Equality and Extensionality in Automated Higher-Order Theorem Proving

Christoph Benzmüller

SEKI Report SR-99-08

Also published as: Dissertation at the Saarland University, 1999
# Contents

1 Introduction ................................................. 1
   1.1 Motivation .............................................. 1
   1.2 Higher-Order Logic ...................................... 3
   1.3 Higher-Order Model Existence ............................. 5
   1.4 Traditional and Extensional Higher-Order Resolution .... 8
   1.5 Adding Primitive Equality ................................. 10
   1.6 The LEO System ........................................... 12

2 Syntax and Semantics of Higher-Order Logic .................. 14
   2.1 Syntax and Preliminaries .................................. 14
   2.2 Pre-$\Sigma$-Structures ..................................... 17
   2.3 $\Sigma$-Structures ......................................... 18
   2.4 Functional $\Sigma$-structures ............................... 20
   2.5 $\Sigma$-Models ............................................. 22
   2.6 Leibniz Equality .......................................... 23
   2.7 Primitive Equality ....................................... 26
   2.8 A Note on Defined and Primitive Equality ................. 27

3 Higher-Order Model Existence ................................ 30
   3.1 Abstract Consistency ..................................... 30
   3.2 Hintikka Sets ............................................. 33
   3.3 Primitive Equality ....................................... 36
   3.4 Model Existence .......................................... 39

4 Extensional Higher-Order Resolution: $\mathcal{ER}$ ........... 42
   4.1 A Review of $\mathcal{HORES}$ and $\mathcal{ER}$ ............... 42
   4.2 Basic Definitions ......................................... 47
   4.3 Lifting Properties ........................................ 50
   4.4 Completeness ............................................. 52
   4.5 Theorem Equivalence ...................................... 54

5 Extensional Higher-Order Paramodulation: $\mathcal{EP}$ ......... 57
   5.1 A Naive and Incomplete Adaptation of Paramodulation .... 57
   5.2 Positive Extensionality Rules ............................... 61
   5.3 Basic Definitions ......................................... 61
   5.4 Lifting Properties ........................................ 62
   5.5 Completeness ............................................. 63
   5.6 Theorem Equivalence ...................................... 71
6 Extensional Higher-Order RUE-Resolution: E\textit{RUE} 72
6.1 Resolution on Unification Constraints .................................................. 72
6.2 Basic Definitions ....................................................................................... 72
6.3 Lifting Properties ...................................................................................... 73
6.4 Completeness ............................................................................................. 74
6.5 Theorem Equivalence ................................................................................ 79

7 The LEO-System ......................................................................................... 80
7.1 Basic Data-Structures and Algorithms .................................................... 81
7.2 Extended SOS Architecture and Main Loop ........................................... 82
7.2.1 Problems ............................................................................................ 82
7.2.2 Extended SOS Architecture ............................................................... 83
7.2.3 LEO's heuristically guided Main Loop ................................................ 84
7.2.4 An illustrated Example ......................................................................... 86
7.2.5 Realisation of the Challenges and the main Sources of Incompleteness ... 87
7.3 New Insights gained from LEO ............................................................... 89
7.3.1 Extensionality and Term indexing ....................................................... 89
7.3.2 Extensionality and Subsumption ......................................................... 89
7.4 Case Study ............................................................................................... 90
7.5 Additional Features of LEO ..................................................................... 93
7.5.1 The Interactive Theorem Prover LEO .................................................. 93
7.5.2 LOUI as a Graphical Interface for LEO ............................................... 93
7.5.3 Integration to \textit{OMEGA} .................................................................... 93

8 Examples ..................................................................................................... 96
8.1 Extensionality in \textit{ER} ............................................................................ 96
8.2 Decomposition in \textit{ER} .......................................................................... 98
8.3 Leibniz Equality and Alternative Definitions in \textit{ER} ............................... 99
8.4 Reasoning about Sets with Leibniz Equality .......................................... 101
8.5 Positive Extensionality Rules in \textit{EP} and \textit{RUE} ...................................... 101
8.6 Comparing \textit{EP} and \textit{RUE} .................................................................. 103
8.7 Raised Questions ..................................................................................... 109

9 Applications and Related Work ............................................................... 111
9.1 Cooperation and Joint Work .................................................................... 111
9.2 Abstract Consistency and Model Existence .......................................... 111
9.3 Extensional Higher-Order Resolution .................................................... 112
9.4 Primitive Equality in Higher-Order Theorem Proving ............................. 114
9.5 Theorem Provers for Higher-Order Logic .............................................. 115
9.6 Examples ............................................................................................... 115

10 Conclusion and Outlook .......................................................................... 117

Bibliography ............................................................................................... 118
Index .......................................................................................................... 129
Abstract

This thesis focuses on equality and extensionality in automated higher-order theorem proving based on Church’s simply typed $\lambda$-calculus (classical type theory).

First, a landscape of various semantical notions is presented that is motivated by the different roles equality adopts in them. Each of the semantical notions in this landscape — including Henkin semantics — is then linked with an abstract consistency principle that can be employed for analysing the connection between syntax and semantics of higher-order calculi.

Apart from this proof theoretic tools, the main contributions of this thesis are the three new calculi $ER$ (extensional higher-order resolution), $EP$ (extensional higher-order paramodulation) and $ERUE$ (extensional higher-order RUE-resolution) which improve the mechanisation of defined and primitive equality in classical type theory. In contrast to the refutation approaches for classical type theory developed so far, these calculi reach Henkin completeness without requiring additional extensionality axioms. The key idea is to allow for recursive calls from higher-order unification to the overall refutation search.

Calculus $ER$, which in contrast to $EP$ and $ERUE$, considers equality only as a defined notion, has been implemented in the theorem prover Li60 and the suitability of this approach for proving simple theorems about sets has been demonstrated in a case study.

Quo facto, quando orientur controversiae, non magis disputatim opus erit inter duos philosophos, quam inter duos Computistas. Sufficit enim calamos in manus sumere sedere ad abacos, et sibi mutuo (accito si placet amico) dicere: calculamus.

Gottfried Wilhelm Leibniz

Chapter 1

Introduction

1.1 Motivation

The dream of formalising and mechanising mathematical reasoning — which also motivates this thesis — reaches back to Gottfried Wilhelm Leibniz. He conceived a *lingua characteristic* and a *calculus ratiocinator*, i.e., a most general framework to mechanise human reasoning, which was certainly inspired by his own contributions to the mechanisation of simple arithmetical operations. He developed and realised a mechanical calculator capable of multiplication. But Leibniz' contributions to the mechanisation of mathematical reasoning in general did not go beyond elementary stages and it was not until the end of the 19th century that the field of modern mathematical logic was born. Taking the important contribution of the ancient Greeks (mostly Aristoteles) into account, who already investigated the laws of human thought and developed a theory of well-chosen axioms and rules, it is probably better to say 'It was not until the 19th century that modern mathematical logic was reborn'. It was mainly the work of George Boole (1815-1864), Gottlob Frege (1848-1925) and Bertrand Russel (1872-1970) that stimulated the new and deep interest of many researchers in the field of mathematical logic. Frege’s fundamental *Begriffsschrift* is described by Davis [Dav83] *not only as the direct ancestor of contemporary systems of mathematical logic but also as the ancestor of all formal languages, including computer programming languages.*

Another milestone in the formalisation of mathematics to be mentioned is Hilbert’s ambitious program at the very beginning of this century [Hil04, Hil27], which aimed at the complete development of modern mathematics in a formal system. In the early 30’s results came fast. Whereas Kurt Gödel, Jacques Herbrand and Thoralf Skolem proved the completeness of the (first-order) predicate calculus in 1930 [Göd30, Her30, Sko28] — i.e., every valid formula in the language of the predicate calculus is derivable from its axioms — it was Gödel who showed in his famous incompleteness theorems [Göd31] that it is impossible to develop a generally complete calculus that mechanisms mathematical reasoning. More precisely, Gödel showed that as soon as a system is rich enough to encode Peano arithmetic, one can construct sentences that are valid in Peano arithmetic but which are not derivable in the system itself.

Caused by the development of electronic computers in the 40’s and 50’s disappointment gradually gave away to attempts of developing and implementing proof procedures in practice. It took quite a few years until J.A. Robinson achieved a first important break-through with his resolution approach in 1965 [Rob65]. The most important improvement of this approach compared to former ones is that in order to prove a theorem it tries to refute the negated theorem in a goal directed way, thereby employing first-order unification as a powerful filter instead of simply enumerating the Herbrand universe like most earlier methods. Robinson’s ideas are still employed in many state of the art theorem provers such as Otter [MW97, McC94], Eqs (which recently solved the Robbins Problem [McC97b]), or the superposition based prover SPASS [WGR96]. Even tableaux based provers like Protein [BF94] or Setheo [GLMS94] are rather closely related to the resolution approach and unification became an essential (filtering) tool for the whole field.
CHAPTER 1. INTRODUCTION

Unfortunately rather few pioneers dared to tackle the mechanisation of higher-order logic based on the simply typed \( \lambda \)-calculus — also called classical type theory. For instance, Robinson presents in [Rob68, Rob69] a higher-order proof procedure based on the tableaux idea that itself employs many ideas from the calculi given in [Sch60] and [Tak53]. The most important works to be mentioned are Peter Andrews’ investigation of higher-order resolution [And71], Jensen and Pietrowski’s approach [JP72] and especially Gerard Huet’s constrained resolution approach [Hue72, Hue73a]. It is well known that one of the great challenges for the mechanisation of classical type theory is the undecidability of higher-order unification [Luc72, Luc73b, Gol84]. Whereas Andrews’ resolution approach still avoids unification (and instead employs an enumeration of the universe), Huet’s constrained resolution approach [Hue72] solves the problem by encoding the particular unification problems as unification constraints and by delaying the application of higher-order unification until the end of a refutation. Huet’s approach additionally gains from the higher-order pre-unification algorithm [Hue75] which avoids the guessing aspects of full higher-order unification (for early non-complete approaches to higher-order unification see [Gou66, Dar71, Ern71]; the first complete approaches are presented in [Hue72] and [JP72]) and which fortunately turned out to be sufficient within a refutation approach.

At present, the most powerful higher-order theorem prover is the TPS-system [ABP96, AINP90] developed at Carnegie Mellon University which is based on the unification approach [And76, And81, And80, Bib83]. This system and the Leo-prover described in this thesis demonstrate the practical feasibility of automated higher-order theorem proving — at least for simple mathematical theorems.

All approaches to automated higher-order theorem proving mentioned above necessarily lack completeness with respect to the intuitive notion of standard semantics as shown by Gödel in 1931 [Göd31]. Instead modern calculi (and systems) aim at completeness with respect to Henkin semantics, which has been invented by Leon Henkin [Hen50, Hen96] and is known as the most general notion of semantics for classical type theory so that complete calculi are possible. Henkin semantics, in [And71] also called general models, thus became the theoretical surveyor’s wooden rod for all calculi in this field.

An interesting aspect of classical type theory is that equality is definable in a very natural way (if the underlying notion of semantics is strong enough). For instance, one can express the Leibniz principle of equality — two things are equal, if they have the same properties — very easily in classical type theory:

\[ \equiv^\alpha := (\lambda x_{\alpha} y_{\alpha} \forall P_{\alpha \rightarrow \alpha} P \ X \Rightarrow P \ Y) \]

In this sense, all approaches mentioned above provide for a very natural equality treatment. But in order to ensure Henkin completeness, they all (including the more recent calculi [Wol93], [Koh94b] and [Koh93]) require the extensionality axioms of Leibniz equality (i.e. the infinitely many functional extensionality axioms\(^2\) and the axiom for Boolean extensionality\(^3\)) to be added to the search space. Furthermore, there are no special techniques for the mechanisation of equality reasoning as in first-order theorem proving.

We can now precisely formulate the goals of this thesis, which are:

1. The clarification of the role of equality and extensionality in automated higher-order theorem proving.
2. The development of proof techniques for analysing higher-order calculi with respect to Henkin completeness.

---

\(^1\) Note that this formula describes an implication instead of an equivalence as one may expect. This is sufficient as one easily gets the other direction by contraposition when instantiating \( P \) with \(-P\).

\(^2\) The functional extensionality axioms express that two functions are equal, if they are equal on all arguments, and they are of form: \( \forall F_{\alpha \rightarrow \beta} \forall G_{\alpha \rightarrow \beta} (\forall X_{\beta} F X = G X) \Rightarrow F =^\beta G \). We need one axiom for each pair of types \( \alpha \) and \( \beta \).

\(^3\) The Boolean extensionality axiom expresses that on the domain of truth values, which contains exactly two truth values in standard or Henkin semantics, the equality relation and equivalence relation coincide: \( \forall A, B, (A =^\alpha B) \equiv (A \equiv B) \).
3. The development of a Henkin complete resolution calculus that avoids the extensionality axioms in the search space.

4. The development of Henkin complete approaches to primitive equality in higher-order resolution based theorem proving.

5. The demonstration of the practicability of these approaches via an implementation.

In the remaining parts of the introduction we will illustrate the different aspects of this thesis in more detail.

1.2 Higher-Order Logic

A higher-order logic is any simply typed logical system that allows quantification over function and predicate variables. It was Bertrand Russell [Rus02, Rus03] who first pointed out in 1902 that in connection with the comprehension principles they may allow for paradoxes. The most prominent example is the set of all non-self-containing sets (also called Russell's paradox). As a possible solution Russell suggested a few years later in [Rus08] a theory of types as a basis for the formalisation of mathematics that differentiates between objects and sets (or functions) consisting of these kinds of objects. This idea was also taken up by Alonzo Church in 1940, who invented the simply typed λ-calculus [Chu40] in order to prevent such paradoxes in the untyped λ-calculus, which he developed with Schönfinkel and Curry ten years earlier. Needless to mention that typed and untyped λ-calculi play an important or even central role in many research fields of modern computer science. Consequently there are several modern textbooks for the typed and untyped λ-calculus available and we refer to [Bar92, And86, HS86, Bar84] for details.

The avoidance of paradoxes like Russell's paradox is also a main reason why we employ a logic based on Church's simply typed λ-calculus [Chu40] — i.e., classical type theory — in this thesis. There are certainly other approaches, e.g., Zermelo-Fraenkel [Zer08, Frä22b, Frä28] or von Neumann's [Neu25] set theory, that solve the paradox problems and they are often more popular among mathematicians as a basis for the formalisation of mathematics.

An argument for type theory, however, is that it allows for more natural and intuitive problem formulations as well as solutions (see the introduction in Andrews' textbook [And86]). Furthermore, the choice of classical type theory has additional advantages as the λ-binding construct in combination with the λ-conversion rules in these languages have the effect that the type restricted comprehension axioms become derivable (see also [And86]). The comprehension axioms are of the form \( \exists U_{\alpha \to \beta} \forall X. (U \cdot X^{n}) = B_{\beta} \) for an arbitrary term \( B \) such that the variable \( U \) does not occur free in \( B \), types \( \alpha \) and \( \beta \), and \( n \geq 0 \) and their intention is to guarantee for each expression \( B \) the existence of the functions (or sets) referred by \( U_{\alpha \to \beta} \). For instance, if \( \beta \) is the type of a truth value the comprehension principle asserts that for any respective formula \( B \) there exists a set \( u \) (referred to by the variable \( U \)) which contains exactly all those elements \( v^{\alpha} \) (referred to by variables \( \sqrt{\alpha} \) for which \( B \) evaluates to true. But fortunately the required sets and functions \( u \) can easily be directly described in the λ-calculus by the term \( \lambda X^{n}. B \), so that the implicit requirement that each term in our language indeed has a denotation (which we call Denotatpflicht), already ensures their existence.

Another advantage of classical type theory is that the functional extensionality principle is, at least to some extent, automatically built into the language. This is due to the λ-conversion rules — especially the \( \eta \)-rule which expresses the convertibility of all terms of the form \( (\lambda X. A) X \) to \( A \) in case variable \( X \) does not occur free in \( A \) — and the existence of respective normal forms allows us to reduce all terms, e.g., to \( \beta \eta \)-normal form or \( \beta \eta \)-head-normal-form (see [Bar92, Bar84]). Also the standard higher-order unification or pre-unification algorithm [Hue75, SG89, Smy91] already applies the functional extensionality principle in a straightforward way in case at least one of the

---

These principles assure the existence of certain functions, cf. the type restricted comprehension axioms below.
CHAPTER 1. INTRODUCTION

two terms to be unified is a $\lambda$-abstraction.\(^5\)

Unfortunately the importance of the Boolean extensionality principle for the mechanisation of classical type theory has widely been overlooked. If one is interested only in syntactical term rewriting or narrowing in the simply typed $\lambda$-calculus (see for instance [Pre98, NP98, NM98b, Pre95, Pre94, Vo94, Wo93]) then the Boolean extensionality principle certainly is not of importance as the special type $o$ denoting the (exactly two valued) set of truth values and the logical connectives, which have fixed instead of arbitrary denotations, do not occur in the language. We clarify our notion of syntactical term rewriting: As already mentioned, higher-order (pre)-unification (as well as higher-order matching) indeed unifies terms modulo the functional extensionality principles. And in this sense unification and matching in the simply typed $\lambda$-calculus can certainly be seen as an $E$-unification algorithm, where $E$ is the theory defined by the functional extensionality axioms. But the functional extensionality principles are quite naturally and without much effort built-in into the $\lambda$-calculus as well as the traditional unification and matching algorithms. Thus, when we speak of syntactical higher-order unification or term rewriting we mean this in the latter sense and consider functional extensionality as automatically built-in.

