\varrho \circ \sigma$ is defined by:
$(s,t) \in \varrho \circ \sigma :\Longleftrightarrow \exists r\ (s,r) \in \varrho\ \wedge\ (r,t) \in \sigma.$
The theorem is:
$\forall \varrho, \sigma, \tau\ \ (\varrho \circ \sigma) \circ \tau = \varrho \circ (\sigma \circ \tau).$
Formally we get:

- Definition of Composition:
  $\forall \varrho_{(\iota \times \iota \to o)} \forall \sigma_{(\iota \times \iota \to o)} \forall x_\iota \forall y_\iota\ \ (\varrho \circ \sigma)(x,y) \iff (\exists z_\iota\ \ \varrho(x,z) \wedge \sigma(z,y))$
- Extensionality Axiom:
  $\forall \varrho_{(\iota \times \iota \to o)} \forall \sigma_{(\iota \times \iota \to o)}\ \ (\forall x_\iota \forall y_\iota\ \ \varrho(x,y) \iff \sigma(x,y)) \Longrightarrow \varrho \equiv \sigma$

– Theorem:
$\forall \varrho_{(\iota\times\iota\rightarrow o)} \forall \sigma_{(\iota\times\iota\rightarrow o)} \forall \tau_{(\iota\times\iota\rightarrow o)} \ (\varrho\circ\sigma)\circ\tau \equiv \varrho\circ(\sigma\circ\tau)$.

This will be translated into first-order logic by a quasi-homomorphism. The $\boldsymbol{\alpha}^{"\iota\times\iota\rightarrow o"}$ predicate is written as `A[IXITO]` and so on. The type of binary relations is translated to the sort `IXITO`, where `I` stands for ι and `O` for o. For example, $\varrho(x, y)$ is translated into the first-order atom `A[IXITO](rho x y)`. The representation in Markgraf Karl syntax is:

```
Formulae given to the editor
============================

Axioms:
* SORTS *
SORT I,ITO,IXITO:ANY
* TERM DECLARATIONS *
TYPE A[IXITO](IXITO I I)
TYPE A[ITO](ITO I)
* DEFINITION OF COMPOSITION *
TYPE COMP(IXITO IXITO):IXITO
ALL RHO:IXITO ALL SIGMA:IXITO (ALL X:I ALL Y:I
  A[IXITO](COMP(RHO SIGMA) X Y) EQV
  (EX Z:I A[IXITO](RHO X Z) AND A[IXITO](SIGMA Z Y)))
* EXTENSIONALITY *
ALL RHO:IXITO ALL SIGMA:IXITO
 (ALL X:I ALL Y:I A[IXITO](RHO X Y) EQV A[IXITO](SIGMA X Y))
 IMPL RHO = SIGMA

Theorems:
ALL RHO:IXITO ALL SIGMA:IXITO ALL TAU:IXITO
  COMP(COMP(RHO SIGMA) TAU) = COMP(RHO COMP(SIGMA TAU))


Refutation:
===========

Initial Clauses:
  A1: All x:Any + =(x x)
* A2: All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](u y f_1(u x y z))
* A3: All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](z f_1(u x y z) x)
* A4: All x,y,z:I u,v:Ixito + A[IXITO](comp(v u) z y)  - A[IXITO](v z x)
                            - A[IXITO](u x y)
* A5: All x,y:Ixito - A[IXITO](y f_2(y x) f_3(y x))  - A[IXITO](x f_2(y x) f_3(y x))
                    + =(y x)
* A6: All x,y:Ixito + A[IXITO](y f_2(y x) f_3(y x))  + A[IXITO](x f_2(y x) f_3(y x))
                    + =(y x)
* T7: - =(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))



A6,3 & T7,1      --> * R1:
  + A[IXITO](comp(comp(c_1 c_2) c_3)
           f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
           f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3))))
  + A[IXITO](comp(c_1 comp(c_2 c_3))
           f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
           f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3))))
```

⋮

```
R143,1 & R136,1 --> * R152:
  - A[IXITO](c_2
             f_1(c_1
                 f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                 f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                 comp(c_2 c_3))
             f_1(c_2
                 f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                 f_1(c_1
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     comp(c_2 c_3))
                 c_3))
  - A[IXITO](c_3
             f_1(c_2
                 f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                 f_1(c_1
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     comp(c_2 c_3))
                 c_3)
             f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3))))


R152,1 & R149,1 --> * R153:
  - A[IXITO](c_3
             f_1(c_2
                 f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                 f_1(c_1
                     f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     f_2(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3)))
                     comp(c_2 c_3))
                 c_3)
             f_3(comp(comp(c_1 c_2) c_3) comp(c_1 comp(c_2 c_3))))


R153,1 & R148,1 --> * R154:
  []

  q.e.d.

Time Used for Refutation: 3893 seconds
```

The MKRP needs 3893 seconds (more than an hour) to prove this simple theorem. The system produces 154 clauses, many of them rather difficult (the last three can be seen above). Although we have translated the theorem such that the preconditions are minimal and the initial clause set is relatively simple, we get an unfolding of the following axiom during normalization*:

*This is an observation of Axel Präcklein. In a joint work [73] we propose to introduce tactics that reformulate problem formulations in order to get a more adequate formulation.

$$
\begin{aligned}
& \boxed{\forall}\,(\,\boxed{\forall}\,A \iff B) \Longrightarrow C \\
\rightsquigarrow\ & \boxed{\forall}\,\neg(\,\boxed{\forall}\,A \iff B) \vee C \\
\rightsquigarrow\ & \boxed{\forall}\,\neg((\,\boxed{\forall}\,A \Longrightarrow B) \wedge (\,\boxed{\forall}\,B \Longrightarrow A)) \vee C \\
\rightsquigarrow\ & \boxed{\forall}\,\neg(\,\boxed{\forall}\,\neg A \vee B) \vee \neg(\,\boxed{\forall}\,\neg B \vee A) \vee C \\
\rightsquigarrow\ & \boxed{\forall}\,(\,\boxed{\exists}\,\neg(\neg A \vee B)) \vee (\,\boxed{\exists}\,\neg(\neg B \vee A)) \vee C \\
\rightsquigarrow\ & \boxed{\forall}\,(\,\boxed{\exists}\,A \wedge \neg B) \vee (\,\boxed{\exists}\,B \wedge \neg A) \vee C \\
\rightsquigarrow\ & \boxed{\forall}\,(\neg A \vee \neg B \vee C) \wedge (A \vee B \vee C)
\end{aligned}
$$

The last formula corresponds to `A5` and `A6` in the normalization above. We have two clauses, where the As and Bs contain complicated SKOLEM functions because of quantifiers in the theorem and they must be resolved in a difficult manner.

Looking at the input formulae and at the result of the normalization we see that it is very useful to first perform a preprocessing step. The structure of the extensionality axiom is $(A \iff B) \Longrightarrow C$ where C matches the theorem C'. Hence we can construct a new theorem $A' \iff B'$ according to this match. Starting with this "resolvent" as theorem we can avoid the unfolding during normalization.

By the preprocessing step one can replace the three-literal clauses `A5` and `A6` as well as the theorem clause `T7` by the simple clauses `T5` and `T6` in the proof below and hence avoid the SKOLEM functions `f_2` and `f_3`. The general *explicit* formulation of the extensionality axiom is replaced by a special *implicit* one. The alternative formulation is:

```
Formulae given to the editor
============================

Axioms:
* SORTS *
SORT I,ITO,IXITO:ANY
* TERM DECLARATIONS *
TYPE A[IXITO](IXITO I I)
TYPE A[ITO](ITO I)
* DEFINITION OF COMPOSITION *
TYPE COMP(IXITO IXITO):IXITO
ALL RHO:IXITO ALL SIGMA:IXITO (ALL X:I ALL Y:I
  A[IXITO](COMP(RHO SIGMA) X Y) EQV
  (EX Z:I A[IXITO](RHO X Z) AND A[IXITO](SIGMA Z Y)))

Theorems:
ALL RHO:IXITO ALL SIGMA:IXITO ALL TAU:IXITO ALL X:I ALL Y:I
  A[IXITO](COMP(COMP(RHO SIGMA) TAU) X Y) EQV A[IXITO](COMP(RHO COMP(SIGMA TAU)) X Y)
```

```
Refutation:
===========

Initial Clauses:
  A1:   All x:Any + =(x x)
* A2:   All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](u y f_1(u x y z))
* A3:   All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](z f_1(u x y z) x)
* A4:   All x,y,z:I u,v:Ixito + A[IXITO](comp(v u) z y)  - A[IXITO](v z x)
                            - A[IXITO](u x y)
* T5:   - A[IXITO](comp(comp(c_1 c_2) c_3) c_4 c_5)
        - A[IXITO](comp(c_1 comp(c_2 c_3)) c_4 c_5)
* T6:   + A[IXITO](comp(comp(c_1 c_2) c_3) c_4 c_5)
        + A[IXITO](comp(c_1 comp(c_2 c_3)) c_4 c_5)



T6,2 & A3,1    --> * R1:
  + A[IXITO](comp(comp(c_1 c_2) c_3) c_4 c_5)
  + A[IXITO](comp(c_2 c_3) f_1(c_1 c_5 c_4 comp(c_2 c_3)) c_5)

                                        ⋮

R5,2 & R37,3   --> * R38:
  + A[IXITO](comp(comp(c_1 c_2) c_3) c_4 c_5)
  + A[IXITO](comp(comp(c_1 c_2) c_3) c_4 c_5)
  + A[IXITO](comp(c_1 c_2) c_4 f_1(c_2 c_5 f_1(c_1 c_5 c_4 comp(c_2 c_3)) c_3))

                                        ⋮

R77,1 & R61,1  --> * R78:
  - A[IXITO](c_3 f_1(c_2 c_5 f_1(c_1 c_5 c_4 comp(c_2 c_3)) c_3) c_5)

R78,1 & R76,1  --> * R79:
  []

   q.e.d.

   Time Used for Refutation: 326 seconds
```

R38 is one of the most difficult clauses in the whole proof. The proof is found in 326 seconds – compared to 3893 seconds a drastical improvement.

In addition, it is now possible to split the theorem automatically into the two parts $A' \Longrightarrow B'$ and $B' \Longrightarrow A'$ with computation times for the splitparts of 20 seconds each, that means, total refutation time of 40 seconds. In this formulation the original theorem clause T7 is replaced by the unit clauses T5 through T8.

```
Set of Axiom Clauses Resulting from Normalization
=================================================

  A1:   All x:Any + =(x x)
* A2:   All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](u y f_1(u x y z))
* A3:   All x,y:I z,u:Ixito - A[IXITO](comp(u z) y x)  + A[IXITO](z f_1(u x y z) x)
* A4:   All x,y,z:I u,v:Ixito + A[IXITO](comp(v u) z y)  - A[IXITO](v z x)
                            - A[IXITO](u x y)
```

```
Set of Theorem Clauses Resulting from Normalization and Splitting
=================================================================

Splitpart 1
* T5: - A[IXITO](comp(comp(c_4 c_1) c_2) c_3 c_5)
* T6: + A[IXITO](comp(c_4 comp(c_1 c_2)) c_3 c_5)

Splitpart 2
* T7: + A[IXITO](comp(comp(c_9 c_6) c_7) c_8 c_10)
* T8: - A[IXITO](comp(c_9 comp(c_6 c_7)) c_8 c_10)

Refutation of Splitpart 1:
==========================

T6,1 & A3,1  --> * R1:
  + A[IXITO](comp(c_1 c_2) f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_5)

T6,1 & A2,1  --> * R2:
  + A[IXITO](c_4 c_3 f_1(c_4 c_5 c_3 comp(c_1 c_2)))

R1,1 & A2,1  --> * R3:
  + A[IXITO](c_1
             f_1(c_4 c_5 c_3 comp(c_1 c_2))
             f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))

R1,1 & A3,1  --> * R4:
  + A[IXITO](c_2 f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2) c_5)

A4,1 & T5,1  --> * R5:
  All x:I - A[IXITO](comp(c_4 c_1) c_3 x)  - A[IXITO](c_2 x c_5)

R4,1 & R5,2  --> * R6:
  - A[IXITO](comp(c_4 c_1) c_3 f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))

R3,1 & A4,3  --> * R7:
  + A[IXITO](comp(c_4 c_1) c_3 f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))
  - A[IXITO](c_4 c_3 f_1(c_4 c_5 c_3 comp(c_1 c_2)))

R7,2 & R2,1  --> * R8:
  + A[IXITO](comp(c_4 c_1) c_3 f_1(c_1 c_5 f_1(c_4 c_5 c_3 comp(c_1 c_2)) c_2))

R8,1 & R6,1  --> * R9:
  []


Refutation of Splitpart 2:
==========================

T7,1 & A3,1    --> * R10:
  + A[IXITO](c_7 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_10)

T7,1 & A2,1    --> * R11:
  + A[IXITO](comp(c_9 c_6) c_8 f_1(comp(c_9 c_6) c_10 c_8 c_7))

R11,1 & A3,1   --> * R12:
  + A[IXITO](c_6
             f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6)
             f_1(comp(c_9 c_6) c_10 c_8 c_7))
```

```
R11,1 & A2,1   --> * R13:
  + A[IXITO](c_9 c_8 f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6))

A4,1 & T8,1    --> * R14:
  All x:I - A[IXITO](c_9 c_8 x)  - A[IXITO](comp(c_6 c_7) x c_10)

R13,1 & R14,1  --> * R15:
  - A[IXITO](comp(c_6 c_7) f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6) c_10)

R10,1 & A4,3   --> * R16:
  + A[IXITO](comp(c_6 c_7) f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6) c_10)
  - A[IXITO](c_6
             f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6)
             f_1(comp(c_9 c_6) c_10 c_8 c_7))

R16,2 & R12,1  --> * R17:
  + A[IXITO](comp(c_6 c_7) f_1(c_9 f_1(comp(c_9 c_6) c_10 c_8 c_7) c_8 c_6) c_10)

R17,1 & R15,1  --> * R18:
  []

   q.e.d.

   Time Used for Refutation of Splitpart 1: 20 seconds
   Time Used for Refutation of Splitpart 2: 20 seconds
```

Summarizing it is possible to say that the problem representation is crucial for automated theorem proving. An automated theorem prover is very sensitive to the problem formulation. Therefore even for essentially first-order theorems the question of how to prove it by a first-order theorem prover is not answered by simply translating them from higher-order into into first-order. The problem, *how* to translate and what kind of problem presentation should be chosen, is largely uninvestigated. In [73] some ideas in this direction can be found. Perhaps the most important question – but as seen in the example not the only one – is how to find a minimal set of preconditions for proving the theorem, because otherwise the search space for a proof is too big. In order to find such a minimal or almost minimal set analogy will play an important rôle. Some examples in this direction can be found in [71].

6.2 A Truly Higher-Order Theorem

As an example for a truly higher-order theorem (for the definition of *truly higher-order* compare definition 3.24) we present a proof of Cantor's theorem that the power-set of a set has greater cardinality than the set itself. We use the formulation of Andrews [3, p.184, X5304] (compare figure 4.12).

A formulation in our higher-order logic is – sets are encoded as predicates of the type $(\iota \to o)$:

- Definition of subset:
  $\forall A_{(\iota\to o)}\forall B_{(\iota\to o)} \;\; A \subseteq B \iff (\forall x_\iota \;\; A(x) \Longrightarrow B(x)).$

- Theorem:
  $\forall s_{(\iota\to o)}\neg\exists g_{(\iota\to(\iota\to o))}\forall f_{(\iota\to o)} \;\; f \subseteq s \Longrightarrow (\exists j_\iota \;\; s(j) \;\wedge\; g(j) \equiv f)$

- In order to prove this theorem by a translation to first-order logic, we need the following comprehension axiom, which incorporates the idea of diagonalization:
  $\forall s_{(\iota\to o)}\forall g_{(\iota\to(\iota\to o))}\exists p_{(\iota\to o)}\forall x_\iota \;\; (p(x) \iff s(x) \;\wedge\; \neg g(x)(x))$