The interest in syntactical higher-order term rewriting is motivated by potential applications in different fields such as functional programming or program verification. The main challenge recently is to develop suitable term-orderings in order to orientate and complete the equations of a rewriting system. All results on syntactical rewriting are therefore of great interest for the field of automated (or interactive) higher-order theorem proving, too, as there are many application domains in this fields in which syntactical rewriting can fruitfully be employed.

But in this thesis we will illustrate that equational theorem proving in classical type theory is generally much more challenging and complicated as pure syntactical term rewriting. Apart from the functional extensionality principle — which is to some extent addressed by syntactical term rewriting — the Boolean extensionality principle is of importance, too. For example, in equational theorem proving for classical type theory one is also interested in unifying — and rewriting — on extensionally equal terms like $(\forall x \cdot \text{red } x \cup \text{circle } x)$ and $(\forall x \cdot \text{circle } x \wedge \text{red } x)$ or $(\forall x \cdot A \lor \neg A)$ and $(\forall x \cdot B = B)$. It is quite obvious that the pure syntactical higher-order unification approach (i.e., syntactical modulo functional extensionality only) is much too weak to solve such unification problems. This example also illustrates that higher-order unification with respect to both extensionality principles requires the application of a general higher-order theorem prover in the sense of a embedded subsystem within higher-order unification in order to examine whether two terms of type $o$ are equal. This illustrates the challenging difference between simple syntactical rewriting, where at most the functional extensionality principle is taken into account, and equational reasoning in classical type theory where both extensionality principles are of importance.

Another aspect of higher-order logic to be clarified concerns the axiom of choice and the description operator. It is well known (see [Frä22a, Göd40]) that the axiom of choice is independent and consistent in set theory. A respective result for classical type theory is shown in [And72b]. The axiom of choice expresses that there exists a function $f$ that chooses exactly one element out of each set belonging to an arbitrary family of sets. This axiom — more precisely this axiom scheme — has a non-constructive character and, e.g., allows a very simple but non-constructive proof, first presented by Zermelo [Zer04] in 1904, for Cantor's conjecture that every set can be well ordered (for a modern discussion of the axiom of choice see also [Jec77]). As many mathematicians have objections to the non-constructive character of the axiom of choice we will not treat it as an imperative in our logic. Analogously we do not treat the description operator as automatically built-in. The description operator, which allows for the description of possibly non-existing objects like Santa Claus (note that introducing a constant in a signature for Santa Claus automatically requires its existence by the Denotatpflicht), was first examined by Alfred N. Whitehead and Bertrand Russel [WR10] (see also [Sco67, And72b] for a discussion).

\(^5\)Obviously, if one requires the unification terms to be in $\beta\eta$-normal form, any two terms of functional type must be $\lambda$-abstractions. If instead one considers only $\beta\eta$-head-normal form terms they not necessarily have to be $\lambda$-abstractions. In the latter case additional rules are probably needed in the unification calculus (see for instance the sorted higher-order unification calculus discussed in [Koh94b]).
1.3. HIGHER-ORDER MODEL EXISTENCE

We want to point out that although the axiom of choice is not built-in, non-constructive proofs are possible in our framework. We still consider a classical notion of higher-order logic in which, e.g., the tertium-non-datur principle is valid and proofs can be carried out indirectly.

This is in contrast to intuitionistic type theory [ML94, GJ98] which was invented by Martin Löf in the early 70’s to provide a formal foundation for constructive mathematics. Intuitionistic type theory has become a very important and active field in computer science and is successfully applied, for instance, for the verification of computer programs and languages. Hofmann discusses in [Ho97] that the extensionality principles are quite important even in intuitionistic type theory for the formalisation of mathematics, but they unfortunately cause major problems in the mechanisation.

Summing up, we employ in this thesis a classical higher-order logic based on the simply typed $\lambda$-calculus (classical type theory) and treat the axiom of choice and the description operator as optional. With respect to semantics we are interested in standard semantics and with respect to the completeness of our calculi we shall refer to Henkin semantics. In standard as well as in Henkin semantics both extensionality principles (the functional as well as the Boolean) are valid and for equational reasoning in classical type theory both principles need to be mechanised in an appropriate way. This differentiates higher-order equational reasoning, which is the topic of this thesis, from syntactical higher-order term rewriting.

In this thesis we shall differentiate between five notions of equality: If we define a concept we use $\equiv$ (e.g., let $\mathcal{D} := \{ \mathcal{T}, \mathcal{F} \}$ and $\equiv$ represents meta-equality. We denote the equality relation as an object of our semantical domains with $q$; note that there is at most one $q^\alpha$ in each domain $\mathcal{D}_q$. The remaining two notions, $= \equiv$, are related to syntax, where $=^\alpha$ may occur as a constant symbol of type $\alpha$ in a signature $\Sigma$ and finally $=^\alpha$ as an abbreviation for the respective formula expressing Leibniz equality. Whereas $=^\alpha$ only denotes semantical equality relation $q^\alpha$ if the underlying semantical notion is strong enough (e.g., in Henkin semantics), $=^\alpha$ always denotes the respective semantical equality relation.

1.3 Higher-Order Model Existence

In classical first-order predicate logic, it is straightforward to assess the deductive power of a calculus: first-order logic has a well-established and intuitive set-theoretic semantics, relative to which completeness can be verified using, for instance, the abstract consistency method (see the introductory textbooks [And86, Fi96]). This well-understood meta-theory supported the development of different calculi well.

In higher-order logics, the situation is rather different: the intuitive set-theoretic standard semantics cannot give a sensible notion of completeness [Göd31]. However, there is a more general notion of semantics (the so-called Henkin semantics [Hen50]), that allows complete calculi and which sets the standard today for deductive power of calculi.

Peter Andrews’ Unifying Principle for Type Theory [And71] provides a method of higher-order abstract consistency that has become the standard tool for completeness proofs in higher-order logic, even though it can only be used to show completeness relative to a certain Hilbert style calculus $\Sigma$. A calculus $\mathcal{C}$ is called complete relative to a calculus $\Sigma$, iff $\mathcal{C}$ proves all theorems of $\Sigma$. Since $\Sigma$ is not necessarily complete with respect to Henkin models, the notion of completeness that can be established by this method is a strictly weaker notion than Henkin completeness.

As a consequence, the calculi developed for higher-order automated theorem proving [And71, Huc72, Huc73a, JP72, Mil83, Koh94b, Koh95] and the corresponding theorem proving systems such as TPS [ABI96], are not complete with respect to Henkin models. Moreover, they are not even sound with respect to $\Sigma$, since all of them utilise $\eta$-conversion, which is not admissible in $\Sigma$. In other words, their deductive power lies somewhere between $\Sigma$ and Henkin models.

Here the aim of this thesis is to provide a semantical meta-theory that will support the development of higher-order calculi for automated theorem proving just as the corresponding methodology does in first-order logic. To reach this goal, we establish
• classes of models that adequately characterise the deductive power of existing theorem-proving calculi (making them sound and complete), and

• a standard methodology of abstract consistency proof methods (by providing the necessary model existence theorems, which extend Andrews’ Unifying Principle), so that the completeness analysis for higher-order calculi will become as simple an exercise as in first-order logic.

Due to the inherent complexity of higher-order semantics we give an informal exposition of the issues covered and the techniques applied.

![Figure 1.1: The landscape of Higher-Order Semantics](image)

Let us now explore the particular semantic notions (see Figure 1.1). We will discuss the model classes from bottom to top, from the most specific notion of standard models \((\mathfrak{S}\Sigma)\) to the most general notion of v-complexes, motivating the respective generalisations as we go along. In Chapter 2.1, we will proceed the other way round, specialising the notion of a \(\Sigma\)-model \((\mathfrak{M}_\beta)\) more and more.

The symbols in the boxes in Figure 1.1 denote model classes, the symbols labelling the arrows indicate the properties inducing the corresponding specialisation, and the \(\nabla\)-symbols next to the boxes indicate the clauses in the definition of the corresponding abstract consistency class (cf. 3.4) that are needed to establish a model existence theorem for this class of models.

A standard model \((\mathfrak{S}\Sigma, \text{cf. Definition 2.31})\) for our higher-order logic provides a fixed set \(\mathcal{V}\) of individuals, and a set \(\mathcal{D}_0 := \{\mathbb{T}, \mathbb{F}\}\) of truth values. All the domains for the complex types are defined inductively: \(\mathcal{D}_{\alpha \rightarrow \beta}\) is the set of all functions \(\{f|f: \mathcal{D}_\alpha \rightarrow \mathcal{D}_\beta\}\). The evaluation function \(I_\varphi\) with respect to an interpretation \(I: \Sigma \rightarrow \mathcal{D}\) of constants and an assignment \(\varphi\) of variables
is obtained by the standard homomorphic construction that evaluates a $\lambda$-abstraction with a function, whose operational semantics is specified by $\beta$-reduction.

The notion of Henkin models ($\mathcal{M}_{\beta}$ or $\beta$) generalises that of standard models in the sense that instead of requiring $P_{\alpha \rightarrow \beta}$ to be the full set of functions it only requires that $P_{\alpha \rightarrow \beta}$ has enough members such that any well-formed formula can be evaluated.\(^6\) Note that with this generalised notion of a model, there are fewer formulae that are valid in all models (intuitively, for any given formulae there are more possibilities for counter-models). In fact the generalisation to Henkin models restricts the set of valid formulae sufficiently, so that all of them can be proven by a Hilbert-style calculus \cite{Hen50}. Thus, whereas standard semantics does not allow complete calculi due to Gödel's result \cite{Göd31}, Henkin semantics does \cite{Hen50, Hen69}.

It is matter of folklore that a primitive notion of equality (expressed by a primitive equality constant $\equiv \in \Sigma$) is not strictly needed, since it can be expressed by the Leibniz formula. However, the Leibniz formula only denotes the semantic equality relation if $P_{\alpha \rightarrow \sigma}$ contains enough properties to discern members of $P_{\alpha}$; in fact, we have to ensure that for all $\sigma \in P_{\alpha}$, the singleton set $\{\sigma\}$ is in $P_{\alpha}$ (see the proof of Lemma 2.36).\(^7\) In other words, we are in the somewhat paradoxical situation, that Leibniz equality (which is commonly used as a substitute for primitive equality) will only denote semantical equality, if we can guarantee that the identity relation is already present in the model (we call this property $q$, cf. Definition 2.27). Hence we introduce the corresponding semantical structures, namely Henkin models without property $q$ ($\mathcal{M}_{\beta\beta}$), in which property $q$ is not necessarily valid and thus Leibniz equality does not necessarily denote the equality relation. An example of a theorem which is valid within the class of Henkin models but is not in the class of $\mathcal{M}_{\beta\beta}$'s is given by the axiom of functional extensionality for Leibniz equality

$$\forall \alpha \forall \beta \forall X \forall \beta (\forall X_\beta \forall X, F X = G X) \Rightarrow F =^\beta G,$$

cf. Lemma 2.37.

The next generalisation of model classes derives from the fact that we want to characterise the deductive power of higher-order theorem provers at a semantic level (we will take Tps \cite{ABI+96} as an example). Note that Tps cannot be complete with respect to Henkin models and is even not generally complete for $\mathcal{M}_{\beta\beta}$'s, although there is some 'extensionality treatment' built into the proof procedure. The incompleteness of Tps for Henkin models\(^8\) can be seen from the fact that it fails to refute formulæ such as $c A_0 \wedge \neg c (\neg \neg A)$, where $c$ is a constant of type $\alpha \rightarrow \sigma$ or $c (\lambda X \alpha, B X \wedge A X) \Rightarrow c (\lambda X \alpha, A X \wedge B X)$, where $c$ is a constant of type $(\alpha \rightarrow \sigma) \rightarrow \sigma$. The problem in the former example is that the higher-order unification algorithm employed by Tps cannot determine that $A$ and $\neg \neg A$ denote identical semantic objects (by Boolean extensionality as already mentioned before), and thus returns failure instead of success. In the second example the principle of functional extensionality is needed in addition in order to prove the theorem.

The lack of completeness of refutation procedures like Tps occurs especially in a situation, where formulæ contain occurrences of propositional formulæ dominated by interpreted constants or variables or where this problem is mixed with the problem of functional extensionality: in our examples the function constant $c$ dominates the proposition $A_0$ or the set expression $\lambda X \alpha, B X \wedge A X$. To give a semantical characterisation of the deductive power of the Tps procedure, we have to generalise the class of Henkin models further, so that there are counter-models to the examples above. Obviously, this involves weakening the assumption that $P_{\beta} \equiv \{T, F\}$ (we call this assumption for Henkin models property $b$), since this entails that the values of $A$ and $\neg \neg A$ are identical: In functional $\Sigma$-models ($\mathcal{M}_{\Sigma}$, \mathcal{M}_{\beta\beta}, \mathcal{M}_{\beta\sigma}$, cf. Definitions 2.28 and 2.24) we only insist that there is a valuation $v$ of $P_{\beta}$, i.e., a function $v : P_{\beta} \rightarrow \{T, F\}$ that is coordinated with the functions $I(\neg), I(\lambda), I(\Pi')$ and (possibly) $I(=\#)$, where $I$ is the interpretation function (i.e., a family of interpretation functions) for the constants given in the signature. Thus we have a notion of validity for $\Sigma$: we call a proposition $A$ valid in $\mathcal{M} \equiv \langle D, I, v \rangle$ under an assignment $\varphi$, iff $v(I_\varphi(A)) \equiv T$.

\footnote{In other words: the functional universes are rich enough to satisfy the comprehension axioms.}

\footnote{On a similar note, Peter Andrews remarked in \cite{And72} that if the set $P_{\alpha \rightarrow \sigma}$ is so sparse that the semantic identity relation is not present, it is then possible to construct a Henkin model where Leibniz equality is non-extensional.}

\footnote{In case the extensionality axioms are not available in the search space. Note that one can add extensionality axioms to the calculus in order to achieve, at least in theory, Henkin completeness. But this increases the search space drastically and is not feasible in practice.}
In our first example, there is a \( \Sigma \)-model structure \( \mathcal{M} \equiv (\mathcal{D}, \mathcal{I}, v) \), where \( \mathcal{I}_p(A) \neq \mathcal{I}_p(\neg A) \), and therefore \( \mathcal{I}_p(c(A)) \neq \mathcal{I}_p(c(\neg A)) \) if we take \( \mathcal{I}(c) \) to be the identity function on \( \mathcal{D}_0 \). In particular, we can have \( \mathcal{I}(\mathcal{I}_p(c(A))) \neq \mathcal{I}(\mathcal{I}_p(c(\neg A))) \), and therefore \( \mathcal{I}(\mathcal{I}_p(cA \land \neg c(\neg A))) \equiv \mathcal{F} \), since \( v \) is a valuation.

Clearly, for functional \( \Sigma \)-models we have the same choices concerning the role of equality, therefore, we distinguish the classes \( \mathcal{M}_{\mathcal{M}_3} \) and \( \mathcal{M}_{\mathcal{M}_4} \) of functional \( \Sigma \)-models without/with property \( q \). Furthermore, we have the class \( \mathcal{M}_{\mathcal{M}_3} \) of functional \( \Sigma \)-models with (only) property \( b \), which expresses that the set of truth values contains exactly two elements, i.e., \( \mathcal{D}_0 \equiv \{ \mathcal{T}, \mathcal{F} \} \). Since functional \( \Sigma \)-models with properties \( b \) and \( q \) are defined to be \( \Sigma \)-Henkin models, we can also view \( \mathcal{M}_{\mathcal{M}_3} \) as "Henkin models without property \( q \)."

Finally, we drop the requirement of functional extensionality for \( \Sigma \)-models, which is encoded as property \( f \) (cf. Definition 2.24). This is the most general notion of semantics we will discuss in this thesis; we only insist that the evaluation function is a homomorphism which respects instantiation. In such models, a function is not uniquely determined by its behaviour on all possible arguments. For the construction of such models we therefore need labelings for functions (e.g., a green and a red version of a function \( f \)) in order to differentiate between them, even though they are functionally equivalent. As already done for functional \( \Sigma \)-models we analyze properties \( q \) and \( b \) for non-functional \( \Sigma \)-Models. Whereas \( b \) may or may not hold for non-functional \( \Sigma \)-Models, it turns out that property \( q \), i.e., the requirement that the intended equality relations are provided in the respective domains, only makes sense in connection with functionality. More precisely, property \( q \) is not independent from functionality.