Now we show a quasi-homomorphic translation into first-order logic (in MKRP-syntax) and a proof by the MKRP system:

```
Formulae given to the editor
============================

Axioms:
* SORTS *
SORT I,ITO,IT[ITO]:ANY
* TERM DECLARATIONS *
TYPE A[ITO](ITO I)
TYPE A[IT[ITO]](IT[ITO] I):ITO
TYPE SUBSET (ITO ITO)
* DEFINITION SUBSET *
ALL A,B:ITO SUBSET(A B) EQV (ALL X:I A[ITO](A X) IMPL A[ITO](B X))
* COMPREHENSION AXIOM *
ALL S:ITO ALL G:IT[ITO] EX P:ITO ALL X:I
  A[ITO](P X) EQV (A[ITO](S X) AND (NOT A[ITO](A[IT[ITO]](G X) X)))

Theorems:
ALL S:ITO (NOT EX G:IT[ITO] ALL F:ITO
  SUBSET(F S) IMPL (EX J:I A[ITO](S J) AND A[IT[ITO]](G J) = F))


Refutation:
===========

Initial Clauses:
  A1:    All x:Any + =(x x)
* A2:    All x:I y:It[ito] z:Ito - A[ITO](f_1(z y) x)  + A[ITO](z x)
* A3:    All x:I y:It[ito] z:Ito - A[ITO](f_1(z y) x)  - A[ITO](a[it[ito]](y x) x)
* A4:    All x:I y:It[ito] z:Ito + A[ITO](f_1(z y) x)  - A[ITO](z x)
                                 + A[ITO](a[it[ito]](y x) x)
* T5:    All x:Ito + A[ITO](x f_2(x))  + A[ITO](c_1 f_3(x))
* T6:    All x:Ito + A[ITO](x f_2(x))  + =(a[it[ito]](c_2 f_3(x)) x)
* T7:    All x:Ito - A[ITO](c_1 f_2(x))  + A[ITO](c_1 f_3(x))
* T8:    All x:Ito - A[ITO](c_1 f_2(x))  + =(a[it[ito]](c_2 f_3(x)) x)



T5,1 & A2,1     --> * R8:
  All x:It[ito] y:Ito + A[ITO](c_1 f_3(f_1(y x)))  + A[ITO](y f_2(f_1(y x)))
```

```
R8,2 & T7,1     --> * R9:
  All x:It[ito] + A[ITO](c_1 f_3(f_1(c_1 x)))  + A[ITO](c_1 f_3(f_1(c_1 x)))

R9 1=2          --> * D10:
  All x:It[ito] + A[ITO](c_1 f_3(f_1(c_1 x)))

T6,1 & A2,1     --> * R12:
  All x:It[ito] y:Ito + =(a[it[ito]](c_2 f_3(f_1(y x))) f_1(y x))
                      + A[ITO](y f_2(f_1(y x)))

R12,2 & T8,1    --> * R13:
  All x:It[ito] + =(a[it[ito]](c_2 f_3(f_1(c_1 x))) f_1(c_1 x))
                + =(a[it[ito]](c_2 f_3(f_1(c_1 x))) f_1(c_1 x))

R13 1=2         --> * D14:
  All x:It[ito] + =(a[it[ito]](c_2 f_3(f_1(c_1 x))) f_1(c_1 x))

D14,1 & A3,2    --> * P15:
  All x:Ito y:It[ito] - A[ITO](f_1(c_1 y) f_3(f_1(c_1 y)))
                      - A[ITO](f_1(x c_2) f_3(f_1(c_1 y)))

P15 (factor)    --> * F16:
  - A[ITO](f_1(c_1 c_2) f_3(f_1(c_1 c_2)))

A4,1 & F16,1    --> * R17:
  - A[ITO](c_1 f_3(f_1(c_1 c_2)))
  + A[ITO](a[it[ito]](c_2 f_3(f_1(c_1 c_2))) f_3(f_1(c_1 c_2)))

R17,2 & D14     --> * RW18:
  - A[ITO](c_1 f_3(f_1(c_1 c_2)))  + A[ITO](f_1(c_1 c_2) f_3(f_1(c_1 c_2)))

RW18,2 & P15,2  --> * R19:
  - A[ITO](c_1 f_3(f_1(c_1 c_2)))  - A[ITO](f_1(c_1 c_2) f_3(f_1(c_1 c_2)))

R19,2 & RW18,2  --> * R20:
  - A[ITO](c_1 f_3(f_1(c_1 c_2)))  - A[ITO](c_1 f_3(f_1(c_1 c_2)))

R20 1=2         --> * D21:
  - A[ITO](c_1 f_3(f_1(c_1 c_2)))

D21,1 & D10,1   --> * R22:
  []

   q.e.d.

   Time Used for Refutation: 44 seconds
```

The main problem with this formulation is to find the corresponding comprehension axiom. Indeed the whole proof idea – the diagonalization – is formulated in this axiom. If the theorem is proved by a higher-order theorem prover such an axiom is not necessary, because the higher-order unification algorithm produces a corresponding unifier. Therefore higher-order theorem provers are usually better suited for proving truly higher-order theorems. However, as in this example so in the general case, even for these theorems a first-order theorem prover can be used, albeit not always with advantage.

6.3 A Sorted Higher-Order Theorem

Now we give a comparison of the behaviour of a theorem prover when we use a sorted and an unsorted formulation of the same problem. The example is theorem (4.11.1) of [40, p.40]: For the formulation of the theorem we need the notion of an induced equivalence relation:
Let $\varphi: S \to U$ be a map, then φ induces an equivalence relation ϱ on S by setting $\varrho(s,t)$ iff $\varphi(s) = \varphi(t)$.
The theorem is:
Let $\varphi_1: S \to U$ and $\varphi_2: S \to V$ be mappings and let ϱ_1 and ϱ_2 be the induced equivalence relations. If there exists a map $\Phi: U \to V$ with $\Phi \circ \varphi_1 = \varphi_2$, then we have $\varrho_1 \subseteq \varrho_2$.

This can be axiomatized in sorted higher-order logic by:

- Subsort Declarations:
  $S \sqsubseteq \iota$, $U \sqsubseteq \iota$, $V \sqsubseteq \iota$
- Definition of Subset:
  $\forall f_{(S\times S\to o)} \forall g_{(S\times S\to o)}\ \ f \subseteq g \iff (\forall x_S \forall y_S\ \ f(x,y) \Longrightarrow g(x,y))$.
- Definition of Composition:
  $\forall f_{(U\to V)} \forall g_{(S\to U)} (\forall x_S\ \ (f\circ g)(x) \equiv f(g(x)))$
- Definition of Induced Equivalence Relation:
  $\forall \varphi_{(S\to\iota)} (\forall x_S \forall y_S\ \ IND(\varphi)(x,y) \iff \varphi(x) \equiv \varphi(y))$
- Theorem:
  $\forall \varphi^1_{(S\to U)} \forall \varphi^2_{(S\to V)} \forall \varrho^1_{(S\times S\to o)} \forall \varrho^2_{(S\times S\to o)}\ \ \varrho^1 \equiv IND(\varphi^1) \wedge \varrho^2 \equiv IND(\varphi^2) \Longrightarrow$
  $(\exists \Phi_{(U\to V)}\ \ \Phi \circ \varphi^1 \equiv \varphi^2 \Longrightarrow \varrho^1 \subseteq \varrho^2)$

```
Formulae given to the editor
============================

Axioms:
* SORTS *
SORT I,O,ITO,IXITO,[SXSTO]X[SXSTO]TO,[ITI]X[ITI]T[ITI],[ITI]T[IXITO],ITI:ANY
SORT S,U,V:I
SORT STO:ITO
SORT SXSTO:IXITO
SORT [UTV]X[STU]T[STV]:[ITI]X[ITI]T[ITI]
SORT [STI]T[SXSTO]:[ITI]T[IXITO]
SORT STI,UTV:ITI
SORT STU,STV:STI
* TERM DECLARATIONS *
TYPE A[IXITO](IXITO I I)
TYPE A[[ITI]X[ITI]T[ITI]]([ITI]X[ITI]T[ITI] ITI ITI):ITI
```

```
TYPE A[[ITI]T[IXITO]]([ITI]T[IXITO] ITI):IXITO
TYPE A[ITI](ITI I):I
* DEFINITION OF SUBSET *
TYPE SUBSET(SXSTO SXSTO)
ALL F,G:SXSTO SUBSET(F G) EQV (ALL X,Y:S A[IXITO](F X Y) IMPL A[IXITO](G X Y))
* DEFINITION OF THE COMPOSITION *
TYPE COMP:[UTV]X[STU]T[STV]
ALL F:UTV ALL G:STU (ALL X:S A[ITI](A[[ITI]X[ITI]T[ITI]](COMP F G) X) =
                             A[ITI](F A[ITI](G X)))
* DEFINITION OF INDUCED EQUIVALENCE RELATION *
TYPE IND:[STI]T[SXSTO]
ALL PHI:STI (ALL X,Y:S A[IXITO](A[[ITI]T[IXITO]](IND PHI) X Y) EQV
                       A[ITI](PHI X) = A[ITI](PHI Y))

Theorems:
ALL PHI1:STU ALL PHI2:STV ALL RHO1,RHO2:SXSTO
  RHO1 = A[[ITI]T[IXITO]](IND PHI1) AND RHO2 = A[[ITI]T[IXITO]](IND PHI2) IMPL
  ((EX PPHI:UTV A[[ITI]X[ITI]T[ITI]](COMP PPHI PHI1) = PHI2) IMPL
   SUBSET (RHO1 RHO2))


Refutation:
===========

Initial Clauses:
  A1:   All x:Any + =(x x)
* A2:   All x:S y:Stu z:Utv + =(a[iti](a[[iti]x[iti]t[iti]](comp z y) x)
                                a[iti](z a[iti](y x)))
* A3:   All x,y:S z:Sti + A[IXITO](a[[iti]t[ixito]](ind z) y x)
                        - =(a[iti](z y) a[iti](z x))
* A4:   All x,y:S z:Sti - A[IXITO](a[[iti]t[ixito]](ind z) y x)
                        + =(a[iti](z y) a[iti](z x))
* T5:   + =(c_4 a[[iti]t[ixito]](ind c_1))
* T6:   + =(c_3 a[[iti]t[ixito]](ind c_2))
* T7:   + =(a[[iti]x[iti]t[iti]](comp c_5 c_1) c_2)
* T8:   + A[IXITO](c_4 c_7 c_6)
* T9:   - A[IXITO](c_3 c_7 c_6)



T7,1 & A2,1   --> * P1:
  All x:S + =(a[iti](c_2 x) a[iti](c_5 a[iti](c_1 x)))

T5,1 & A4,1   --> * P2:
  All x,y:S - A[IXITO](c_4 y x)  + =(a[iti](c_1 y) a[iti](c_1 x))

T8,1 & P2,1   --> * R3:
  + =(a[iti](c_1 c_7) a[iti](c_1 c_6))

R3,1 & P1,1   --> * P4:
  + =(a[iti](c_2 c_6) a[iti](c_5 a[iti](c_1 c_7)))

P4,1 & P1     --> * RW5:
  + =(a[iti](c_2 c_6) a[iti](c_2 c_7))

T6,1 & A3,1   --> * P8:
  All x,y:S + A[IXITO](c_3 y x)  - =(a[iti](c_2 y) a[iti](c_2 x))

P8,2 & RW5,1  --> * R9:
  + A[IXITO](c_3 c_7 c_6)
```

```
R9,1 & T9,1   --> * R10:
  []

  q.e.d.

  Time Used for Refutation: 22 seconds
```

Now we prove the same theorem in an unsorted version. We translate it by $\partial\Re$ (compare definition 5.36) into a many-sorted first-order formulation. The subsort declarations `SORT STU,STV:STI` are translated to the formulae `ALL PHI:ITI STV(PHI) IMPL STI(PHI)` and `ALL PHI:ITI STU(PHI) IMPL STI(PHI)`. All other subsort relations are omitted, because they are not necessary for the proof. We present the formulation in the MKRP syntax and a proof fragment:

```
Formulae given to the editor
============================

Axioms:
* SORTS *
SORT I,O,ITO,IXITO,[ITI]X[ITI]T[ITI],[ITI]T[IXITO],ITI:ANY
* TERM DECLARATIONS *
TYPE A[IXITO](IXITO I I)
TYPE A[[ITI]X[ITI]T[ITI]]([ITI]X[ITI]T[ITI] ITI ITI):ITI
TYPE A[[ITI]T[IXITO]]([ITI]T[IXITO] ITI):IXITO
TYPE A[ITI](ITI I):I
TYPE STI(ITI)
TYPE STV(ITI)
TYPE STU(ITI)
* TRANSLATED SUBSORT RELATIONS *
ALL PHI:ITI STV(PHI) IMPL STI(PHI)
ALL PHI:ITI STU(PHI) IMPL STI(PHI)
* DEFINITION OF SUBSET *
TYPE SUBSET(IXITO IXITO)
ALL F:IXITO SXSTO(F) IMPL (ALL G:IXITO SXSTO (G) IMPL
  (SUBSET(F G) EQV (ALL X:I S(X) IMPL (ALL Y:I S(Y) IMPL
                  (A[IXITO](F X Y) IMPL A[IXITO](G X Y))))))
* DEFINITION OF THE COMPOSITION *
TYPE COMP:[ITI]X[ITI]T[ITI]
ALL F:ITI UTV(F) IMPL (ALL G:ITI STU(G) IMPL
  (ALL X:I S(X) IMPL A[ITI](A[[ITI]X[ITI]T[ITI]](COMP F G) X) =
                   A[ITI](F A[ITI](G X))))
* DEFINITION OF INDUCED EQUIVALENCE RELATION *
TYPE IND:[ITI]T[IXITO]
ALL PHI:ITI STI(PHI) IMPL (ALL X:I S(X) IMPL (ALL Y:I S(Y) IMPL
  (A[IXITO](A[[ITI]T[IXITO]](IND PHI) X Y) EQV
   A[ITI](PHI X) = A[ITI](PHI Y)))

Theorems:
ALL PHI1:ITI STU(PHI1) IMPL (ALL PHI2:ITI STV(PHI2) IMPL
  (ALL RHO1:IXITO SXSTO(RHO1) IMPL (ALL RHO2:IXITO SXSTO(RHO2) IMPL
     (RHO1 = A[[ITI]T[IXITO]](IND PHI1) AND RHO2 = A[[ITI]T[IXITO]](IND PHI2) IMPL
     ((EX PPHI:ITI UTV(PPHI) AND (A[[ITI]X[ITI]T[ITI]](COMP PPHI PHI1) = PHI2) IMPL
      SUBSET (RHO1 RHO2)))))))
```

```
Refutation:
===========


Initial Clauses:
  A1:   All x:Any + =(x x)
* A2:   All x:Iti - STV(x)  + STI(x)
* A3:   All x:Iti - STU(x)  + STI(x)
* A4:   All x,y:Ixito - SXSTO(y)  - SXSTO(x)  + SUBSET(y x)  + S(f_1(x y))
* A5:   All x,y:Ixito - SXSTO(y)  - SXSTO(x)  + SUBSET(y x)  + S(f_2(x y))
* A6:   All x,y:Ixito - SXSTO(y)  - SXSTO(x)  + SUBSET(y x)
                    + A[IXITO](y f_1(x y) f_2(x y))
* A7:   All x,y:Ixito - SXSTO(y)  - SXSTO(x)  + SUBSET(y x)
                    - A[IXITO](x f_1(x y) f_2(x y))
* A8:   All x:I y,z:Iti - UTV(z)  - STU(y)  - S(x)
                      + =(a[iti](a[[iti]x[iti]t[iti]](comp z y) x)
                          a[iti](z a[iti](y x)))
  A9:   All x,y:I z:Iti - STI(z)  - S(y)  + S(x)  + =(a[iti](z y) a[iti](z x))
* A10:  All x,y:I z:Iti - STI(z)  - S(y)  - A[IXITO](a[[iti]t[ixito]](ind z) y x)
                      + =(a[iti](z y) a[iti](z x))
* A11:  All x,y:I z:Iti - STI(z)  - S(y)  - S(x)
                      + A[IXITO](a[[iti]t[ixito]](ind z) y x)
                      - =(a[iti](z y) a[iti](z x))
  A12:  All x,y:I z,u:Ixito - SXSTO(u)  - SXSTO(z)  - SUBSET(u z)  - S(y)  - S(x)
                          - A[IXITO](u y x)  + A[IXITO](z y x)
* T13: + STU(c_1)
* T14: + STV(c_2)
* T15: + SXSTO(c_3)
* T16: + SXSTO(c_4)
* T17: + =(c_3 a[[iti]t[ixito]](ind c_1))
* T18: + =(c_4 a[[iti]t[ixito]](ind c_2))
* T19: All x:Iti + UTV(x)
* T20: All x:Iti + =(a[[iti]x[iti]t[iti]](comp x c_1) c_2)
* T21: - SUBSET(c_3 c_4)



A8,1 & T19,1  --> * R1:
  All x:I y,z:Iti - STU(y)  - S(x)
                + =(a[iti](a[[iti]x[iti]t[iti]](comp z y) x) a[iti](z a[iti](y x)))

                                    ⋮

R119,1 & A2,2   --> * R120:
  - STV(c_2)

R120,1 & T14,1  --> * R121:
  []

   q.e.d.

   Time Used for Refutation: 343 seconds
```

We see that the sorted representation is not only much easier to understand and to formulate, but – at least in this example – the MKRP system found the proof much faster than in the unsorted formulation (22 seconds compared to 343 seconds).