Peter Andrews has pioneered the construction of non-functional models with his \( v \)-complexes in [And71]. These are even more general constructions than our \( \Sigma \)-models, since totality of the evaluation function is not assumed. His construction is based on Schitte’s semi-valuation method [Sch60], which only needs partial valuations to construct a model for a given Hintikka set.

In this thesis, we concentrate on the other aspects of higher-order models and ensure totality of our evaluation functions by a saturation condition (cf. 3.9) in our abstract consistency classes. This does not restrict the applicability of our model existence theorems, since saturation is relatively simple to prove for a given calculus (see [Koh94b, Koh98, BK97b]). For all the notions of a model we present model existence theorems lying the differentiating conditions of the models to suitable conditions in the abstract consistency classes (see Chapter 3). We can use the classical construction in all cases: abstract consistent sets are extended to Hintikka sets 3.13, which induce a valuation on a term structure (see Definition 2.14). In some cases, we have to take a quotient structure (see Definition 2.12) to ensure that the set of truth values is exactly \( \{ \mathcal{T}, \mathcal{F} \} \) for property \( b \).

The simplest way to ensure property \( q \) is by assuming that the signature contains a primitive connective for equality, which is evaluated as semantical identity (we call this property \( c \)). We will study the case in Section 3.3. On the one hand, the semantical situation becomes much simpler now (see Figure 2.1), since \( \mathcal{M}_{\mathcal{M}_3}, \mathcal{M}_{\mathcal{M}_4} \) and \( \mathcal{M}_{\mathcal{M}_3} \) are identified, just as \( \mathcal{M}_{\mathcal{M}_3}, \mathcal{M}_{\mathcal{M}_3} \) and \( \mathcal{H} \) on the other hand, the existence of another logical constant induces further conditions in the definition of the abstract consistency classes.

### 1.4 Traditional and Extensional Higher-Order Resolution

As we have seen, traditional higher-order resolution approaches [And71], [JP72], [Hue72] as well as the TPS-system generally need to add all extensionality axioms to the search space in order to ensure Henkin completeness. This holds for more recent approaches like [Koh94b] and [Wol93] as well. Although the functional extensionality principles taken alone is integrated into syntactic higher-order unification both extensionality principles still have to be additionally axiomatised in the search space in order to reach Henkin completeness. This can be seen for problems like \( (\lambda X . \text{red} X \land (\text{circle} X \land (\text{large} X \lor \neg (\text{large} X)))) = (\lambda Y . \text{circle} X \land \text{red} X) \). It does not help at all that functional extensionality is already integrated in higher-order unification as we here
1.4. TRADITIONAL AND EXTENSIONAL HIGHER-ORDER RESOLUTION

obviously need an appropriate combination of both principles in order to equalise the terms. And in the above approaches — which all follow the idea of generally delaying higher-order unification in order to overcome the undecidability problem — such a combination can only be achieved by adding all extensionality axioms to the search space.

Wolfram [Wol93] applies higher-order E-unification instead of pure syntactical higher-order unification, but he does not provide an explicit account of theories $E$ containing both extensionality principles: they require potential recursive calls to an (again) Henkin complete higher-order theorem prover. And also the higher-order E-unification algorithm of Snyder [Sny90] provides no solution to the problem with the extensionality principles, as the suggested approach is restricted to first-order theories and, e.g., does not take the Boolean extensionality principle into account.

The idea of such recursive calls was first mentioned in [Koh95]. But unfortunately this approach still lacks Henkin completeness as the single rule added to the calculus (which is analogous to the rule $Equiv$ in our approach; see Figure 4.2) is not strong enough to realize all the necessary aspects of a suitable interaction of Boolean and functional extensionality.

Another unfortunate aspect about [Koh94b] and [Koh95] is that both approaches not only lack a general extensionality treatment but also lack soundness. E.g., in both approaches it is possible to prove that each function has a fixed point, which can be formulated as follows: $\forall F_\alpha \exists \lambda_\alpha F_\alpha \lambda_\alpha = \lambda_\alpha$. This is caused by the extra logical treatment of Skolemisation with the so called variable conditions employed in both approaches. But the conditions added with each eliminated existential quantifier are not strong enough to prevent a proof of the above statement. As the author was not able to suitably fix this problem we employ in this thesis traditional Skolemisation again and avoid the usage of variable conditions.\footnote{Michael Kohlhase remarked that he is currently working on a solution to the problem. If this solution turns out to be sound, it can an probably should be employed instead of traditional Skolemisation within the calculi presented in this thesis as well.}

More precisely, we employ Miller’s sound adaptation of traditional first-order Skolemisation [Mil83], which associates with each Skolem constant the minimum number of arguments the constant has to be applied to. Higher-order Skolemisation becomes sound\footnote{Without this additional restriction the calculus does not really become unsound, but one can prove an instance of the axiom of choice, which we want to be optional in our approach.}, if any Skolem function $f^n$ only occurs in a Skolem term, i.e., a formula $S \equiv f^n A$, where none of the $A$ contains a bound variable. Thus the Skolem terms only serve as descriptions of the existential witnesses and never appear as functions proper.

As already mentioned, a main motivation for this thesis is to get rid of the drawback of the above approaches, which have to add the (generally infinitely many) extensionality axioms to the search space in order to reach Henkin completeness. In fact, none of the currently available systems for classical type theory actually adds the extensionality to the search space but they instead accept incompleteness. The TPS-system [AB1+96] probably offers the most practicable solution to the situation: It avoids the extensionality axioms and instead analyses the input problems in order to try to modify them in an appropriate way (e.g., by applying the functional extensionality principles to input equations of functional type or by replacing equations on type $\theta$ by equivalences). But Example $E_{ax}$ in Section 8.1 demonstrates that there are many examples which can not be appropriately modified this way before being passed to the refutation process. Also the Ho1-system [GM93] and the ISABELLE-system, which are other prominent theorem provers for higher-order logic, do not automatically add all extensionality axioms to the search space when proving subgoals automatically.

In order to back up the motivation for this thesis we will briefly sketch the drawbacks of the option to add the extensionality axioms to the search space. This will also clarify why the available systems indeed avoid this option. First, we remember the definition of the functional and Boolean extensionality principles (for Leibniz equality):

$$\begin{align*}
\text{EXT}_{L}^{\alpha \rightarrow \beta} := & \forall F_\alpha \rightarrow \beta, \forall G_\alpha \rightarrow \beta, (\forall X_\beta, F_\alpha X \equiv G_\alpha X) \Rightarrow F_\alpha \equiv^\beta G_\alpha \\
\text{EXT}_{L}^{\alpha} := & \forall A_\alpha \forall B_\alpha (A \equiv^\alpha B) \equiv (A \equiv B)
\end{align*}$$

One challenging problem for the mechanisation of these principles, namely the general need for infinitely many axioms $\text{EXT}_{L}^{\alpha \rightarrow \beta}$, has already been mentioned. But even apart from this problem
these axioms would cause an enormous explosion of the search space. This becomes immediately clear if one considers the clauses obtained by the normalisation of just one single extensionality axiom. Note that Leibniz Equality \( \equiv \) is just an abbreviation for a higher-order term and that the expanded axioms are thus of form:

\[
\begin{align*}
\text{EXT}_L^{\approx} & \coloneqq \\
\forall F_{\alpha \rightarrow \beta} \forall G_{\alpha \rightarrow \beta} (\forall X_{\beta} (\forall P_{\beta \rightarrow \alpha} P (F X) \Rightarrow P (G X))) \Rightarrow (\forall Q_{(\alpha \rightarrow \beta) \rightarrow \alpha} Q F \Rightarrow^\beta Q G)
\end{align*}
\]

Note that normalising these formulas introduces many flexible variables into the search space.\(^{11}\) And the problem with the free predicate variables at head position in literals — which are also called flexible literal heads — is that the primitive substitution rule Prim becomes applicable to them (cf. [And71] or rule Prim in Figure 4.2 in Chapter 4). This rule, which is very important in higher-order theorem proving for reaching completeness, blindly instantiates each flexible literal head with a most general binding (partial binding) that imitates a logical connective, i.e., with a most general formula that introduces a logical connective at head position. Note that there are infinitely many universal quantifiers (one for each type) and thus the primitive substitution rule is infinitely branching. And as this principle aims at introducing most general terms apart from the new logical connective at head position, new free predicate variables are generated which subsequently become — after the necessary normalisation of the modified clauses — new flexible literal heads. And not enough, the primitive substitution rule can thereby even duplicate flexible literals. It is thus obvious that each single clause derived from an extensionality axiom with a flexible literal leads to explosion of the search space that is awkward to handle in practice. Thus, in principle a higher-order theorem prover can spend an arbitrary amount of time just in applying primitive substitution to the clauses belonging to the extensionality principles. And unfortunately the primitive substitution principle cannot be generally avoided as otherwise a higher-order resolution approach even fails to prove such trivial theorems like \( \exists X_{\beta} X_\beta \) or \( \exists P_{\alpha \rightarrow \alpha} \forall Y_{\beta} P Y \).

The illustrated serious drawbacks of the option to add all extensionality axioms to the search space was the main motivation for the development of the extensional higher-order resolution calculus \( \mathcal{RE} \) presented in Chapter 4, which instead of adding these axioms employs the idea of recursive calls to the overall refutation process from within syntactical higher-order pre-unification as first mentioned in [Koh95]. But in contrast to [Koh95] the approach presented here (which is also described in [BK98a]) realises the necessary interaction of both extensionality principles in a suitable way, and this finally makes it the first Henkin complete refutation approach for classical type theory. Note that this approach can also be viewed as a test calculus for general higher-order \( E \)-unification in the following sense: If we pass the conditional equations (\( \forall X_{\alpha} (L_1 = R_1) \wedge \ldots \wedge (L_n = R_n) \)) to our extensional higher-order resolution calculus, then this calculus tests for the \( E \)-unifiability of the term (\( L = R \)), where \( E \) is the theory defined by the equations (\( \forall X_{\alpha} (L_1 = R_1) \wedge \ldots \wedge (L_n = R_n) \)) and the extensionality principles.

### 1.5 Adding Primitive Equality

Whereas the research in the field of higher-order term rewriting is very active (see for instance [Pre98, NP98, NM98b, Nip95, Pre95, Pre94, vO94, Wd93] for higher-order term rewriting and narrowing or [JR99, JR98, LP95] for recent work on higher-order term orderings) the integration of primitive equality and the application of term rewriting techniques in a refutation based higher-order theorem proving context is still rather unexamined. At a first glance one might assume that as soon as suitable higher-order rewriting techniques are available — the currently most

---

\(^{11}\) In Chapter 2.8 we will illustrate that there are infinitely many more or less natural ways of defining equality in classical type theory apart from Leibniz equality. Thus it may be possible to find some slightly more appropriate formulations of the extensionality principles for defined equality as the ones presented here, even though this seems to be rather unlikely. It will certainly not be possible, though, to avoid all the free predicate variables introduced here.
1.5. ADJUSTED PRIMITIVE EQUALITY

Challenging problem is to find suitable term-orderings in higher-order logic — they can be successfully employed in an automated higher-order theorem proving context as well. And furthermore one may argue that because automated first-order provers that are heavily based on term rewriting techniques (such as WALDMEISTER [HBVL97], BILKBEM or SPASS [Wei97]) seem to have taken the lead in many application domains over systems that employ difference reducing techniques, an analogous situation will arise in higher-order logic as soon as the available higher-order term rewriting approaches become strong enough. Whereas this may indeed happen in specific domains, this thesis provides a counterargument for the overall success of term rewriting techniques in automated higher-order theorem proving by pointing out new serious problems that in addition to the term-ordering problem need to be solved:

- As already motivated higher-order unification takes only the functional but not the Boolean extensionality principle into account.

- In Chapter 2.8 we illustrate that in classical type theory infinitely many different terms (apart from the Leibniz definition) define equality and that we cannot decide whether a (probably automatically generated) proof problem contains a defined equation at some sub-term position. Thus, even if we consider a higher-order logic with primitive equality and try to employ term rewriting techniques, we cannot generally restrict equality handling in a calculus that aims at Henkin completeness only to primitive equality as our input problem may still contain some defined equations we cannot even detect. And Examples $E_2^{eq}$ and $E_3^{eq}$ in Section 8.1 illustrate that there are even theorems which neither contain a defined nor a primitive equation but where the extensionality principles are nevertheless of central importance for the proof. Furthermore, in classical type theory with a primitive notion of equality it is not possible to take care of both kinds of extensionality principles, those for Leibniz equality as well as those for primitive equality. And a Henkin complete approach therefore also needs to realize a general interleaving between both concepts of equality. This sharply contrasts with the situation in first-order theorem proving where one can choose between defining equality (e.g., by axiomatising it) or considering a primitive notion of equality and providing new calculus rules. In classical type theory one simply does not have this choice: Defined equality is always built-in.

- In Chapter 5 we adapt traditional first-order paramodulation [RW69] to higher-order logic and define a higher-order paramodulation approach $\mathcal{P}$ based on the extensional higher-order resolution calculus $\mathcal{R}$. Whereas the important reflexivity rule (or axiom) known from first-order paramodulation is naturally built-in in our approach, we will show that in order to ensure Henkin completeness additional extensionality rules (or axioms) for primitive equality are needed. The problem is that in classical type theory even single positive equations can be contradictory, which again contrasts with the situation in first-order logic. Unfortunately the needed additional extensionality rules further strengthen (as would the respective axioms) the already strong difference-reducing character of the underlying calculus $\mathcal{R}$. It will be motivated by examples in Chapter 8 that a proper term rewriting approach will be quite hard to achieve as the realisation of a suitable interaction of the functional and Boolean extensionality principles has a rather natural difference-reducing character.

As the development of suitable heuristics for the higher-order paramodulation approach $\mathcal{P}$ with its intrinsic mixed term rewriting and difference reducing character seems to be quite difficult (if possible at all) we additionally develop in Chapter 6 the difference-reducing approach $\mathcal{E}_{\text{RED}}$ which adapts the ideas of first-order RUE-resolution [Dig79] to our higher-order setting. This approach faces the same problem about primitive equality as the paramodulation approach $\mathcal{P}$ and has to add new extensionality for primitive equality in order to reach Henkin completeness. The only difference between $\mathcal{R}$ and $\mathcal{E}_{\text{RED}}$ is actually that the latter avoids the paramodulation rule and instead allows to resolve and factorise on unification constraints. Thereby $\mathcal{E}_{\text{RED}}$ gains a pure difference-reducing character that may be easier to guide and handle in practical applications than the intrinsic mixed term rewriting/difference-reducing character of $\mathcal{P}$. This aspect is illustrated by examples $E_1^{eq}$ and $E_2^{eq}$ in Section 8.6.
1.6 The Leo System

Leo realises the calculus $\mathcal{ER}$ and is based on an extended set of support architecture, that adapts this well known technique from first-order theorem proving (e.g., see [McC94]) with respect to the very specific requirements of extensional higher-order theorem proving. The system is implemented in Common Lisp [Ste90] and employs many data structures and basic algorithms offered by the KEIM-toolbox [HKK+94] for deduction systems. This toolbox, e.g., provides the higher-order term indexing module described in [Kle97] that adapts the first-order term indexing techniques of [Gra95] to the higher-order setting.

The prototypical prover Leo has been implemented mainly during a 5 months stay at Carnegie Mellon University, Pittsburgh, USA, and is discussed in Chapter 7 in detail (see also [Ben97]).

Originally two contrary search strategies have been developed. In this thesis we only discuss the one that has become the standard strategy in Leo. This strategy employs like most first-order approaches unification and subsumption as filter in order to sort out clauses which are either superfluous or which cannot contribute to the refutation. The main difference to first-order theorem provers is that these filtering side-computations are computationally much more expensive in the higher-order setting and generally even undecidable. Thus, Leo artificially restricts and interrupts these side-computations with the result that these filters become quite imperfect. Especially the employed subsumption check is based only on the simplification part of higher-order unification and we do not develop and realise a notion of extensional higher-order subsumption, which would lead to a much stronger and more appropriate (but undecidable) filter for our purposes.