Chapter 7

Summary and Open Problems

> Wenn eine Aufgabe in ihrer vollen Allgemeinheit unlösbar scheint, so beschränke man sie vorläufig; dann wird vielleicht durch allmähliche Erweiterung ihre Bewältigung gelingen.
>
> *Gottlob Frege*, Begriffsschrift

One priority of this dissertation is the representation of mathematical knowledge. In chapter 3 we introduced a notion of higher-order logic – not based on the λ-calculus, but using Church's simple theory of types – and extended this logic to higher-order sorted logic by following notions of Schmidt-Schauss and Kohlhase.

Some mathematical construct are not easily expressed in our higher-order languages. For instance, we have not included specialized quantifiers such as $\exists!$ (*there exists exactly one*). (Such quantifiers are introduced by Mostowski in [101].) Another problem is that of incorporating meta-level descriptions into the object level. For example, in reasoning about linear equations, one may want to express that "$a \cdot x = b$" is a linear equation. This whole object is viewed then as a purely syntactic object, and in particular, it is not semantically evaluated, but only "spoken about". The realization of meta aspects in logic is of lifely interest in artificial intelligence, but most approaches separate the meta-level and the object level strictly. For further discussion see [31, 1, 130, 92, 111, 112, 99, 90, 47, 48, 49, 50]. We have had to omit these problems in this dissertation, but it is currently one of our research topics.

In chapter 4 we have used these logics to introduce a frame-based knowledge representation formalism, which allows for a conceptual representation of the factual knowledge of mathematics. We distinguish three different knowledge primitives, namely, "axiom", "definition", and "theorem". A frame should consist of all the knowledge that belongs to one concept. Thereby we obtain a conceptual description of axioms and definitions. It remains an open problem how to structure the theorems, because they generally interrelate many different concepts and so they do not belong to just one. A preliminary attempt of such a structure can be found in [71]. (Further efforts in this direction will be necessary finding proofs by analogy.) In order to have a clear semantics for the frame language we presented translations of the encoded knowledge into the underlaying higher-order logic. The main properties of this representation are:

- Definition 4.10 guarantees that concepts are introduced in a controlled way, that is, unknown concepts must not be used in the definiens of a definition or in a theorem.
- Theorem 4.24 states that definitions and theorems cannot hurt the consistency of a knowledge base, if the comprehension axioms (compare definition 3.22) are assumed.
- The frame approach is flexible for adding further informations like control information.

Other kinds of knowledge are not represented so far, a very important omission is the representation of *proofs*. If we want to reason about proofs, for instance, in order to find an analogous proof, we have to represent them in an adequate way, which goes beyond the usual first-order conception of a sequence of well-formed formulae. A representation and methods like those used in the field of proof presentation will be more adequate for such purposes [64, 65, 66, 86, 87, 88, 89]. Another important kind of knowledge that is not possible to represent right now, is that of *examples*, in particular "typical examples". Human mathematicians often *reason* on the model level. A model can often be given as a structure in a programming language. For instance, Common Lisp has the type "integer", which can be used directly as a model for the set $\mathbb{Z}$. Lenat chose a similar form of representation in his AM-system [82, 83, 84] to represent his concepts. A model can be used to find counterexamples of a hypothesis as it was done by Gelernter in his geometry theorem prover [46]: if a theorem is false in some model it cannot be true in the general case. In some cases it might be possible to find a proof for the model and this proof can then be used to guide the search for a proof at the general level (compare [55]), but little is known of how to automatize this. Ralf Koerstein currently implements the frame representation of this thesis in a standard SQL data base. (For applying data base techniques to AI system compare also [128].)

In chapter 5 we investigated the operationalization of our knowledge. The main goal was to use existing first-order theorem provers like the MKRP system.

Therefore we presented a whole class of translations from higher-order into first-order logic, which are sound (compare theorem 5.12). As stated in remark 5.17 these translations are bidirectional, that is, we can map the first-order proofs back to higher-order logic. In theorem 5.23 we showed that a particular translation is not only sound, but also complete with respect to a weak semantics. In consequence we can prove *in principle* everything that is provable in higher-order logic via translations into first-order logic. The main drawback is however the need for the so-called comprehension axioms (compare definition 3.22) for *truly higher-order theorems* (compare definition 3.24). The distinction between "truly higher-order theorems" and "essentially first-order theorems" made explicit the difference between theorems, which are difficult, *because* they are higher-order, and theorems, which are formulated in a higher-order syntax, but are essentially first-order. There is some support for the opinion that this is also the borderline between theorems that should be proved using a higher-order theorem prover and those that should be proved using a translation and a first-order theorem prover. The results stated for unsorted higher-order logic are generalized to sorted higher-order logic.

In chapter 6 we presented some examples for translations and gave some intuition for the sensitivity of the behaviour of a theorem prover to the actual presentation of theorems.

A further open problem is to find a characterization of sound morphisms from higher-order into first-order logic in order to have the greatest possible flexibility in formulating problems, that is, to know the version space of sound morphisms. There are some results, but no general theory [72]. Furthermore we need good heuristics for finding an adequate translation [73]. A prototypical realization of the translations presented in this thesis has been implemented by Dirk Schröder; a revised version will be integrated into the Ω-MKRP system [121].

This work contributes to tools that are indispensable in a computer-based system supporting mathematicians in finding proofs. The most exciting problem in the field of automated theorem proving is in my opinion how to combine human-like reasoning and machine-oriented theorem proving by finding and operationalizing high-level heuristics like for instance those proposed by Pólya. I hope that the methods developed in this thesis will be fruitful in attacking this long term goal.

Acknowledgement

Above all I like to thank Prof. JÖRG H. SIEKMANN for inspiring this work, for teaching me AI, for many motivating talks, for giving generously of his time, and last but not least for providing a grant and then a job, which made this work possible. Many improvements of this thesis are due to his suggestions.

I like to thank Prof. KLAUS MADLENER for his constructive critique and the suggestion to present more of the practical experience, what lead to chapter 6.

Of great influence to this work has been a higher-order study group comprised of FRANZ BAADER, MICHAEL KOHLHASE, JÖRG H. SIEKMANN, and myself. In particular we read PETER B. ANDREWS' book on higher-order logic [3], of which I learned most about higher-order logic.

I also like to thank all my colleagues of the MKRP group for clarifying and inspiring talks. Especially I like to thank NORBERT EISINGER for explaining numerous details of automated theorem proving and for acquainting me with the MKRP system. For practical support and for many discussions, also on rather vague ideas, I want to thank AXEL PRÄCKLEIN. Many unclear ideas could already be eliminated in these discussions. For many clarifying talks upon sorted higher-order logic I thank MICHAEL KOHLHASE, they resulted in particular in a more readable form of the corresponding parts of this thesis. For the strenuous work of reading drafts of this thesis I have to thank MICHAEL KOHLHASE, DAVID POWERS, SUSAN POWERS, and AXEL PRÄCKLEIN. Their suggestions lead to several improvements. Of course, all remaining errors are mine.

DIRK SCHRÖDER and RALF KOERSTEIN worked on building up a knowledge base and used this knowledge for proving theorems with the MKRP system. Since at that time the theory was not yet clear this work was not always easy, but helped much in developing our theory.

During this work I learned to appreciate the value of libraries in the departments of mathematics and computer science. Thanks to all who have built them up.