Anyhow the extensionality principles cause new practical problems for Leo. E.g., term indexing techniques cannot be employed to the same extent and for the same purposes as in first-order automated theorem proving. This is illustrated by the following example, which shows that the usage of term indexing techniques within the computation of resolution partners causes incompleteness: Assume we want to prove

$$\forall p_{(\alpha \rightarrow \alpha)} \in \{X, \text{red } X \land \text{circle } X\} \in P \Rightarrow \{X, \text{circle } X \land \text{red } X\} \in P$$

while coding sets as characteristic functions and defining $\in := \lambda x \alpha, \lambda M_{\alpha \rightarrow \alpha} M X$. Definition expansion and clause normalisation leads to the clauses

$$C_1 : [p_{(\alpha \rightarrow \alpha)} (\lambda X_{\alpha \rightarrow \alpha} \text{red}_{\alpha \rightarrow \alpha} X \land \text{circle}_{\alpha \rightarrow \alpha} X)]^T \quad C_2 : [p_{(\alpha \rightarrow \alpha)} (\lambda X_{\alpha \rightarrow \alpha} \text{circle}_{\alpha \rightarrow \alpha} X \land \text{red}_{\alpha \rightarrow \alpha} X)]^F$$

where $p_{(\alpha \rightarrow \alpha)}$ is a new Skolem constant. Leo can easily solve this problem by first resolving between $C_1$ and $C_2$, decomposing $p_{(\alpha \rightarrow \alpha)}$ in the resulting unification constraint and employing the functional extensionality principle, thereby deriving the constraint

$$[\text{red}_{\alpha \rightarrow \alpha} s \land \text{circle}_{\alpha \rightarrow \alpha} s = \text{circle}_{\alpha \rightarrow \alpha} s \land \text{red}_{\alpha \rightarrow \alpha} s]^F$$

Now Leo employs the Boolean extensionality principle, i.e., replaces $=$ by $\equiv$, and performs a recursive call to the overall refutation search, thereby deriving the empty clause in a quite straightforward way. To come to the point, the sketched goal-directed refutation is not possible when employing syntactical term indexing techniques within the computation of resolution partners as the first, essential resolution step between $C_1$ and $C_2$ would not even be suggested. The problem is that syntactical term indexing lacks the extensionality principles, and for the same reason Leo can generally not employ term indexing techniques in unification.

We want to point out that the prototypical implementation of Leo is not a complete refutation procedure and in Chapter 7 we will discuss the different sources of incompleteness in detail. But we will also sketch possible solutions to the incompleteness problems.

A case study, which is illustrated in chapter 7, has demonstrated that Leo is indeed capable of solving simple theorems about sets, which require the application of the extensionality principles. In this experiment Leo could solve 95 of 97 theorems from the article Boolean Properties of Sets [TS89] of the Mizar library [Rud92]. This experiment also showed that on this domain Leo outperforms well known first-order theorem provers, which cannot exploit the expressiveness of
classical type theory and encode these examples in the case study in Tarski Grothendieck set theory [Try89] (see the results of this case study at http://www-irm.mathematik.hu-berlin.de/~ilf/miz2atp/mizstat.html. At present the case study with LEO is continued with the slightly more challenging examples from the Mizar-article Some Basic Properties of Sets [By89], and without any modifications LEO can already solve about 40% of them.

In Chapter 7 we also sketch further aspects of LEO like its features as an interactive theorem prover or its integration to ΩMEGA [BCF+97].
Chapter 2

Syntax and Semantics of Higher-Order Logic

2.1 Syntax and Preliminaries

In this section we introduce the preliminaries in a quite compact form.\(^1\) Apart from the notational conventions most of the introduced concept are as defined as usual.

We start with a higher-order logic based on Church’s simply typed lambda calculus [Chu40] and choose the set of base types \( \mathcal{BF} \) to consist of the types \( \iota \) and \( \alpha \), where \( \iota \) denotes the set of truth values and \( \iota \) the set of individuals. The set of all types \( \mathcal{T} \) is inductively defined over \( \mathcal{BF} \) and the type constructor \( \rightarrow \).

We define the order of types and \( \lambda \)-terms as in [SG89, Sny91].

We assume that our signature \( \Sigma \) contains a countably infinite set of variables \( \mathcal{Y} \) and constants \( \mathcal{C} \) for every type \( \tau \in \mathcal{T} \). Additionally we postulate the existence of the logical connectives \( \neg_{\alpha \rightarrow \alpha} \), \( \lor_{\alpha \rightarrow \alpha} \), \( \Pi_{\alpha \rightarrow \alpha} \rightarrow_{\alpha} \) (in short \( \Pi^\alpha \)) for every type \( \alpha \in \mathcal{T} \). A signature that contains additionally the logical connectives \( \equiv_{\alpha \rightarrow \alpha} \rightarrow_{\alpha} \) (in short \( \equiv^\alpha \)) for all types \( \alpha \in \mathcal{T} \) is noted by \( \Sigma^\alpha \). All the logical connectives in \( \Sigma \) or \( \Sigma^\alpha \) denote their intuitive semantical counterparts.

The remaining logical connectives are defined as abbreviations of the given ones: \( \mathcal{A} \land \mathcal{B} := \neg(\neg \mathcal{A} \lor \neg \mathcal{B}) \), \( \mathcal{A} \Rightarrow \mathcal{B} := \neg \mathcal{A} \lor \mathcal{B} \), \( \mathcal{A} \equiv \mathcal{B} := (\mathcal{A} \Rightarrow \mathcal{B}) \land (\mathcal{B} \Rightarrow \mathcal{A}) \). \( \forall \mathcal{X} \), \( \exists \mathcal{X} \), \( \mathcal{A} := \Pi_{\alpha} \rightarrow_{\alpha} \) \( (\lambda \mathcal{X} \alpha \mathcal{A}) \). \( \exists \mathcal{X} \), \( \mathcal{A} := \forall \mathcal{X} \), \( \neg \mathcal{A} \). All other constants are called parameters, since the argumentation in this paper is parametric in their choice.\(^2\)

Unlike stated otherwise, variables are printed as upper-case (e.g., \( \mathcal{X} \)), constants as lower-case letters (e.g., \( \alpha \)), and arbitrary terms appear as bold capital letters (e.g., \( \mathcal{T} \)). If the type of a symbol or term is either not of importance or uniquely determined by the given context, we do not explicitly mention it.

We denote the set of all terms over a signature \( \Sigma \) \( \equiv_{\alpha} \) by \( \text{wff}(\Sigma) \) (resp. \( \text{wff}^\alpha(\Sigma^\alpha) \)), and \( \text{wff}_\alpha(\Sigma) \) (resp. \( \text{wff}_\alpha^\alpha(\Sigma^\alpha) \)) denotes the set of all \( \Sigma \)-terms of type \( \alpha \). Terms of type \( \alpha \) are also called propositions or formulae, and closed propositions are also called sentences. The set of propositions is abbreviated as \( \text{prop}(\Sigma) \) and the set of all sentences as \( \text{cprop}(\Sigma) \). A is called atomic, if its \( \beta \eta \)-normal form (cf. below) does not have a logical connective at head position.

We will take the order of a formula to be the highest order of the type of any of its subterms, and the order of a set of formulae to be the maximum of the orders of its members.

In order to avoid confusion we clarify the meaning of the different equality symbols used in this paper. \( \equiv^\alpha \in \Sigma \) is the syntactic equality constant in our higher-order language. We will illustrate below that equality can also be defined in higher-order logic and we refer to this definition by

\(^1\)We apologise that all the important definitions are introduced here at once, instead of introducing them when needed. The motivation is to provide a compact reference.

\(^2\)In particular, we do not assume the existence of description or choice operators. For a detailed discussion of the semantic issues raised by the presence of these logical constants see [And72b].
\[\equiv.\] The intuitive semantical equality relations in \(\mathcal{D}_{\alpha \rightarrow \gamma \rightarrow \delta}\) are denoted by \(q^\alpha\). For the meta-level argumentation we use \(\equiv\) and \(\vdash\) for definitions.

To ease readability we assume right-associativity for the type constructor \(\rightarrow\) and left-associativity of function application: \(A_{\alpha \rightarrow \beta \rightarrow \gamma}B_\alpha C_\beta := ((A_{\alpha \rightarrow \beta \rightarrow \gamma}B_\alpha) C_\beta)\). Furthermore, we sometimes abbreviate function applications by \(b_{\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta} U^1_{\alpha_n} \cdots U^n_{\alpha_n}\), which stands for \((\cdots (b_{\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \beta} U^1_{\alpha_n}) \cdots U^n_{\alpha_n})\). A dot \(\cdots\) occurring in a \(\lambda\)-term stands for a left bracket whose mate is as far to the right as consistent with all other brackets and the construction of the term. We avoid brackets in every case where the construction of an expression is uniquely determined by the context.

The structural equality relation in our higher-order logic is induced by \(\beta\eta\)-reduction
\[
(\lambda X. A) B \rightarrow_\beta [B/X]A \quad (\lambda X. C X) \rightarrow_\eta C
\]
where \(X\) is not free in \(C\). It is well-known (c.f. [Bar84]), that the reduction relations \(\beta\), \(\eta\), and \(\beta\eta\) are terminating and confluent, so that there are unique normal forms for each term \(T\) denoted by \(T_{\beta}\), \(T_{\eta}\), \(T_{\beta\eta}\). The induced structural equality relations are denoted by \(\equiv_{\beta}\), \(\equiv_{\eta}\) (and \(\equiv_{\beta\eta}\) for the equality relation induced by the renaming of bound variables). Another important normal form used in this paper is the **head-normal form** (which is unique only with respect to \(\beta\eta\)-equality): a term \(\lambda X^a h U^m\) is in head-normal form, iff \(h\) is a variable or a constant. The head-normal form of a term \(T\) is denoted by \(T_{\text{hn}}\).

The definitions of **free and bound variables, substitutions and the application of substitution** are as usual (see [Bar84]). In this paper we denote the set of free variables of a term \(T\) (analogously for literals and clauses) by \(\text{free}(T)\). Whereas the usual application of a substitution \([A/X]T\) to \(T\) is denoted by \([A/T]\), we denote with \([T_{\text{hn}}]\) the combination of usual substitution with subsequent reduction of the resulting term (literal or clause) to head normal form.

We define **satisfiability, unsatisfiability, and validity** of a formula \(F\) or set of formulae \(\Phi\) with respect to a mode \(P\) \(\mathcal{M}\) as usual.

When we speak of a **Skolem term** \(s_{\alpha}\) for a clause \(C\) and \(\text{free}(C) \equiv \{X^1_{\alpha}, \ldots, X^n_{\alpha}\}\), then \(s_{\alpha}\) is an abbreviation for the term \((f^\alpha_{\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \alpha} X^1_{\alpha} \cdots X^n_{\alpha})\), where \(f\) is a new constant from \(C_{\alpha \rightarrow \cdots \rightarrow \alpha^n \rightarrow \alpha}\) and \(n\) specifies the number of necessary arguments for \(f\). The latter is important as a naive treatment of Skolemisation results in a calculus that is not sound with respect to Henkin models, since Skolem functions are special choice functions\(^4\), which are not guaranteed to exist in Henkin models. A solution due to [Mil83] is to associate with each Skolem constant the minimum number of arguments the constant has to be applied to. Skolemisation becomes sound, if any Skolem function \(f^\star\) only occurs in a **Skolem term**, i.e., a formula \(S \equiv f^\star A^\gamma\), where none of the \(A^\gamma\) contains a variable that is bound outside of \(S\). Thus the Skolem terms only serve as descriptions of the existential witnesses and never appear as choice functions.

Let \(\alpha := (\beta^\gamma \rightarrow \gamma)\) and let \(h\) be a constant or variable of type \(\langle \beta_m \rightarrow \gamma \rangle\) in \(\Sigma\), then \(G := \lambda X^a h U^m\) is called a **partial binding** of type \(\alpha\) and head \(h\) (see also [SG89, Sny91]), if \(V^h \equiv \{X^h_{\beta} \mid X^h_{\beta} \in \beta^h\}\) and the \(h^i\) are new variables of types \(\beta^i \rightarrow \delta^i\). It is easy to see that general bindings indeed have the type and head claimed in the name and are most general in the class of all such terms.

Partial bindings, where the head is a bound variable \(X^h_{\beta}\), are called **projection bindings** (we write them as \(G^h_{\beta}\)) and **imitation bindings** (written \(G^j_{\alpha}\)) otherwise. Since we need both imitation and projection bindings for higher-order unification, we collect them in the set of **approximating bindings for \(h\) and \(\alpha\)** \(AB^h_{\beta} := \{G^h_{\beta}\} \cup \{G^j_{\alpha} \mid j \leq l\}\).

For a general introduction to higher-order unification we refer to [SG89, Sny91].

The calculi in this paper are defined on **clauses**, which are disjunctions of **literals** (e.g., \([\varphi_0 \rightarrow X_{\alpha}]^T \lor [\varphi_1 \rightarrow X_{\alpha}]^T \lor \{\varphi_2 \equiv X_{\alpha}\}^F\)). For literals we differentiate between **pre-literals** and **proper literals**. A **pre-literal** consists of a proposition \(N_{\alpha}\) in head-normal form (atom) and a **polarity** \(T\) or \(F\) which states whether this literal is positive or negative. We call a literal **proper**, iff it contains no logical constant beside \(=\) at head position.

---

\(^3\)The different notions of models, e.g. Henkin models and standard models will be introduced in Sections 2.2–2.7.

\(^4\)They choose an existential witness from the set of possible witnesses for an existential formula.
We further differentiate between positive literals, negative literals, and unification constraints. Unification constraints refer to negative literals with primitive equations as atoms. For the calculi discussed in this paper unification constraints are handled in one of the following ways:

- As special negative literals with a special head symbol $\rightarrow^\alpha \in \Sigma$, i.e., $\rightarrow^\alpha$ is not a logical constant. By special literal we mean that this literal is treated as a unification constraint only, such that no rule but the unification rules are allowed to operate on them. This will be the case in the extensional higher-order resolution calculus $\mathcal{ER}$ discussed in Chapter 4. 
  
  E.g., consider the clauses $C_1 : [H \rightarrow^\alpha a]T$ and $C_2 : [X = A]^T$. Clause $C_2$ consists only of a unification constraint, which is encoded as a negative primitive equation. This literal is not treated as usual negative literal in $\mathcal{ER}$, such that resolving between $C_1$ and $C_2$ is forbidden and positive primitive literals like $[f = g]^T$ are not allowed.

- As special negative literals with a logical constant $\rightarrow^\alpha \in \Sigma$ as head symbol. Special negative literals means that despite the fact that $\rightarrow^\alpha$ is a logical constant provided by the signature, no rule other than the unification rules are allowed to operate on them. This will be the case in the extensional higher-order paramodulation calculus $\mathcal{EP}$ in Chapter 5. Now positive primitive equations like $[f = g]^T$ are allowed (since $\rightarrow^\alpha \in \Sigma$), but resolution (e.g., between $C_1$ and $C_2$ above), factorisation, etc. on unification constraints is still forbidden.

- As ordinary negative literals with the logical constant $\rightarrow^\alpha \in \Sigma$ as head symbol. In this case all rules, e.g., the resolution and factorisation rules, are allowed to operate on these literals. This holds for the extensional higher-order paramodulation calculus $\mathcal{EP}$ in Chapter 6. Now resolution between clauses $C_1$ and $C_2$ from above is allowed.

A clause $C$ is called a proper clause, iff it is in clause normal form, i.e., if all literals of $C$ are proper. Otherwise we call $C$ a pre-clause. Similarly a clause $C$ is in head-normal form, iff all its literals are.

An unification constraint $U := [X_a = X_b]^P$ or $U := [N_a = X_b]^P$ is called solved, iff $X_a \notin \text{free}(N_a)$. In this case $X$ is called the solved variable of $U$. Furthermore, a unification constraint in head-normal form $[[H \mathbf{Un}] = (G \mathbf{Un}m)]^P$ for $n, m \geq 1$ is called a flex-flex constraint (flex-rigid constraint), iff $H$ and $G$ are variables (either $H$ or $G$ is a variable).

We define a clause $C$ to be empty (denoted by $\Box$), iff $C$ consists only of flex-constraints. As it is well known that any set of flex-flex constraints is unsatisfiable, such that they evaluate to $F$, we know that $\Box$ is unsatisfiable (cf. [Hue75]).

Let $C := U^1 \lor \cdots \lor U^m$ be a clause with unification constraints $U^{1}, \ldots, U^{m}$ ($1 \leq m$). Then a disjunction $U^{i_1} \lor \cdots \lor U^{i_{k}}$ ($j \in \{1, \ldots, m\}$) of solved unification constraints occurring in $C$ is called solved for $C$, iff for every $U^{i_1} \lor \cdots \lor U^{i_{k}}$ ($1 \leq j \leq k$) holds: the solved variable of $U^{i_1}$ does not occur free in any of the $U^{i_l}$ for $l \neq j, 1 \leq l \leq k$.

Given a calculus $R$, i.e., a set of rules $r_n (n \geq 0)$ defined on clauses, we define the following derivation relation: $\Phi \vdash^rs C (\mathcal{C} \vdash^rs C)$, iff $C$ is the result of one application of rule $r_n \in R$ to premise clauses $C' \in \Phi$ (to premise clause $C'$). Multiple step derivations within a calculus $R$, e.g., $\Phi_1 \vdash^r \ldots \vdash^r \Phi_k$ (or $C_1 \vdash^r \ldots \vdash^r C_k$) where $k \geq 0$ and $r_{i_j} \in R$ for $1 \leq k \leq n$, are abbreviated by $\Phi_1 \vdash^R \Phi_k$ (or $C_1 \vdash^R C_k$). Derivations in a calculus $R$ of exactly $n$ steps are symbolised by $\vdash^*_R$.

A rule $r$ is called admissible in one of our resolution calculi $R$, iff adding rule $r$ to $R$ does not increase the set of refutable formulae. Furthermore, a rule $r$ is called derivable in $R$, iff each application of rule $r$ can be replaced by an alternative derivation in $R$.

We shall now introduce a variety of semantical constructions for classical type theory motivated by the different of roles equality and extensionality. We will start out by defining $\Sigma$-structures (and as an intermediate step pre-$\Sigma$-structures) as algebraic semantics for the simply typed $\lambda$-calculus and then specialise them to our notions of models by requiring a special treatment of the propositional formulae.

\footnote{Note that the concepts admissible and derivable as introduced here differ from the standard meaning.}
2.2 Pre-\(\Sigma\)-Structures

**Definition 2.1 (Pre-\(\Sigma\)-Structure).** A collection \(\mathcal{D} := \mathcal{D}_\alpha := \{\mathcal{D}_\alpha \mid \alpha \in \mathcal{T}\}\) of sets \(\mathcal{D}_\alpha\), indexed by the set \(\mathcal{T}\) of types, is called a typed collection (of sets). Let \(\mathcal{D}_\alpha\) and \(\mathcal{E}_\alpha\) be typed collections, then a collection \(\mathcal{I} := \{I^\alpha : \mathcal{D}_\alpha \rightarrow \mathcal{E}_\alpha \mid \alpha \in \mathcal{T}\}\) of mappings is called a typed mapping \(\mathcal{I} : \mathcal{D} \rightarrow \mathcal{E}\). We call the triple \(\mathcal{A} := (\mathcal{D}, \@, \mathcal{I})\) a pre-\(\Sigma\)-structure, if \(\mathcal{D} \equiv \mathcal{D}_\alpha\) is a typed collection of sets and
\[\@ := \{@^{\alpha\beta} : \mathcal{D}_{\alpha \rightarrow \beta} \rightarrow \mathcal{D}_\beta \mid \alpha, \beta \in \mathcal{T}\}\]
and \(\mathcal{I} : \mathcal{S} \rightarrow \mathcal{D}\) are typed total functions.

The collection \(\mathcal{D}\) is called the frame of \(\mathcal{A}\), the set \(\mathcal{D}_\alpha\) the universe of type \(\alpha\), the function \(\@\) the application operator, and the function \(\mathcal{I}\) the interpretation of constants.

We call a pre-\(\Sigma\)-structure \(\mathcal{A} := (\mathcal{D}, \@, \mathcal{I})\) functional, iff the following statement holds for all \(f, g \in \mathcal{D}_{\alpha \rightarrow \beta}\): \(f \equiv g\) if and only if for all \(a \in \mathcal{D}_\alpha\), we have that \(f@a \equiv g@a\). Note that functionality is a restriction on the function universes only.

**Remark 2.2.** The application operator \(\@\) in a pre-\(\Sigma\)-structure is an abstract version of function application. It is no restriction to exclusively use a binary application operator, which corresponds to unary function application, since we can define higher-arity application operators from the binary one by setting (“Currying”)
\[f@a^1, \ldots, a^n := (\ldots((f@a^1)@\ldots@a^n)\ldots)\]

**Example 2.3.** If we define \(\mathcal{A} \@ B := (A B)\) for \(A \in \wff_\alpha(\Sigma)\) and \(B \in \wff_\beta(\Sigma)\), then \(\@ : \wff_{\alpha \rightarrow \beta}(\Sigma) \times \wff_\alpha(\Sigma) \rightarrow \wff_\beta(\Sigma)\) is a total function. Thus \(\wff(\Sigma), @, \Id_\Sigma\) is a pre-\(\Sigma\)-structure. The intuition behind this example is that we can think of the formula \(A \in \wff_{\alpha \rightarrow \beta}(\Sigma)\) as a function \(\mathcal{A} : \wff_\alpha(\Sigma) \rightarrow \wff_\beta(\Sigma)\) that maps \(B\) to \((A B)\).

Analogously, we can define the pre-\(\Sigma\)-structure \((\wff(\Sigma), @, \Id_\Sigma)\) of closed formulae.

**Example 2.4.** The following is a (trivial) example for a functional pre-\(\Sigma\)-structure:

1. \((\{a\} \times \mathcal{T}, \@^a, \mathcal{T}^a)\), where \(a@^a \equiv a\) and \(\mathcal{T}^a(c) \equiv a\) for all constants \(c \in \Sigma\), is called the singleton pre-\(\Sigma\)-structure.

**Definition 2.5 (\(\Sigma\)-Homomorphism).** Let \(\mathcal{A} := (\mathcal{D}, \@^A, \mathcal{I}^A)\) and \(\mathcal{B} := (\mathcal{E}, \@^B, \mathcal{I}^B)\) be pre-\(\Sigma\)-structures. A \(\Sigma\)-homomorphism is a typed mapping \(\kappa : \mathcal{D} \rightarrow \mathcal{E}\) such that

1. \(\kappa \circ \mathcal{I} \equiv \mathcal{I}\).

2. For all types \(\alpha, \beta \in \mathcal{T}\), all \(f \in \mathcal{D}_{\alpha \rightarrow \beta}\), and \(g \in \mathcal{D}_\alpha\) we have: \(\kappa(f@^A g) \equiv \kappa(f@^B g)\).

The most important method for constructing \(\Sigma\)-structures with given properties in this thesis is well-known for algebraic structures and consists in building a suitable \(\Sigma\)-congruence and passing it to the quotient structure. We will now develop the formal basis for it.

**Definition 2.6 (\(\Sigma\)-Congruence).** Let \(\mathcal{A} := (\mathcal{D}, \@, \mathcal{I})\) be a pre-\(\Sigma\)-structure, then a typed equivalence relation \(\sim\) is called a \(\Sigma\)-congruence on \(\mathcal{A}\), iff \(f \sim f' \in \mathcal{D}_{\alpha \rightarrow \beta}\) and \(g \sim g' \in \mathcal{D}_\alpha\) imply \(f@g \sim f'@g'\). Let \(f \in \mathcal{D}_\alpha\), then the equivalence class of \(f\) modulo \(\sim\), \([f]_\sim\), is the set of all \(g \in \mathcal{D}_\alpha\) such that \(f \sim g\).

A \(\Sigma\)-congruence is called functional, iff for all types \(\alpha, \beta\) and all \(f, g \in \mathcal{D}_{\alpha \rightarrow \beta}\) the fact that \(f@a \sim g@a\) holds for all \(a \in \mathcal{D}_\beta\) implies \(f \sim g\). Note that, since \(\sim\) is a congruence, we also have the other direction, so we have \(f@a \sim g@a\) for all \(a \in \mathcal{D}_\beta\), iff \(f \sim g\).

**Lemma 2.7.** The \(\beta\) and \(\beta\eta\) equality relations \(\equiv_\beta\) and \(\equiv_{\beta\eta}\) are congruences on the pre-\(\Sigma\)-structures \(\wff(\Sigma)\) and \(\wff(\Sigma)\) by definition. Moreover, \(\beta\eta\)-equality is functional on \(\wff(\Sigma)\) and \(\wff(\Sigma)\).
Proof: The congruence properties are a direct consequence of the fact that \( \beta \eta \) reduction rules are defined to act on sub-term positions. We will establish functionality of \( \equiv_{\beta \eta} \) on \( \mathit{wff}(\Sigma) \) first and then use this to obtain the closure for closed formulas.

Let \( (A, \gamma \rightarrow \alpha, \mathcal{C}) \equiv_{\beta \eta} (B, \gamma \rightarrow \alpha, \mathcal{C}) \) for all \( \mathcal{C} \), then in particular, for any variable \( X \in \mathcal{V}_\gamma \) that is not free in \( A \) or \( B \), we have \( (\lambda X. A) \equiv_{\beta \eta} (\lambda X. B) \). By definition we have \( (A \equiv_{\beta \eta} (\lambda X. A, B) X) \equiv_{\beta \eta} B \).

To show functionality of \( \beta \eta \) on closed formulas, let \( A, B \in \mathit{wff}_\alpha \rightarrow \beta \eta (\Sigma) \), such that \( A \not\equiv_{\beta \eta} B \). Since \( \beta \eta \) is functional on \( \mathit{wff}(\Sigma) \), there must be a formula \( C \) with \( (A, C) \not\equiv_{\beta \eta} (B, C) \). Now let \( C' \) be a ground instance of \( C \), i.e., \( C' := \sigma(C) \), where \( \sigma \) is a closed substitution\(^6\), then we have \( (A, C') \not\equiv_{\beta \eta} (B, C') \). Thus we have shown that \( A \not\equiv_{\beta \eta} B \) entails \( A, C' \not\equiv_{\beta \eta} B, C' \), which gives us the assertion.

Definition 2.8 (Quotient Pre-\( \Sigma \)-Structure). Let \( A := (D, @, \mathcal{I}) \) be a pre-\( \Sigma \)-structure, \( \sim \) a \( \Sigma \)-congruence on \( A \), \( D_\sim := \{ [f]_\sim \mid f \in D_\alpha \} \), and \( \mathcal{I}^*(c_\alpha) := [\mathcal{I}(c_\alpha)]_\sim \) for all constants \( c_\alpha \) in \( \Sigma_\alpha \). Furthermore let \( @^* \) be defined by \( [f]_\sim @^* [a]_\sim := [f@a]_\sim \). To see that this definition only depends on equivalence classes of \( \sim \), consider \( f' \in [f]_\sim \) and \( g' \in [g]_\sim \), then \( [f@g]_\sim \equiv [f'@g']_\sim \equiv [f'@g']_\sim \). So \( @^* \) is well-defined and total, thus \( A/_\sim := (D^*_\sim, @^*_\sim, \mathcal{I}^*) \) is also a pre-\( \Sigma \)-structure. We call \( A/_\sim \) the quotient structure of \( A \) for the relation \( \sim \) and the typed function \( \pi_\sim : A \rightarrow A/_\sim \) that maps \( f \) to \([f]_\sim \) its canonical projection.

This definition is justified by the following theorem.

Theorem 2.9. Let \( A \) be a pre-\( \Sigma \)-structure and let \( \sim \) be an \( \Sigma \)-congruence on \( A \), then the canonical projection \( \pi_\sim \) is a surjective \( \Sigma \)-homomorphism. Furthermore, \( A/_\sim \) is functional, iff \( \sim \) is functional.

Proof: Let \( A := (D, @, \mathcal{I}) \) be a pre-\( \Sigma \)-structure. To convince ourselves that \( \pi_\sim \) is indeed a surjective \( \Sigma \)-homomorphism, we note that \( \pi_\sim \) is surjective by definition and \( T^* \equiv T \circ \mathcal{I} \). Now let \( f \in D_{\beta \rightarrow \alpha} \) and \( g \in \text{domain}(f) \subseteq D_\beta \), then \( g' \in [g]_\sim \) for all \( g' \in \text{domain}(f) \) and therefore \( [g]_\sim \equiv \pi_\sim(g) \in \text{domain}([f]_\sim) \equiv \text{domain}(\pi_\sim(f)) \) and \( \pi_\sim(f) @^* \pi_\sim(g) \equiv [f]_\sim @^* [g]_\sim \equiv [f@g]_\sim \equiv \pi_\sim(f@g) \).

The quotient construction trivialises ~ to (meta-)equality, so functionality of \( \sim \) is equivalent to functionality of \( A \). Formally we have \([f]_\sim \equiv [g]_\sim \), iff \( f \sim g \), iff \( f@a \sim g@a \), iff \([f@a]_\sim \equiv [g@a]_\sim \), iff \([f]_\sim @^* [a]_\sim \equiv [g]_\sim @^* [a]_\sim \) for all \( a \in D_\alpha \) and thus for all \([a]_\sim \in D_\sim \).

2.3 \( \Sigma \)-Structures

\( \Sigma \)-structures are pre-\( \Sigma \)-structures with a notion of evaluation for \( \mathit{wff}(\Sigma) \).

Definition 2.10 (\( \Sigma \)-Structure). Let \( A := (D, @, \mathcal{I}) \) be a pre-\( \Sigma \)-structure. A typed function \( \varphi : \mathcal{V} \rightarrow D \) is called a variable assignment into \( A \). We call a total typed mapping\(^7\) \( E : \mathcal{F}(\mathcal{V}; D) \rightarrow D \) an evaluation function for \( A \), iff for any assignment \( \varphi \) into \( A \), we have

1. \( E_{\varphi} \big|_{\mathcal{T}} \equiv T \) and \( E_{\varphi} \big|_{\mathcal{V}} \equiv \varphi \)
2. \( E_{\varphi} \) is a \( \Sigma \)-homomorphism
3. \( E_{\varphi}(A) \equiv E_{\varphi}(A) \), whenever \( \varphi \) and \( \psi \) coincide on \text{free}(A)
4. \( E_{\varphi}(B/X)[A] \equiv E_{\varphi}[\varphi(x)](B/X)[A] \)

We call \( A := (D, @, E) \) a \( \Sigma \)-structure, iff \((D, @, E)\) is a pre-\( \Sigma \)-structure and \( E \) is an evaluation function for \( A \). We call \( E_{\varphi}(A_\alpha) \in D_\alpha \) the denotation of \( A_\alpha \) in \( A \) for \( \varphi \).

If \( A \) is a closed formula, then \( E_{\varphi}(A) \) is independent of \( \varphi \), since \( \text{free}(A) \equiv \emptyset \). In these cases we sometimes drop the reference from \( E_{\varphi}(A) \) and simply write \( E(A) \).

\(^6\)This has to exist, since we have assumed infinitely many constants for each type \( \alpha \) in our signature \( \Sigma \), i.e., all types are inhabited.

\(^7\)We write \( \mathcal{F}(\mathcal{V}; D) \) for the set of functions \( f : \mathcal{V} \rightarrow D \)
2.3. \(\Sigma\)-STRUCTURES

Example 2.11. The singleton pre-\(\Sigma\)-structure is a \(\Sigma\)-structure if we take \(E(A) \equiv a\), where \(a\) is the (unique) member of \(D_a\).

For a detailed discussion on the closure conditions needed for the function univeses to be rich enough, we refer the reader to [And72a, And73].

Note that the pre-\(\Sigma\)-structure \(\text{wff}(\Sigma)\) from 2.3 cannot be made into a \(\Sigma\)-structure by providing an evaluation function, since there is no formula \(C \equiv T_\psi(\lambda X_\alpha B) \in \text{wff}_\alpha \beta(\Sigma)\), such that \(C @ A \equiv CA \equiv T_\psi[A/X](B)\). In particular, the “obvious” choice \(\lambda X_\alpha B\) for \(C\) does not work, since \((\lambda X_\alpha B)A \not\equiv T_\psi[A/X](B)\). In fact, if \(\text{wff}(\Sigma)\) were a \(\Sigma\)-structure, \(\beta\)-equality would have to be valid in \(\text{wff}(\Sigma)\) (cf. 2.17), which it clearly not.

Definition 2.12 (Quotient \(\Sigma\)-Structure). Let \(A \equiv (D, @, E)\) be a \(\Sigma\)-structure, \(\sim\) a \(\Sigma\)-congruence on \(A\) and let \(A/\sim := (D^\sim, @^\sim, I^\sim)\) be the quotient \(\Sigma\)-structure of \(A\), where \(I^\sim := I\big|_{D^\sim}\).

For any assignment \(\psi\) into \(A/\sim\), there exists an assignment \(\varphi\) into \(A\) such that \(\psi \equiv \pi_\sim \circ \varphi\), since \(\pi_\sim\) is a surjective \(\Sigma\)-homomorphism. So we can define \(E^\sim\) as \(\pi_\sim \circ E_\psi\), and call \(A/\sim := (D^\sim, @^\sim, E^\sim)\) the quotient \(\Sigma\)-structure of \(A\) modulo \(\sim\).

Theorem 2.13 (Quotient \(\Sigma\)-Structure). Let \(A\) be a \(\Sigma\)-structure and let \(\sim\) be a \(\Sigma\)-congruence on \(A\), then \(A/\sim\) is a \(\Sigma\)-structure.

Proof: We prove that \(E^\sim\) is a legal value function by verifying the conditions in 2.10: Let \(\varphi\) and \(\psi\) be assignments, such that \(\psi \equiv \pi_\sim \circ \varphi\), then

1. \(E^\sim_\psi \big|_{D^\sim} \equiv (\pi_\sim \circ E_\psi) \big|_{D^\sim} \equiv \pi_\sim \circ E_\psi \big|_{D} \equiv \pi_\sim \circ I \equiv I^\sim\) and \(E^\sim_\psi \big|_{D^\sim} \equiv (\pi_\sim \circ E_\psi) \big|_{D^\sim} \equiv \pi_\sim \circ E_\psi \big|_{D} \equiv \pi_\sim \circ \psi \equiv \varphi\).