For more than one year this research was financed by a grant of the Land Rheinland/Pfalz. I greatfully acknowledge this support.

References

[1] Luigia Aiello and Richard W. Weyhrauch. Using meta-theoretic reasoning to do algebra. In Wolfgang Bibel and Robert Kowalski, editors, *Proceedings of the 5th CADE*, pages 1–13, Les Arcs, France, 1980. Springer Verlag, Berlin, Germany. LNCS 87.

[2] Peter B. Andrews. Theorem proving via general matings. *Journal of the ACM*, **28**:193–214, 1981.

[3] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic Press, Orlando, Florida, USA, 1986.

[4] Peter B. Andrews, Sunil Issar, Dan Nesmith, and Frank Pfenning. The TPS theorem proving system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 641–642, Kaiserslautern, Germany, July 1990. Springer Verlag, Berlin, Germany. LNAI 449.

[5] Aristotle. Organon. Eugen Rolfes, Hamburg.

[6] Johan van Benthem and Kees Doets. Higher order logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, chapter I.4, pages 275–329. D.Reidel Publishing Company, Dodrecht, Netherlands, 1983. Volume I: Elements of Classical Logic.

[7] Karel Berka and Lothar Kreiser. *Logik-Texte*, volume I. Akademie Verlag, Berlin, Germany, 1983.

[8] Paul Bernays. A system of axiomatic set-theory. *Journal of Symbolic Logic*, **2**:65–77, 1937.

[9] Paul Bernays. A system of axiomatic set-theory. *Journal of Symbolic Logic*, **6**:1–17, 1941.

[10] Wolfgang Bibel. *Automated Theorem Proving.* Vieweg, Wiesbaden, Germany, 1982.

[11] SUSANNE BIUNDO, BIRGIT HUMMEL, DIETER HUTTER, and CHRISTOPH WALTHER. The Karlsruhe induction theorem proving system. In Jörg H. Siekmann, editor, *Proceedings of the 8th CADE*, pages 672–674, Oxford, United Kingdom, July 1986. Springer Verlag, Berlin, Germany.

[12] STEPHEN BLAMEY. Partial logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, chapter III.1, pages 1–70. D.Reidel Publishing Company, Dodrecht, Netherlands, 1986. Volume III: Alternatives to Classical Logic.

[13] W. W. BLEDSOE. Some thoughts on proof discovery. In Robert Keller, editor, *Proceedings of the IEEE Symposium on Logic Programming*, pages 2–10, Salt Lake City, Utah, USA, September 1986. IEEE Computer Society.

[14] GEORGE BOOLE. *The Mathematical Analysis of Logic.* Macmillan, Barclay, & Macmillan, Cambridge, United Kingdom; reprinted by Basil Blackwell, Oxford, United Kingdom, 1965, 1847.

[15] NICOLAS BOURBAKI. *Théorie des ensembles.* Éléments de mathématique, Fascicule 1. Hermann, Paris, France, 1954.

[16] ROBERT BOYER, EWING LUSK, WILLIAM MCCUNE, ROSS OVERBEEK, MARK STICKEL, and LAWRENCE WOS. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, **2**:287–327, 1986.

[17] ROBERT S. BOYER and J STROTHER MOORE. *A Computational Logic.* Academic Press, New York, USA, 1979.

[18] RONALD J. BRACHMAN. On the epistemological status of semantic networks. In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, chapter 10, pages 191–215. Morgan Kaufmann, 1985, San Mateo, California, USA, 1979. also in: *Associative Networks: Representation and Use of Knowledge by Computers*, p.3–50, N. V. Findler, editor, New York, USA, Academic Press.

[19] RONALD J. BRACHMAN and JAMES G. SCHMOLZE. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, **9**:171–216, 1985.

[20] BISHOP BROCK, SHAUN COOPER, and WILLIAM PIERCE. Analogical reasoning and proof discovery. In Ewing Lusk and Ross Overbeek, editors, *Pro-*

ceedings of the 9th CADE, pages 454–468, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany. LNCS 310.

[21] Brockhaus Enzyklopädie. 19.Auflage. F.A. Brockhaus, Mannheim, Germany, 1986.

[22] Luitzen Egbertus Jan Brouwer. Intuitionism and formalism. *Bull. Amer. Math. Soc.*, **20**:81–96, 1914.

[23] Luitzen Egbertus Jan Brouwer. Zur Begründung der intuitionistischen Mathematik. *Mathematische Annalen*, **93**:244–257, 1925.

[24] Frank Malloy Brown. Towards the automation of set theory and its logic. *Artificial Intelligence*, **10**:281–316, 1978.

[25] Nicolaas Govert de Bruijn. AUTOMATH, a language for mathematics. Séminaire de Mathématiques Supérieures 52, Département de Mathématiques, Université de Montréal, Montréal, Canada, 1973.

[26] Nicolaas Govert de Bruijn. AUTOMATH – ein Projekt zur Kontrolle von Mathematik. Talk given at Innsbrucker Mathematikertag, 1974. German translation of "The AUTOMATH Mathematics Checking Project", Proceedings Symp. APLASM, Vol. I, P. Braffort, editor, Orsay, France, 1973.

[27] Nicolaas Govert de Bruijn. A survey of the project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, London, United Kingdom, 1980.

[28] Alan Bundy. Discovery and reasoning in mathematics. In Aravind Joshi, editor, *Proceedings of the 9th IJCAI*, pages 1221–1230, Los Angeles, California, USA, May 1985. Morgan Kaufmann, San Mateo, California, USA.

[29] Alan Bundy. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th CADE*, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany. LNCS 310.

[30] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The OYSTER-CLAM system. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany. LNAI 449.

[31] ALAN BUNDY and BOB WELHAM. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. In Wolfgang Bibel and Robert Kowalski, editors, *Proceedings of the 5th CADE*, pages 25–38, Les Arcs, France, 1980. Springer Verlag, Berlin, Germany. LNCS 87.

[32] ALONZO CHURCH. An unsolvable problem of elementary number theory. *American Jounal of Mathematics*, **58**:345–363, 1936.

[33] ALONZO CHURCH. A formulation of the simple theory of types. *Journal of Symbolic Logic*, **5**:56–68, 1940.

[34] R.L. CONSTABLE, S.F. ALLEN, H.M. BROMLEY, W.R. CLEAVELAND, J.F. CREMER, R.W. HARPER, D.J. HOWE, T.B. KNOBLOCK, N.P. MENDLER, P PANANGADEN, J.T. SASAKI, and S.F. SMITH. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.

[35] JOHN CORCORAN. Categoricity. *History and Philosophy of Logic*, **1**:187–207, 1980.

[36] HASKELL B. CURRY, R. FEYS, and W. CRAIG. *Combinatory Logic*, volume 1. North-Holland, Amsterdam, The Netherlands, 1958.

[37] MARTIN DAVIS. A computer program for Presburger's algorithm. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning – Classical Papers on Computational Logic, Volume 1, 1957–1966*, pages 41–48. Springer Verlag, Berlin, Germany, 1983, 1957. reprint.

[38] MARTIN DAVIS. The prehistory and early history of automated deduction. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning – Classical Papers on Computational Logic, Volume 1, 1957–1966*, pages 1–28. Springer Verlag, Berlin, Germany, 1983.

[39] RENÉ DESCARTES. Regulae ad Directionem Ingenii. first published 1701; Latin original with French translation, Bovin et Cie., Paris, France, 1933; German translation in: René Descartes – Ausgewählte Schriften, Fischer, 1986, Frankfurt, Germany, 1628/29.

[40] PETER DEUSSEN. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher.* Springer Verlag, Berlin, Germany, 1971.

[41] NORBERT EISINGER and HANS JÜRGEN OHLBACH. The Markgraf Karl Refutation Procedure (MKRP). In Jörg H. Siekmann, editor, *Proceedings of*

the 8th CADE, pages 681–682, Oxford, United Kingdom, July 1986. Springer Verlag, Berlin, Germany.

[42] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, San Diego, California, USA, 1972.

[43] William M. Farmer. A partial functions version of Church's simple theory of types. Technical Report M88-52, Revision 1, The MITRE Corporation, Bedford, Massachusetts, USA, February 1990.

[44] Adolf Abraham Fraenkel. Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre. *Mathematische Annalen*, **86**:230–237, 1922.

[45] Gottlieb Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879. reprint in: Begriffsschrift und andere Aufsätze, J. Angelelli, editor, Hildesheim. See also in Logiktexte, Karel Berka, Lothar Kreiser, editors, pages 82–112.

[46] H. Gelernter. Realization of a geometry theorem-proving machine. In *Proceedings of the International Conference on Information Processing, UNESCO*, 1959.

[47] Fausto Giunchiglia and Luciano Serafini. Multilanguage first order theories of propositional attitudes. IRST-Technical Report 9001-02, Istituto per la Ricerca Scientifica e Tecnologica, Trento, Italy, 1990.

[48] Fausto Giunchiglia and Paolo Traverso. Plan formation and execution in an uniform architecture of declarative metatheories. IRST-Technical Report 9003-12, Istituto per la Ricerca Scientifica e Tecnologica, Trento, Italy, 1990.

[49] Fausto Giunchiglia and Paolo Traverso. Reflective reasoning with and between a declarative metatheory and the implementation code. IRST-Technical Report 9012-03, Istituto per la Ricerca Scientifica e Tecnologica, Trento, Italy, 1990.

[50] Fausto Giunchiglia and Paolo Traverso. A system for multi-level mathematical reasoning. IRST-Technical Report 9011-12, Istituto per la Ricerca Scientifica e Tecnologica, Trento, Italy, 1990.

[51] Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, **37**:349–360, 1930.

[52] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, **38**:173–198, 1931.

[53] Kurt Gödel. *The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, New Jersey; eighth printing 1970, 1940.

[54] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* LNCS 78. Springer Verlag, Berlin, Germany, 1979.

[55] Jacques Hadamard. *The Psychology of Invention in the Mathematical Field.* Dover Publications, New York, USA; edition 1949, 1944.

[56] Patrick J. Hayes. The logic of frames. In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, chapter 14, pages 287–295. Morgan Kaufmann, 1985, San Mateo, California, USA, 1979. also in: *Frame Conceptions and Text Understanding*, p.46–61, D. Metzing, editor, Berlin, Germany, Walter de Gruyter.

[57] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, **15**:81–91, 1950.

[58] Lawrence J. Henschen. N-sorted logic for automatic theorem-proving in higher-order logic. In *Proceedings of the Annual Conference of the ACM*, pages 71–81, Boston, Massachusetts, USA, 1972. Association for Computing Machinery, Washington, DC, USA, ACM Press, New York, USA.

[59] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Sci. Lett. Varsovie, Classe III sci. math. phys.*, **33**, 1930.

[60] Arend Heyting. *Intuitionism.* North-Holland Publishing Company, third edition 1971, Amsterdam, Netherlands, 1956.

[61] David Hilbert. *Grundlagen der Geometrie.* Teubner, 10th ed., 1968, Stuttgart, Germany, 1899.

[62] David Hilbert. Probleme der Grundlegung der Mathematik. *Mathematische Annalen*, **102**:1–9, 1930.

[63] CHRISTIAN HORN. Deduktive Programmierung. Seminarbericht 100, Sektion Mathematik, Humboldt-Universität zu Berlin, Berlin, Germany, 1988.

[64] XIAORONG HUANG. A human oriented proof presentation model. SEKI Report SR-89-11, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1989.

[65] XIAORONG HUANG. Proof transformation towards human reasoning style. In B. Metzing, editor, *Proceedings of the 13th GWAI*, pages 37–42, Eringerfeld, Germany, 18-22nd September 1989. Springer Verlag, Berlin, Germany.

[66] XIAORONG HUANG. Reference choices in mathematical proofs. In Luigia Carlucci Aiello, editor, *Proceedings of the 9th ECAI*, pages 720–725, Stockholm, Sweden, 6-10th August 1990. Pitman, London, Great Britain.

[67] GÉRARD HUET. A unification algorithm for the typed λ-calculus. *Theoretical Computer Science*, **1**:27–57, 1975.

[68] GÉRARD HUET, editor. *Logical Foundations of Functional Programming*. Addison-Wesley, Reading, Massachusetts, USA, 1990.

[69] DIETER HUTTER. Vollständige Induktion. In Karl Hans Bläsius and Hans-Jürgen Bürckert, editors, *Dedukionssysteme – Automatisierung des logischen Denkens*, chapter V, pages 153–172. Oldenbourg, München, Germany, 1987.

[70] L.S. VAN BENTHEM JUTTING. *Checking Landau's "Grundlagen" in the AUTOMATH System*, volume 83 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, Netherlands, 1979.

[71] MANFRED KERBER. Some aspects of analogy in mathematical reasoning. In Klaus P. Jantke, editor, *Analogical and Inductive Inference; International Workshop AII '89, Reinhardsbrunn Castle, GDR*, pages 231–242. Springer Verlag, Berlin, Germany, October 1989. LNAI 397.

[72] MANFRED KERBER. Towards a classification of sound morphisms from higher-order to first-order logic. SEKI Report, Universität des Saarlandes, Saarbrücken, Germany (in preparation), 1992.

[73] MANFRED KERBER and AXEL PRÄCKLEIN. Using tactics to reformulate formulae for resolution theorem proving. Presented on the 2nd International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, USA, also as SEKI Report, forthcoming, Universität des Saarlandes, Saarbrücken, Germany, January 1992.

[74] MORRIS KLINE. *Mathematics – The Loss of Certainty.* Oxford University Press, New York, USA, 1980.

[75] MICHAEL KOHLHASE. Order-sorted type theory I: Unification. SEKI Report SR-91-18 (SFB), Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1991.

[76] N.I. KONDAKOW. *Wörterbuch der Logik.* VEB Bibliographisches Institut, Leipzig, Germany; german edition, Erhard Albrecht, Günter Asser, editors, 1983.

[77] ROBERT KOWALSKI. A proof procedure using connection graphs. *Journal of the ACM*, **22**, 1975.

[78] LOTHAR KREISER, SIEGFRIED GOTTWALD, and WERNER STELZNER, editors. *Nichtklassische Logik*, volume I. Akademie Verlag, Berlin, Germany, 1990.

[79] EDMUND LANDAU. *Grundlagen der Analysis.* Wissenschaftliche Buchgesellschaft, Darmstadt, Germany; second edition, 1930. Reprint of the edition, Leipzig, 1970.

[80] GOTTFRIED WILHELM LEIBNIZ. Projet et essais pour arriver â quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer. In Karel Berka and Lothar Kreiser, editors, *Logiktexte*, chapter I.2, pages 16–18. Akademie-Verlag, german translation, 1983, Berlin, Germany, 1686.

[81] GOTTFRIED WILHELM LEIBNIZ. Historia et commendatio linguae characteristicae universalis quae simul sit ars inveniendi et judicandi. German translation in Gottfried Wilhelm Leibniz: Gott – Geist – Güte: Eine Auswahl aus seinen Werken, pages 163–173, Bertelsmann Verlag, Gütersloh, Germany, 1947, not dated.

[82] DOUGLAS B. LENAT. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* PhD thesis, AI Lab, Stanford University, Stanford, California, USA, 1976. AIM-286, STAN-CS-76-570, and Heuristic Programming Project Report HPP-76-8.

[83] DOUGLAS B. LENAT. The nature of heuristics. *Artificial Intelligence*, **19**:189–249, 1982.

[84] Douglas B. Lenat. Eurisko: A program that learns new heuristics and domain concepts. *Artificial Intelligence*, **21**:61–98, 1983.

[85] Douglas B. Lenat and Ramanathan V. Guha. *Building Large Knowledge-Based Systems – Representation and Inference in the CYC Project.* Addison-Wesley Publishing Company, Readings, Massachusetts, USA, 1990.

[86] Christoph Lingenfelder. Structuring computer generated proofs. In N.S. Sridharan, editor, *Proceedings of the 11th IJCAI*, pages 378–383, Detroit, Michigan, USA, 1989. Morgan Kaufman, San Mateo, California, USA.

[87] Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[88] Christoph Lingenfelder and Axel Präcklein. Presentation of proofs in an equational calculus. In Michel De Glas and Dov Gabbay, editors, *WOCFAI '91 – Proceedings of the First World Conference on the Fundamentals of Artificial Intelligence*, pages 313–321, Paris, France, 1991. Angkor.