2. \(E^\sim_\psi \equiv \pi_\sim \circ E_\psi\) is a \(\Sigma\)-homomorphism, since \(\pi_\sim\) and \(E_\psi\) are.

3. \(E^\sim_\psi(A) \equiv \{E_\psi(A)\}_\sim \equiv \{E^\sim_\psi(A)\}_\sim \equiv E^\sim_\psi(A)\), iff \(\varphi\) and \(\varphi'\) coincide on \(\text{free}(A)\), since this entails that \(\psi\) and \(\psi'\) do too.

4. \(E^\sim_\psi([B/X]) \equiv \{E_\psi([B/X])\}_\sim \equiv \{E^\sim_\psi([B/X])\}_\sim \equiv E^\sim_\psi([E^\sim_\psi([B/X])\]) \equiv \{E^\sim_\psi([B/X])\}_\sim \equiv E^\sim_\psi([B/X])\), and therefore \(\pi_\sim \circ \varphi \equiv \{E_\psi(B)/X\}_\sim \equiv \varphi \equiv \{E^\sim_\psi(B)/X\}_\sim\).

Definition 2.14 (Term Structures for \(\Sigma\)). Let \(\text{c\text{wff}}(\Sigma)\big|_{\beta}\) be the collection of closed well-formed formulae in \(\beta\)-normal form and \(A @ B\) be the \(\beta\)-normal form of \(A \cdot B\). For the definition of an evaluation function let \(\varphi\) be an assignment into \(\text{c\text{wff}}(\Sigma)\big|_{\beta}\). Note that \(\sigma := \varphi\big|_{\text{free}(A)}\) is a substitution, since \(\text{free}(A)\) is finite. Thus, we can choose \(E^\beta_\sigma(A) := \sigma(A)\big|_{\beta}\), where \(A\big|_{\beta}\) is the \(\beta\)-normal form of \(A\). We call \(TS(\Sigma)^\beta := (\text{c\text{wff}}(\Sigma)\big|_{\beta}, @^\beta, E^\beta)\) the \(\beta\)-term structure for \(\Sigma\).

Analogously, we can define \(TS(\Sigma)^\eta := (\text{c\text{wff}}(\Sigma)\big|_{\eta}, @^\eta, E^\eta)\) the \(\eta\)-term structure for \(\Sigma\).

The name term structure in the previous definition is justified by the following lemma.

Lemma 2.15. \(TS(\Sigma)^\beta\) is a \(\Sigma\)-structure and \(TS(\Sigma)^\beta\) is a functional \(\Sigma\)-structure.

Proof: Note that constants are \(\beta\)-normal forms, therefore \(TS(\Sigma)^\beta\) is the quotient structure of \(\text{c\text{wff}}(\Sigma)\) for the congruence \(\equiv^\beta\). As we have remarked in 2.11, \(\text{wff}(\Sigma)\) is not a \(\Sigma\)-structure, so we cannot use 2.13, but have to convince ourselves directly that \(TS(\Sigma)^\beta\) is a \(\Sigma\)-structure by verifying the conditions of 2.10. The first three are direct consequences of the definition of \(E^\beta\) as substitution application.

1. \(E^\beta_\sigma \big|_{D} \equiv I^\beta \equiv I^\Sigma\) and \(E^\beta_\sigma \big|_{D} \equiv \varphi\)
2. \(E^\beta_\sigma\) is a \(\Sigma\)-homomorphism
3. \(E^\beta_\varphi(A) \equiv \sigma(A) \equiv \sigma'(A) \equiv E^\beta_{\varphi'}(A)\), iff \(\varphi\) and \(\varphi'\) coincide on free(\(A\)).

4. \(E^\beta_{\varphi}(\langle B / X \rangle A) \equiv \sigma(\langle B / X \rangle A) \equiv [\sigma(B)](\sigma'(A)) \equiv [\sigma(B)](\sigma''(A)) \equiv E^\beta_{\varphi'}(\langle B / X \rangle A)\), where \(\sigma'\) is \(\sigma, [X / X]\).

Since \(\equiv_{\beta}\) is a super-relation of \(\equiv_{\beta}\), a similar argument shows that \(TS(\Sigma)_{\beta}\) is a \(\Sigma\)-structure.

Furthermore, \(\equiv_{\beta}\) is a functional \(\Sigma\)-congruence on \(\text{wff}(\Sigma)\) (cf. 2.7), so we know by 2.9 that \(TS(\Sigma)_{\beta}\) is functional.

Remark 2.16. Note that \(TS(\Sigma)_{\beta}\) is not a functional \(\Sigma\)-structure since, e.g., \((\lambda X \cdot Y \cdot X)_{\gamma \rightarrow \delta} Y\) \(\not\equiv Y\) in \(TS(\Sigma)_{\beta}\) but \((\lambda X \cdot Y \cdot X) \not\equiv Y\).

In a general structure \(\mathcal{A} := (\mathcal{D}, @, \mathcal{E})\), the constants are given a meaning by the interpretation function \(\mathcal{I}: \Sigma \rightarrow \mathcal{D}\), and variables get their meaning by assignments \(\varphi: \mathcal{V} \rightarrow \mathcal{D}\). Furthermore, the evaluation function has to respect instantiation as in first-order logic. This is enough to ensure soundness of \(\beta\)-equality.

Lemma 2.17 (Soundness of \(\beta\)-equality). Let \(\mathcal{A} := (\mathcal{D}, @, \mathcal{E})\) be a \(\Sigma\)-structure and \(\varphi\) an assignment into \(\mathcal{A}\), then \(E_{\varphi}((\lambda X \cdot A)B) \equiv E_{\varphi}(\langle B / X \rangle A)\) provided that \(X\) is not bound in \(A\).

Proof: By the definition of \(\Sigma\)-structures, we have \(E_{\varphi}((\lambda X \cdot A)B) \equiv E_{\varphi}(\lambda X \cdot A)@E_{\varphi}(B) \equiv E_{\varphi}(\langle B / X \rangle A)\). \(\square\)

2.4 Functional \(\Sigma\)-structures

For functional \(\Sigma\)-structures, there is another way to define evaluation: since well-formed formulae are inductively built up from constants and variables we can extend \(\varphi\) and \(\mathcal{I}\) to a \(\Sigma\)-homomorphism on well-formed formulae.

Definition 2.18 (Homomorphic Extension). Let \(\mathcal{A} := (\mathcal{D}, @, \mathcal{I})\) be a functional pre-\(\Sigma\)-structure and let \(\varphi\) be an assignment into \(\mathcal{A}\). Then the homomorphic extension \(\mathcal{I}_{\varphi}\) of \(\varphi\) to \(\text{wff}(\Sigma)\) is inductively defined to be a typed partial function \(\mathcal{I}_{\varphi}: \text{wff}(\Sigma) \rightarrow \mathcal{D}\), such that

1. \(\mathcal{I}_{\varphi}(X) \equiv \varphi(X)\), if \(X\) is a variable,
2. \(\mathcal{I}_{\varphi}(c) \equiv \mathcal{I}(c)\), if \(c\) is a constant,
3. \(\mathcal{I}_{\varphi}(\langle A \circ B \rangle) \equiv \mathcal{I}_{\varphi}(A)@\mathcal{I}_{\varphi}(B)\),
4. \(\mathcal{I}_{\varphi}(\lambda X \cdot A_{\beta})\) is the function in \(\mathcal{D}_{\alpha \rightarrow \beta}\), such that \(\mathcal{I}_{\varphi}(\lambda X \cdot A_{\beta})@: \mathcal{I}_{\varphi}(\langle \lambda X \cdot A \rangle B)\). Note that this function is unique, since we have assumed \(\mathcal{A}\) to be functional.

We have to assume that the universes of functions \(\mathcal{D}_{\alpha \rightarrow \beta}\) are rich enough to contain a value for all \(A_{\alpha \rightarrow \beta} \in \text{wff}_{\alpha \rightarrow \beta}(\Sigma)\) for this construction to yield a total function.

Lemma 2.19. Let \(\mathcal{A} := (\mathcal{D}, @, \mathcal{I})\) be a functional pre-\(\Sigma\)-structure, then \(\mathcal{E}: \varphi \mapsto \mathcal{I}_{\varphi}\) is an evaluation function for \(\mathcal{A}\).

Proof: To prove the assertion, we have to show the conditions of 2.10. The first one is trivially met by construction, the second is a direct consequence of the fact that \(\mathcal{I}_{\varphi} \circ \text{Id}_\Sigma \equiv \mathcal{I} \circ \text{Id}_\Sigma \equiv \mathcal{I}\) on \(\Sigma\).

For the third condition, we prove that the value of a function depends only on its free variables (by induction on the structure of \(A\)). The only interesting case is the one, where \(A\) is an abstraction, since the assertion is trivial for constants and variables, and a simple consequence of the inductive hypothesis for applications. So let \(A := (\lambda X \cdot B)\), then \(\mathcal{I}_{\varphi}(A)@: \mathcal{I}_{\varphi}(\langle \lambda X \cdot A \rangle B)\) by inductive hypothesis, since \(\varphi, [a / X]\) and \(\psi, [a / X]\) coincide on the free variables of \(B\). Thus we obtain the assertion from the definition of \(\mathcal{I}_{\varphi}\).
2.4. FUNCTIONAL Σ-STRUCTURES

Finally, we prove the fourth condition by induction on the structure of $A$. If $A$ is a constant or variable, then the assertion is trivial. The case where $A$ is the application $CD$ is entailed by the fact, that substitution and homomorphic extension are defined inductively on the structure of applications: we have

$$I_\varphi([B/X](CD)) \equiv I_\varphi([B/X]C)@I_\varphi([B/X]D)$$

$$\equiv I_{i\varphi,[C,B]/X}((C)@I_{i\varphi,[D,B]/X}(D))$$

$$\equiv I_{i\varphi,[X,B]/X}(C\ D)$$

If $A \equiv (\lambda Y, D)$ and $\psi \equiv \varphi, [a/Y]$, then

$$I_\varphi([B/X]A)[a] \equiv I_\varphi(\lambda Y. [B/X]D)[a] \equiv I_{i\varphi,[B/X]D}(\lambda Y, D)[a]$$

by inductive hypothesis. Note that $\psi$ and $\varphi$ coincide on the free variables of $A$, therefore by the third condition, which we have proven above, we have $I_{i\varphi,[X,B]/X}(D) \equiv I_{i\varphi,[X,B]/X}((\lambda Y, D)[a]$, which implies the assertion, since $A$ is functional.

In fact, for functional $\Sigma$-structures, the two notions of evaluation coincide:

**Lemma 2.20 (Evaluation in functional $\Sigma$-Structures).** If $A := (\mathcal{D}, @, \mathcal{E})$ is a functional $\Sigma$-structure, then $E_\varphi \equiv I_\varphi$ for any assignment $\varphi$ into $A$.

**Proof:** Let $A \in \text{wff}(\Sigma)$, we prove the assertion by induction over the size of $A$. The assertion is trivial, if $A$ is a constant or variable, and a simple consequence of the inductive hypothesis, if $A$ is an application. So let $A := (\lambda X, B)$, furthermore let $Y$ be a variable not in free($A$) and $\psi := \varphi, [a/Y]$. Then

$$E_\varphi(A)[a] \equiv E_\varphi(A)[a] \equiv E_\varphi(A)[E_\varphi(Y) \equiv E_\varphi(\lambda Y)[X]D)$$

since $\beta$-equality is sound in $\Sigma$-structures. Now $[Y/X]B$ is smaller than $A$, so we can use the inductive hypothesis to obtain

$$E_\varphi(A)[a] \equiv I_{i\varphi}(\lambda Y)[X]B) \equiv I_{i\varphi}(\lambda Y) \equiv I_{i\varphi}(A)\varnothing I_{i\varphi}(\lambda Y)[X] \equiv I_{i\varphi}(A)[a]$$

which entails the assertion since $A$ is functional.

**Lemma 2.21.** Let $A := (\mathcal{D}, @, \mathcal{E})$ be a functional $\Sigma$-structure and $X$ be a variable that is not free in $A$, then $E_\varphi(\lambda X. A X) \equiv E_\varphi(A)$ for all assignments $\varphi$ into $A$.

**Proof:** With 2.103 and the fact that $X$ is not free in $A$ we have

$$E_\varphi(\lambda X. A X)[a] \equiv E_\varphi[A/X]B)[a] \equiv E_\varphi(A)[a]$$

which implies the assertion $E_\varphi(\lambda X. A X) \equiv E_\varphi(A)$, as $A$ is functional.

We now specialise the notion of $\Sigma$-structures to the classical general model semantics for $\Lambda^\rightarrow$.

**Definition 2.22 ($\Sigma$-Algebra).** A pre-$\Sigma$-algebra $A := (\mathcal{D}, I)$ is a pre-$\Sigma$-structure $\mathcal{D}$, such that $\mathcal{D}_{\alpha \rightarrow \beta} \subseteq \mathcal{F}(\mathcal{D}_{\alpha}; \mathcal{D}_{\beta})$ and $f@a \equiv f(a)$. A pre-$\Sigma$-algebra is called full, iff $\mathcal{D}_{\alpha \rightarrow \beta} \equiv \mathcal{F}(\mathcal{D}_{\alpha}; \mathcal{D}_{\beta})$. We call a pre-$\Sigma$-algebra a $\Sigma$-algebra, if it is a $\Sigma$-structure.

**Remark 2.23.** Note that pre-$\Sigma$-algebras are functional, since they are defined as structures of mathematical functions. On the other hand, for any functional $\Sigma$-structure $A$, we can define an isomorphic $\Sigma$-algebra $A'$.

**Proof:** For a functional $\Sigma$-structure $A := (\mathcal{D}, @, \mathcal{E})$ we define a $\Sigma$-algebra $A' := (\mathcal{D}', I')$ and a bijective $\Sigma$-homomorphism $\kappa : A \rightarrow A'$ by an induction on the type:

- $\mathcal{D}'_{\alpha} := \mathcal{D}_{\alpha}$ for all $\alpha \in \mathcal{B}^\rightarrow$ and $\kappa := \text{Id}_{\mathcal{D}}$; obviously $\kappa$ is bijective.
- $\mathcal{D}'_{\alpha \rightarrow \beta} := \kappa(\mathcal{D}_{\alpha \rightarrow \beta})$ and $\kappa(f) := \kappa(o)(@f)\circ\kappa^{-1}$ for $f \in \mathcal{D}_{\alpha \rightarrow \beta}$. Note that with this construction $\kappa$ is a homomorphism, since

$$\kappa(f)(\kappa(a)) \equiv \kappa(f(\kappa^{-1}(\kappa(a)))) \equiv \kappa(f(a))$$
\[ \kappa \text{ is surjective by construction and injective, since } A \text{ is functional: If } f \neq g \in D_{\alpha} \to \beta, \text{ then there is an } a \in D_{\alpha}, \text{ such that } f(a) \neq g(a), \text{ in particular, we have} \]
\[ \kappa(f(a)) \equiv \kappa(f) \circ \kappa(a) \neq \kappa(f) \circ \kappa(a) \equiv \kappa(g(a)) \]

since \( \kappa \) is injective on \( D_{\beta} \). Thus and therefore \( \kappa(f) \neq \kappa(g) \), since \( \kappa(f), \kappa(g) \in F(D_{\alpha}; D_{\beta}) \).

Now, we only have to choose \( T' := \kappa \circ T \) to complete the construction of \( A' \).

\[ \square \]

As a consequence, we can always consider functional \( \Sigma \)-structures as \( \Sigma \)-algebras.

### 2.5 \( \Sigma \)-Models

The semantic notions so far are independent of the set of base types. Now, we specialise these to obtain a notion of models by requiring specialised behaviour on the type \( o \) of truth values. For this we use the notion of a \( \Sigma \)-valuation, which intuitively gives a truth-value interpretation to the domain \( D_{o} \) of a \( \Sigma \)-structure, which is consistent with the intuitive interpretations of the logical constants. Since models are semantic entities that are constructed first of all to make a statement about the truth or falsity of a formula, the requirement that there exists a \( \Sigma \)-valuation is perhaps the most general condition under which one wants to speak of a model. Thus we will define our most general notion of semantics as \( \Sigma \)-structures that have \( \Sigma \)-valuations.

**Definition 2.24 (\( \Sigma \)-Model).** Let \( A := (D, @, E) \) be a \( \Sigma \)-structure, then a surjective total function \( v : D_{\alpha} \to \{ \top, \bot \} \) such that

1. \( v(\neg a) @ a \equiv \top \text{ iff } v(a) \equiv \bot \),
2. \( v(a @ b @ a) @ b \equiv \top \text{ iff } v(a) \equiv \top \text{ or } v(b) \equiv \top \),
3. \( v(a @ b @ f) \equiv \top \text{ iff } v(f @ a @ b) \equiv \top \text{ for each } a \in D_{\alpha} \)

is called a \( \Sigma \)-valuation for \( A \) and \( M := (D, @, E, v) \) is called a \( \Sigma \)-model. The class of all \( \Sigma \)-models is denoted by \( \mathcal{M}_{\beta} \).

We say that an assignment \( \varphi \) satisfies a formula \( A \in wff_{\alpha}(\Sigma) \) in \( \mathcal{M} (\models_{\varphi} A) \), iff \( v(\varphi(A)) \equiv \top \) and that \( A \) is valid in \( \mathcal{M} \), iff \( \mathcal{M} \models_{\varphi} A \) for all assignments \( \varphi \). Finally, we say that \( \mathcal{M} \) is a \( \Sigma \)-model for a set \( H \subseteq wff_{\alpha}(\Sigma) \) (\( \mathcal{M} \models H \)) iff \( \mathcal{M} \) satisfies all \( A \in H \).

**Lemma 2.25 (Truth and Falsity in \( \Sigma \)-Models).** Let \( \mathcal{M} := (D, @, E, v) \) be a \( \Sigma \)-model and \( \varphi \) an assignment. Furthermore let \( T_{a} := A_{a} \lor \neg(A_{a}) \) for some \( A_{a} \in wff_{\alpha} \) and let \( F_{a} := \neg T_{a} \). Then \( v(\varphi(T_{a})) \equiv \top \) and \( v(\varphi(F_{a})) \equiv \bot \).

**Proof:** We have \( v(\varphi(T_{a})) \equiv \top \text{ iff } v(\varphi(A_{a}) \lor \neg(A_{a})) \equiv \top \). Evaluation shows that this statement is equivalent to \( v(\varphi(A_{a})) \equiv \top \text{ or } v(\varphi(A_{a})) \equiv \bot \), which is valid since \( \varphi : V_{\alpha} \to D_{\alpha} \) and \( v : D_{\alpha} \to \{ \top, \bot \} \) are total functions.

Note further that \( v(\varphi(F_{a})) \equiv \bot \) evaluates to \( v(\varphi(T_{a})) \equiv \top \), which we already know. \[ \square \]

**Remark 2.26.** We only constrain the functional behaviour of the values of the logical constants. In particular this does not fully specify these values, since

- \( \mathcal{M} \) need not be functional,
- and there can be more than two truth values.

**Definition 2.27 (Properties \( \varphi, \beta, \text{ and } \{ \} \)).** Given a \( \Sigma \)-model \( \mathcal{M} := (D, @, E, v) \), we say that \( \mathcal{M} \) has property

\( \downarrow \text{ iff } \mathcal{M} \) is functional,
2.6. LEIBNIZ EQUALITY

$q$ iff $M$ has property $f$ and for all $\alpha \in T$ there is a function $q^\alpha \in D_\alpha \to M$, such that for all $a, b \in D_\alpha$ holds $v(q^\alpha \circ a \circ b) = T$ iff $a \equiv b$.

$b$ iff $D_\alpha$ has at most two elements. Note that $D_\alpha$ must always have at least the two elements $E_\alpha(T_\alpha)$ and $E_\alpha(F_\alpha)$ by Lemma 2.25, so we can assume without loss of generality that $D_\alpha \equiv \{E_\alpha(F_\alpha) \equiv F, E_\alpha(T_\alpha) \equiv T\}$ and that $v$ is the identity function.

**Definition 2.28 (Specialised Model Classes).** We define special classes of $\Sigma$-models depending on the validity of the properties $f, q$ and $b$. Thus we obtain the specialised classes of $\Sigma$-models $M_{\beta f}, M_{\beta q}, M_{\beta f}, M_{\beta q}$ by requiring that the properties specified in the index are valid.

**Remark 2.29 (Property $q$).** The idea of property $q$ is to ensure for all types $\alpha$ that the intuitive equality relation $q_\alpha \to \alpha \to \alpha$, i.e., a functional congruence relation, is contained in $D_\alpha \to \alpha \to \alpha$. This ensures the existence of unit sets in the domains $D_\alpha \to \alpha$, which in turn makes Leibniz equality the intended equality relation, as the membership in this unit sets can be used as a strong argument in order to distinguish between different elements of $D_\alpha$. For a detailed discussion see [And72a].

Property $q$ as stated in [BK97a] is not correct. Whereas the motivation for the formulation there was the same as sketched above, this formulation does unfortunately not ensure functionality (although this was intended), which is needed in the proof of Lemma 2.35. We want to thank an unknown referee of the Journal of Symbol Logic for pointing to this problem.

**Remark 2.30 (Property $q$ in Henkin models).** As Peter Andrews has noted in [And72a], Leon Henkin unintentionally introduced $M_{\beta q}$ in [Hen50] instead of the class of Henkin models in the sense below. An element of $M_{\beta q}$ does not necessarily have property $q$ and as Andrews has shown in [And72a], a consequence is, that such an element may lack the principle of functional extensionality $\text{EXT}_L^{\alpha \to \beta}$, which he corrected by introducing property $q$.

**Definition 2.31 ($\Sigma$-Henkin models).** A functional $\Sigma$-model is called a $\Sigma$-Henkin model, iff it has properties $q$ and $b$. The class of all $\Sigma$-Henkin models is denoted by $\mathcal{H}$ or $M_{\beta f}$. If furthermore, all domains $D_\alpha \to \beta$ are full then we call $H$ a $\Sigma$-standard model ($\mathcal{S}$).

Now let us extend the notion of a quotient structure to $\Sigma$-models.

**Definition 2.32 (Quotient $\Sigma$-model).** Let $M := (D, \circ, E, v)$ be a $\Sigma$-model, $\sim$ a congruence on the corresponding $\Sigma$-structure $A := (D, \circ, E)$, and $A/\sim$ be the quotient $\Sigma$-structure of $A := (D, \circ, E)$ modulo $\sim$ as defined in 2.12.

If $v(A) \equiv v(B)$ for all $A, B \in \text{wff}_\Sigma(A)$ with $A \sim B$, then $\sim$ is called a congruence for $M$. Then $M/\sim := (D/\sim, \circ/\sim, E/\sim, v/\sim)$ is called the quotient $\Sigma$-model of $M$ modulo $\sim$, if $v/\sim([a]_\sim) \equiv v(a)$ for all $a \in D/\sim$.

**Remark 2.33.** Note the importance of the additional requirement for functional congruence relations stated in 2.32. Without this requirement the quotient $\Sigma$-models are not well-defined.

**Lemma 2.34.** Let $M$ be a $\Sigma$-model, $H \subseteq \text{wff}_\Sigma(A)$, and $\sim$ be a congruence for $M$, then $M/\sim \models_{\varphi} H$, iff $M \models_{\varphi} H$.

**Proof:** Let $A_{\alpha} \in H$. We have $v/\sim(E^\varphi_{\alpha}(A_{\alpha})) \equiv v/\sim([E^\varphi_{\alpha}(A_{\alpha})]_\sim) \equiv v(E^\varphi_{\alpha}(A_{\alpha})).$ 

2.6 Leibniz Equality

**Definition 2.35 (Extensionality for Leibniz equality).** We call the following formula schemata

\begin{align*}
\text{EXT}_L^{\alpha \to \beta} & := \forall F_{\alpha \to \beta}, \forall G_{\alpha \to \beta} (\forall X_{\beta}, F \equiv G X) \Rightarrow F \equiv \beta G \\
\text{EXT}_L & := \forall A_{\alpha}, \forall B_{\alpha} (A \iff B) \iff A \equiv \beta B
\end{align*}

the axioms of full extensionality for Leibniz equality; we refer to the first as axiom of functional extensionality and to the latter formula as the extensionality axiom for truth values. Note that $\text{EXT}_L^{\alpha \to \beta}$ specifies functionality of the relation denoted by the Leibniz formula $\equiv$. We will use the terms functionality and extensionality interchangeably.
Lemma 2.36 (Leibniz Equality in $\Sigma$-models). Let $\mathcal{M} := (\mathcal{D}, @, \mathcal{E}, v)$ be a $\Sigma$-model and $\varphi$ be an assignment.

1. If $\mathcal{E}_\varphi(\mathbf{A}) \equiv \mathcal{E}_\varphi(\mathbf{B})$, then $v(\mathcal{E}_\varphi(\mathbf{A} \vdash^\varphi \mathbf{B})) \equiv \top$.

2. If $\mathcal{M} \in \mathfrak{M}_{2\beta}$ and $v(\mathcal{E}_\varphi(\mathbf{A} \vdash^\varphi \mathbf{B})) \equiv \top$, then $\mathcal{E}_\varphi(\mathbf{A}) \equiv \mathcal{E}_\varphi(\mathbf{B})$.

3. If $\mathcal{M} \in \mathfrak{M}_{2\beta}$ and $v(\mathcal{E}_\varphi(\mathbf{A} \vdash^\varphi \mathbf{B})) \equiv \top$, then $\mathcal{E}_\varphi(\mathbf{A}) \equiv \mathcal{E}_\varphi(\mathbf{B})$.

Proof: Let $a, b \in \mathcal{D}_a$ and $\psi := \varphi; [a/X], [b/Y]$.

1. We show that $v(\mathcal{E}_\varphi(\equiv^\varphi)@a@b) \equiv \top$, if $a \equiv b$, which entails the assertion. By definition $\mathcal{E}_\varphi(\equiv^\varphi) \equiv \mathcal{E}_\rho(\lambda X. \forall Y. (\forall P. P X \Rightarrow P Y))$ and thus $\mathcal{E}_\varphi(\equiv^\varphi)@a@b \equiv \mathcal{E}_\varphi(\forall P. P X \Rightarrow P Y)$.

Now let $r \in \mathcal{D}_{a\rightarrow b}$, then $v(\mathcal{E}_\varphi(\equiv^\varphi)[P](X)) \equiv r@a \equiv F$ or $v(\mathcal{E}_\varphi(\equiv^\varphi)[P](Y)) \equiv r@b \equiv r@a \equiv \top$, since $v$ is total and $a \equiv b$. So we see that $v(\mathcal{E}_\varphi(\equiv^\varphi)@a@b) \equiv v(\mathcal{E}_\varphi(\equiv^\varphi)[P](\forall X \Rightarrow P Y)) \equiv \top$ for all $r \in \mathcal{D}_{a\rightarrow b}$, which yields the assertion.

2. First note that by property $b$ we have $\mathcal{D}_a \equiv \{T, F\}$ and $v$ is the identity function on $\mathcal{D}_a$.

Let us assume that $v(\mathcal{E}_\varphi(\mathbf{A} \vdash^\varphi \mathbf{B})) \equiv \mathcal{E}_\varphi(\forall P. P A \Rightarrow P B) \equiv \top$ but $\mathcal{E}_\varphi(\mathbf{A}) \not\equiv \mathcal{E}_\varphi(\mathbf{A})$, which means that either $\mathcal{E}_\varphi(\mathbf{A}) \equiv \top$ and $\mathcal{E}_\varphi(\mathbf{B}) \equiv F$ or vice versa. In the first case we choose a predicate $r := \mathcal{E}_\varphi(\lambda X. \forall X)$ and get from the first assumption that $\mathcal{E}_\varphi(\equiv^\varphi)[P](\forall A) \equiv \mathcal{E}_\varphi(\equiv^\varphi)[P](\forall B) \equiv \mathcal{E}_\varphi(\equiv^\varphi)[P](\top) \equiv \mathcal{E}_\varphi(\equiv^\varphi)[P](\forall B) \equiv \top$, which gives us the contradiction. Note that $P$ does not occur free in $\mathbf{A} \lor \mathbf{B}$ by definition of $\equiv$.

The second case is analogous with $r := \mathcal{E}_\varphi(\lambda X. \forall X)$.

3. We show that if $v(\mathcal{E}_\varphi(\equiv^\varphi)@a@b) \equiv \top$, then $a \equiv b$, which entails the assertion. Suppose $a \not\equiv b \in \mathcal{D}_a$ and $r \equiv q^\varphi@a$, where $q^\varphi \in \mathcal{D}_{a\rightarrow a \rightarrow \top}$ is the function guaranteed by property $q$.

We know that $q^\varphi@a@a \equiv T$ and $q^\varphi@a@b \equiv F$, since $a \not\equiv b$ by assumption. Hence $v(\mathcal{E}_\varphi(\equiv^\varphi)@a@b) \equiv v(\mathcal{E}_\varphi(\forall P. P X \Rightarrow P Y)) \equiv F$ for $\psi := \varphi; [a/X], [b/Y]$, since $v(\mathcal{E}_\varphi(\equiv^\varphi)[P](\forall X \Rightarrow P Y)) \equiv F$, as $v(\mathcal{E}_\varphi(\equiv^\varphi)[P](P X)) \equiv q^\varphi@a@a \equiv r@a \equiv \top$ and $v(\mathcal{E}_\varphi(\equiv^\varphi)[P](P Y)) \equiv q^\varphi@a@b \equiv r@b \equiv F$.

Lemma 2.37 (Extensionality in $\Sigma$-models).

1. There exists a model $\mathcal{M} \in \mathfrak{M}_\beta$ which is not functional.

2. There exists a model $\mathcal{M} \in \mathfrak{M}_{2\beta}$ for which EXT$_L^\alpha$ is not valid.

3. There exists a model $\mathcal{M} \in \mathfrak{M}_{2\beta}$ for which EXT$_L^\alpha$ is not valid.

4. EXT$_L^\alpha$ is valid in model $\mathcal{M}$, if $\mathcal{M} \in \mathfrak{M}_{2\beta}$.

5. EXT$_L^\alpha$ is valid in model $\mathcal{M}$, if $\mathcal{M} \in \mathfrak{M}_{2\beta}$.

As a consequence the following table characterizes the different properties of the introduced semantical structures. If a formula is valid for a certain semantical structure we use a '+' and a '-' otherwise. Each entry is further marked with a justification referring to one of the above statements.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>$\mathfrak{M}_\beta$</th>
<th>$\mathfrak{M}_{2\beta}$</th>
<th>$\mathfrak{M}_{2\beta}$</th>
<th>$\mathfrak{M}_{3\beta}$</th>
<th>$\mathfrak{M}_{2\beta\beta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT$_L^\alpha$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>EXT$_L^\alpha$</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Proof: Let $v$ be a $\Sigma$-valuation and let $\mathcal{M}$ be a $\Sigma$-model based on the $\beta$-termstructure $TS(\Sigma)^\beta$ for $\Sigma$, i.e., $\mathcal{M} := (\text{eval}(\Sigma), \beta, \mathcal{E}, v)$. Note that $\mathcal{M}$ need not to be functional by Remark 2.16 and 2.26.
2.6. LEIBNIZ EQUALITY

For the proof of 2, we refer to [And72a], where Andrews constructs a functional Σ-Model (actually a \( M \in \mathfrak{M}_\beta \)) that lacks the principle of functional extensionality of Leibniz equality.

For 3, note that EXT \( S \) can only be valid if \( D_\alpha \equiv \{ \alpha, \beta \} \), which is not required for \( M \in \mathfrak{M}_\beta \). For a concrete example of a \( M \in \mathfrak{M}_\beta \), which lacks EXT \( S \), see 3.29 (for \( \mathfrak{M}_\alpha \)).

Next we consider 4. Let \( \psi := \varphi \circ [F], [g/G] \). From \( \mathcal{V}_\psi (A \circ A \circ [F] \circ [g/G]) = \mathfrak{T} \) we get that for all \( a \in D_\alpha \), \( \mathcal{V}_\psi ([a/A] \circ (F \circ [g/G]) = \mathfrak{T} \). By lemma 2.36(3) we can conclude that \( \mathcal{E}_\psi ([a/A] \circ (F \circ [g/G]) = \mathcal{E}_\psi ([a/A] \circ (G \circ [g/G]) \) for all \( a \in D_\alpha \) and hence \( \mathcal{E}_\psi ([a/A] \circ (F \circ [g/G]) = \mathcal{E}_\psi ([a/A] \circ (G \circ [g/G]) \) for all \( a \in D_\alpha \). By definition of property \( q \), which includes property \( j \) and thus ensures functionality, we get \( \mathcal{E}_\psi (G) \equiv \mathcal{E}_\psi (F) \equiv \mathfrak{T} \). This finally gives us that \( \mathcal{V}_g (F = \alpha \rightarrow \beta \circ G) = \mathfrak{T} \) with lemma 2.36(1) and in the right to left direction the assertion follows with 2.36(2).

Next we discuss the role of Leibniz equality within the different semantic structures.