[89] Christoph Lingenfelder and Axel Präcklein. Proof transformation with built-in equality predicate. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th IJCAI*, pages 165–170, Sydney, 1991. Morgan Kaufman, San Mateo, California, USA.

[90] J.W. Lloyd. Directions for meta-programming. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 609–617. ICOT, 1988.

[91] Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, **76**:447–470, 1915.

[92] Pattie Maes. Introspection in knowledge representation. In Benedict du Boulay, editor, *Proceedings of the 7th ECAI*, pages 256–269, Brighton, United Kingdom, July 1986. Volume 1.

[93] Karl Mark G Raph. The Markgraf Karl Refutation Procedure. Technical Report Memo-SEKI-MK-84-01, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, January 1984.

[94] Per Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175. North Holland, Amsterdam, Netherlands, 1982.

[95] David A. McAllester. *ONTIC – A Knowledge Representation System for Mathematics*. The MIT Press, Cambridge, Massachusetts, USA, 1989.

[96] William McCune. Otter 2.0. In Mark E. Stickel, editor, *Proceedings of the 10th CADE*, pages 663–664, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany. LNAI 449.

[97] Marvin Minsky. A framework for representing knowledge. In Patrick Henry Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, USA, 1975. also in: *Mind Design*, pages 95–128, J. Haugeland, editor, Cambridge, Massachusetts, USA, MIT-Press, 1981, and *Readings in Knowledge Representation*, pages 245–262, chapter 12, Ronald J. Brachman and Hector J. Levesque, editors, San Mateo, California, USA, Morgan Kaufmann, 1985.

[98] Gregory H. Moore. Beyond first-order logic: The historical interplay between mathematical logic and axiomatic set theory. *History and Philosophy of Logic*, **1**:95–137, 1980.

[99] Katharina Morik. Anything you can do – I can do meta. KIT-Report 40, Fachbereich Informatik, Technische Universität Berlin, Berlin, Germany, November 1986.

[100] Andrzej Mostowski. An undecidable arithmetical statement. *Fundamenta Mathematicae*, **36**:143–164, 1949.

[101] Andrzej Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, **44**:12–36, 1957.

[102] John von Neumann. Die Axiomatisierung der Mengenlehre. *Mathematische Zeitschrift*, **27**:669–752, 1928.

[103] Allen Newell. The heuristic of George Polya and its relation to artificial intelligence. Technical Report CMU-CS-81-133, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, July 1981. also in Rudolf Groner, Marina Groner and Walter F Bishoof, editors, Methods of Heuristics, Lawrence Erlbaum, Hillsdale, New Jersey, USA, 195–243.

[104] ALLEN NEWELL, CLIFF SHAW, and HERBERT SIMON. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings of the 1957 Western Joint Computer Conference*, New York, USA, 1957. McGraw-Hill. reprinted in Computers and Thougt, Edward A. Feigenbaum, Julian Feldman, editors, New York, USA, 1963.

[105] NILS J. NILSSON. *Principles of Artificial Intelligence.* Morgan Kaufman, San Mateo, California, USA, 1980.

[106] ARNOLD OBERSCHELP. Untersuchungen zur mehrsortigen Quantorenlogik. *Mathematische Annalen*, **145**:297–333, 1962.

[107] HANS JÜRGEN OHLBACH. Context logic. SEKI Report SR-89-08, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1989.

[108] DOMINIQUE PASTRE. Automatic theorem proving in set theory. *Artificial Intelligence*, **10**:1–27, 1978.

[109] LAWRENCE C. PAULSON. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, pages 361–386, 1990.

[110] GUISEPPE PEANO. Sul concetto di numero. *Rivista di Mat*, **1**:87–102, 1891.

[111] DONALD PERLIS. Languages with self-reference I: Foundations (or: We can have everything in first-order logic!). *Artificial Intelligence*, **25**:301–322, 1985.

[112] DONALD PERLIS. Languages with self-reference II: Knowledge, belief, and modality. *Artificial Intelligence*, **34**:179–212, 1988.

[113] GEORGE PÓLYA. *How to Solve It.* Princeton University Press, Princeton, New Jersey, USA, also as Penguin Book, 1990, London, United Kingdom, 1945.

[114] GEORGE PÓLYA. *Mathematics and Plausible Reasoning.* Princeton University Press, Princeton, New Jersey, USA, 1954. Two volumes, Vol.1: Induction and Analogy in Mathematics, Vol.2: Patterns of Plausible Inference.

[115] GEORGE PÓLYA. *Mathematical Discovery – On understanding, learning, and teaching problem solving.* Princeton University Press, Princeton, New Jersey, USA, 1962/1965. Two volumes, also as combined edition, 1981, John Wiley and Sons, New York, USA.

[116] John Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, **12**:23–41, 1965.

[117] Manfred Schmidt-Schauss. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. LNAI 395. Springer Verlag, Berlin, Germany, 1989.

[118] Jörg H. Siekmann. Geschichte und Anwendungen. In Karl Hans Bläsius and Hans-Jürgen Bürckert, editors, *Dedukionssysteme – Automatisierung des logischen Denkens*, chapter I, pages 3–21. Oldenbourg, München, Germany, 1987.

[119] Jörg H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 1987.

[120] Jörg H. Siekmann and Graham Wrightson. *Automation of Reasoning – 1. Classical Papers on Computational Logic 1957–1966*, volume I. Springer Verlag, Berlin, Germany, 1983.

[121] Jörg H. Siekmann et al. Ω-MKRP. SEKI Report, forthcoming, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1992.

[122] Albert Thoralf Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze. *Skrifter-Vidensk*, **I**:1–36, 1919.

[123] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, **1**:333–356, 1985.

[124] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia philosophia*, **1**:261–405, 1936.

[125] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Second Series*, **42**:230–265; **43**:544–546, 1937.

[126] Bartel L. van der Waerden. Wie der Beweis der Vermutung von Baudet gefunden wurde. *Abh. Math. Sem. Univ. Hamburg*, **28**:6–15, 1964.

[127] Bartel L. van der Waerden. *Algebra I*, volume 12 of *Heidelberger Taschenbücher*. Springer Verlag, Berlin, Germany, eighth edition, 1971.

[128] INGRID WALTER. *Datenbankgestützte Repräsentation und Extraktion von Episodenbeschreibungen aus Bildfolgen*, volume 213 of *Informatik Fachberichte KI*. Springer Verlag, Berlin, Germany, 1989.

[129] CHRISTOPH WALTHER. Ein mehrsortiger Resolutionskalkül mit Paramodulation. Interner Bericht 35/82, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1982.

[130] RICHARD W. WEYHRAUCH. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, **13**:133–170, 1980.

[131] ALFRED NORTH WHITEHEAD and BERTRAND RUSSELL. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, Great Britain; second edition, 1910.

[132] LARRY WOS, ROSS OVERBEEK, EWING LUSK, and JIM BOYLE. *Automated Reasoning – Introduction and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1984.

[133] ERNST ZERMELO. Untersuchungen über die Grundlagen der Mengenlehre. I. *Mathematische Annalen*, **65**:261–281, 1908.

Notation

Index of Subjects

Index of Names

Curriculum Vitae

4.6.1956	geboren in Merzig an der Saar als Sohn von Anneliese und Gerhard Kerber
1963	Einschulung in die Volksschule am Kreuzberg in Merzig
1967	übergewechselt an das Staatliche Gymnasium am Stefansberg in Merzig
24.5.1976	Abitur
1.10.1976	Studienbeginn an der Universität Kaiserslautern: Lehramt Mathematik/Physik
7.8.1978	Zwischenprüfung Mathematik
21.11.1978	Zwischenprüfung Physik
14.10.1981	Vordiplom Mathematik
21.9.1983	Diplom Mathematik
3.10.1983–31.1.1985	Zivildienst beim Roten Kreuz Kaiserslautern-Stadt, ab 30.1.1984 als Rettungshelfer
5.12.1985	1. Staatsexamen Lehramt an Gymnasien in Mathematik und Physik
seit Frühjahr 1985	Mitarbeit bei Prof. Siekmann, teilweise als Stipendiat, teilweise als wissenschaftlicher Mitarbeiter des SFB 314 "Künstliche Intelligenz – Wissensbasierte Systeme" und des Fachbereichs Informatik der Universität Kaiserslautern
seit 2.5.1989	verheiratet mit Barbara Schütt-Kerber