**Theorem 2.38 (Properties of Leibniz Equality).** Let \( M \) be a \( \Sigma \)-model. For all assignments \( \varphi \) and all terms \( A, B, C \in \mathfrak{W}_{\alpha, \beta}(\Sigma) \) and \( F, G \in \mathfrak{W}_{\alpha, \beta}(\Sigma) \) we have:

\( \mathfrak{M}_\beta \) If \( M \in \mathfrak{M}_\beta \), then \( \mathcal{E}_\psi (\alpha \sim \alpha) \) is an equivalence relation on \( D_\alpha \) with respect to \( v \). In particular:

- \( \text{ref} \): \( v(\mathcal{E}_\psi (A \sim \alpha A)) \equiv \mathfrak{T} \).

- \( \text{sy} \): \( v(\mathcal{E}_\psi (A \sim \equiv B)) \equiv \mathfrak{T} \), then \( v(\mathcal{E}_\psi (B \equiv \alpha A)) \equiv \mathfrak{T} \).

- \( \text{tr} \): \( v(\mathcal{E}_\psi (A \equiv \alpha B)) \equiv \mathfrak{T} \) and \( v(\mathcal{E}_\psi (B \equiv \alpha C)) \equiv \mathfrak{T} \), then \( v(\mathcal{E}_\psi (A \equiv \alpha C)) \equiv \mathfrak{T} \).

\( \mathfrak{M}_\beta \) If \( M \in \mathfrak{M}_\beta \), then \( \mathcal{E}_\psi (\equiv \alpha) \) is a congruence relation on \( D_\alpha \) with respect to \( v \). In particular:

- \( \text{co} \): \( v(\mathcal{E}_\psi (A \equiv \alpha B)) \equiv \mathfrak{T} \), then \( v(\mathcal{E}_\psi (F A \equiv \beta F B)) \equiv \mathfrak{T} \).

\( \mathfrak{M}_\beta \) If \( M \in \mathfrak{M}_\beta \), then \( \mathcal{E}_\psi (\equiv \alpha) \) is a functional congruence relation on \( D_\alpha \) with respect to \( v \). In particular:

- \( \text{fun} \): \( v(\mathcal{E}_\psi (F \equiv \alpha \beta G)) \equiv \mathfrak{T} \), if \( v(\mathcal{E}_\psi (F A \equiv \beta \beta F A)) \equiv \mathfrak{T} \) for all \( A \in \mathfrak{W}_{\alpha, \beta} \).

\( \mathfrak{M}_\beta \) If \( M \in \mathfrak{M}_\beta \), then \( \mathcal{E}_\psi (\equiv \alpha) \) is the equality relation on \( D_\alpha \).

**Proof:**

\( \mathfrak{M}_\beta \) \( \text{ref} \): \( v(\mathcal{E}_\psi (A \equiv \alpha A)) \equiv \mathfrak{T} \), if for all \( p \in D_{\alpha, \alpha} \), we have that \( v(\mathcal{E}(p, p)(P A)) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(p, p)(P A)) \equiv \mathfrak{T} \), which is obvious since \( v \) is total and surjective.

- \( \text{sy} \): Suppose \( v(\mathcal{E}(A \equiv \alpha B)) \equiv \mathfrak{T} \), but \( v(\mathcal{E}(B \equiv \alpha A)) \equiv \mathfrak{F} \). From the latter we get that \( v(\mathcal{E}(p, p)(P B)) \equiv \mathfrak{T} \) and \( v(\mathcal{E}(p, p)(P A)) \equiv \mathfrak{F} \) for some \( p \in D_{\alpha, \alpha} \). Without loss of generality, let \( p := \mathcal{E}(V) \) for a fresh variable \( V \in \Sigma_{\alpha, \alpha} \). From the former assumption we know that for all \( q \in D_{\alpha, \alpha} \), holds \( v(\mathcal{E}(q, q)(P A)) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(q, q)(P A)) \equiv \mathfrak{T} \) and hence \( v(\mathcal{E}(q, q)(P A)) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(q, q)(P A)) \equiv \mathfrak{T} \) which is equivalent with \( v(\mathcal{E}(p, p)(P A)) \equiv \mathfrak{T} \) or \( v(\mathcal{E}(p, p)(P B)) \equiv \mathfrak{F} \) and contradicts the latter assumption.

- \( \text{tr} \): Similar to \( \text{sy} \).

\( \mathfrak{M}_\beta \) \( \text{co} \): Suppose \( v(\mathcal{E}(F \equiv \alpha \beta G)) \equiv \mathfrak{T} \), but \( v(\mathcal{E}(F A \equiv \alpha G A)) \equiv \mathfrak{F} \). From the latter we get that \( v(\mathcal{E}(p, p)(P (F A))) \equiv \mathfrak{T} \) and \( v(\mathcal{E}(p, p)(P (G A))) \equiv \mathfrak{F} \) for some \( p \in D_{\alpha, \alpha} \). Without loss of generality, let \( p := \mathcal{E}(V) \) for a fresh variable \( V \in \Sigma_{\alpha, \alpha} \). From the former assumption we know that for all \( q \in D_{\alpha, \alpha} \), holds \( v(\mathcal{E}(q, q)(P (F A))) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(q, q)(P (F A))) \equiv \mathfrak{T} \) and hence \( v(\mathcal{E}(q, q)(P (F A))) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(q, q)(P (F A))) \equiv \mathfrak{T} \), which is equivalent with \( v(\mathcal{E}(p, p)(P (F A))) \equiv \mathfrak{F} \) or \( v(\mathcal{E}(p, p)(P (G A))) \equiv \mathfrak{T} \) and contradicts the former assumption.
2.7 Primitive Equality

The situation of higher-order semantics becomes much simpler if we introduce equality as a primitive logical constant = in \( \Sigma \), which we will assume for the rest of this section. Since = is logical, we have to specialise the notion of \( \Sigma \)-valuation (cf. 2.24) by requiring that \( v(\varepsilon(\mathbf{=}^\mathbf{a})@\mathbf{b}) = \top \) iff \( \mathbf{a} \equiv \mathbf{b} \). In this case, we call \( v \) a \( \Sigma \)-valuation with equality.

Furthermore, we say that a \( \Sigma \)-model \( \mathcal{M} \) has property \( \varepsilon \), iff \( \mathcal{M} \) has property \( \mathcal{f} \) and for all types \( \alpha \), \( v(\varepsilon(\mathbf{=}^\mathbf{a})@\mathbf{b}) \equiv \top \) iff \( \mathbf{a} \equiv \mathbf{b} \).

Definition 2.39 (Henkin Model with Primitive Equality).

A (functional) \( \Sigma \)-model, which has property \( \varepsilon \) is called a (functional) \( \Sigma \)-model with primitive equality and a functional one with additional property \( \mathcal{f} \) is called a \( \Sigma \)-Henkin model with primitive equality. The class of all \( \Sigma \)-models with primitive equality are denoted by \( \mathcal{M}_{\mathcal{f}} \) and the class of all \( \Sigma \)-Henkin model with primitive equality by \( \mathcal{M}_{\mathcal{f}\mathcal{b}} \).

Remark 2.40 (Property \( \varepsilon \) implies \( \mathcal{f} \)). It is easy to see that property \( \varepsilon \) implies \( \mathcal{f} \). The only difference between both properties is, that property \( \mathcal{f} \) only ensures the existence of the intended semantical equality relation, while \( \varepsilon \) additionally requires that our new logical connectives \( =^\varepsilon \) are indeed associated with this relations. The connection between property \( \mathcal{f} \) and \( \varepsilon \) is already discussed in [And72a]. Andrews concludes that it seems natural to require the existence of logical connectives \( =^\varepsilon \) in the signature, if one is interested in extensionality. In this thesis we are especially interested to shed some light on both: in extensionality of Leibniz equality in case \( =^\varepsilon \notin \Sigma \) and in extensionality of Leibniz equality and/or primitive equality in case \( =^\varepsilon \in \Sigma \).

As property \( \varepsilon \) implies \( \mathcal{f} \) which in turn implies property \( \mathcal{f} \), it is easy to see that the landscape of higher-order semantics from Figure 1.1 at Page 6 collapses to the one in Figure 2.1.
2.8. A NOTE ON DEFINED AND PRIMITIVE EQUALITY

Definition 2.41 (Extensionality for primitive equality). Analogous to the extensionality axioms for Leibniz equality, we can define such for primitive equality.

\[
\begin{align*}
\text{EXT}^{\alpha \rightarrow \beta} & := \forall F_{\alpha \rightarrow \beta} \forall G_{\alpha \rightarrow \beta} (\forall X_{\alpha} F \equiv G X) \Rightarrow F \equiv^{\beta} G \\
\text{EXT}^\alpha & := \forall A_{\alpha} \forall B_{\alpha} (A \equiv B) \Leftrightarrow A \equiv^\alpha B
\end{align*}
\]

the axioms of full extensionality for primitive equality.

The following lemma shows that in a \( \Sigma \)-model with full equality the denotations of primitive equations and corresponding Leibniz equations are identical modulo \( v \).

Lemma 2.42 (Primitive and Leibniz equality). If \( M := (D, \alpha, \epsilon, v) \in \mathcal{M}_\beta \), we have that \( v(\mathcal{E}_v(A = B)) \equiv v(\mathcal{E}_v(A \equiv B)) \) for all \( A, B \in \mathcal{M}_\beta \).

Proof: By lemma 2.36(3) we have \( v(\mathcal{E}_v(A \equiv B)) \equiv T \), if \( \mathcal{E}_v(A) \equiv \mathcal{E}_v(B) \), since each \( M \in \mathcal{M}_\beta \) is also in \( \mathcal{M}_\beta \). By property \( \epsilon \) this is equivalent with \( v(\mathcal{E}_v(A = B)) \equiv T \).

Lemma 2.43 (Extensionality in \( \Sigma \)-models with primitive equality).

1. There exists a \( M \in \mathcal{M}_\beta \) for which \( \text{EXT}^\alpha \) and \( \text{EXT}_L^\beta \) are not valid.
2. \( \text{EXT}^{\alpha \rightarrow \beta} \) and \( \text{EXT}_L^{\alpha \rightarrow \beta} \) are valid in \( M \), if \( M \in \mathcal{M}_\beta \).
3. \( \text{EXT}^\alpha \) and \( \text{EXT}_L^\beta \) is valid in \( M \), if \( M \in \mathcal{M}_\beta \).

Thus we can extend the table in Lemma 2.37 to the following one:

<table>
<thead>
<tr>
<th>valid in</th>
<th>( \mathcal{M}_\beta )</th>
<th>( \mathcal{M}_\beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{EXT}_L^{\alpha \rightarrow \beta}, \text{EXT}^{\alpha \rightarrow \beta} )</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( \text{EXT}_L^\alpha, \text{EXT}^\alpha )</td>
<td>−</td>
<td>+</td>
</tr>
</tbody>
</table>

Proof: The assertions follow from their respective counterparts in Lemma 2.37(3) with Lemma 2.42 and the fact that each \( M \in \mathcal{M}_\beta \) is also in \( \mathcal{M}_\beta \).

Theorem 2.44. Let \( M \in \mathcal{M}_\beta \), then \( \mathcal{E}_v(\equiv^\alpha) \) and \( \mathcal{E}_v(\equiv^\alpha) \) are equivalence relations on \( D_\alpha \) with respect to \( v \) for all assignments \( \varphi \). If \( M \in \mathcal{M}_\beta \), then \( \mathcal{E}_v(\equiv^\alpha) \equiv \mathcal{E}_v(\equiv^\alpha) \) is the equality relation on \( D_\alpha \).

Proof: If \( M \in \mathcal{M}_\beta \), then the proofs for \( \equiv \) are provided by lemma 2.38. For \( \equiv \) they follow immediately with 2.42.

If \( M \in \mathcal{M}_\beta \), then the argumentation is analogous, and we first show that \( \equiv \) and \( = \) denote a functional congruence relation. As furthermore \( v \) is the identity relation on \( D_\alpha \), we have that \( \equiv \) and \( = \) indeed denote the intended semantical equality relations.

2.8 A Note on Defined and Primitive Equality

The extensional higher-order resolution approach discussed in Chapter 4 treats equality as a concept defined by Leibniz' principle. While this way of handling equality is theoretically convenient and suitable it turns out that it is quite inappropriate in practice, mainly because Leibniz equality introduces new flexible literals into the search space and thereby increases the amount of blind search with primitive substitution rule \textit{Prim}. Furthermore proofs containing Leibniz equations are rather unintuitive and hard to understand for non-experts. Illustrating examples for such unintuitive problem formulations and proofs when using Leibniz equality are \( B_4^{ext} \) and \( B_5^{ext} \) presented in Section 8.1.

\(^8\) Anyhow, the already mentioned successful case study with the LEO-system on the the examples from the MIZAR-articles Boolean and Basic Properties of sets showed that Leibniz equality can be handled at least to some extent in practice. Also the Trs-system [ABT*96, AINF90], which has proven non-trivial theorems that require substantial amount of equational reasoning (e.g., theorem THM15b in [ABT*96]), still employs Leibniz equality.
Our aim therefore is to avoid Leibniz equality and to use primitive equality instead. But in contrast to first-order logic we do not have the choice in higher-order logic to consider equality as a primitive notion only. The reason is, that infinitely many definitions of equality are always implicitly provided by our higher-order language (if we choose Standard- or Henkin semantics as the underlying semantical notion), and that there is no way to get rid of them. A definition of equality different from Leibniz equality, e.g., is discussed in Andrews’ textbook [And86] (page 155), where he defines \( \equiv^a := \lambda X_n \lambda Y_n \forall Q_{n \to m} (Q_Z = Q_X) \Rightarrow (Q_X = Q_Y) \). This definition is based on a reflexivity property, whereas Leibniz equality in some sense employs a substitutivity property. A proof for the equivalence of both definitions of equality (and the one introduced below) in calculus \( \mathcal{R} \) will be presented in Section 8.3.

Apart from all the sensible definitions of equality in higher-order logic, one can always introduce additional artificial ones by adding arbitrary tautologous sub-formulæ to a sensible definition of equality. For instance, if \( A^1 \ldots A^n \) and \( B^1 \ldots B^n \) are tautologous sentences, then we can define equality based on the Leibniz idea also by \( \equiv^a := \lambda X_n \lambda Y_n \forall P_{n \to m} \left( A^1 \wedge \ldots \wedge A^n \wedge P \right) \Rightarrow \left( P \wedge A_1 \wedge \ldots \wedge A^n \wedge P \right) \).

By employing this idea, it is easy to see, that there are infinitely many valid but different ways of defining equality. As it is undecidable, whether a formula is a tautology, we also have the unfortunate fact that it is undecidable, whether a formula is equivalent to Leibniz equality. This argument immediately leads to the following corollary:

**Corollary 2.45 (Defined equality in higher-order logic).**

*Given a higher-order sentence \( A \). It is undecidable whether \( A \) contains a sub-formula that expresses the equality between two terms.*

A consequence of this lemma is that we generally have to assume the presence of some defined equations in the input problems. We cannot generally detect all defined equations and remove them by primitive equations, as one might wish to. Consequently, even if we add a primitive equality treatment to the calculus, we still have to ensure that the calculus can handle Leibniz equality and all alternative definitions of equality as well (we want to point out, that calculus \( \mathcal{R} \) indeed can handle all forms of defined equality). This obviously contrasts with the situation in first-order logic, where one has the choice to either define equality (e.g., by axiomatising equality as a congruence relation) or to consider it as a primitive notion and to add special inference rules for equality (e.g., paramodulation rules) to the calculus.

Hence we have the requirement that we still have to take care of defined equality even if our calculus can handle primitive equality and we cannot remove the extensionality or other rules for defined equality from the calculus, as one might wish to. Of course, when adding some special primitive equality rules we can afterwards examine if some of the other rules became admissible and hence superfluous (c.f. remark 8.1).

A first approach to primitive equality treatment in higher-order logic is presented in Chapter 5, where we adapt the first-order paramodulation approach to our higher-order setting. The anticipated result of this attempt is, that just adapting the first-order paramodulation rules is not sufficient to ensure Henkin completeness. Incompleteness is caused by missing extensionality principles — this time with respect to positive primitive equations. At first glance the resulting higher-order paramodulation calculus seems to be inappropriate for some problem domains — e.g., for comparisons of sets, when sets are coded as characteristic functions, such that the extensionality properties play an important role. The reason is that the developed paramodulation approach combines term rewriting with difference reduction, whereas the difference reduction aspect comes from the the extensionality treatment already provided by calculus \( \mathcal{R} \) as well as from the new extensionality rules for positive primitive equations that are added to the calculus.

Hence, apart from the problem that well founded reduction orderings for improved paramodulation approaches are hard to develop in higher-order logic, the adapted paramodulation approach has to face a second problem: it has an intrinsic mixed term rewriting and difference reducing character which will be quite hard to control in practice.

As an alternative we therefore develop in Chapter 6 a second approach to primitive equality treatment in higher-order logic which adapts the ideas of first-order RUE-resolution [Dig79]. In contrast to the intrinsic mixed term rewriting and difference reducing character of the extensional
higher-order paramodulation approach this extensional higher-order RUE-resolution approach has
a pure difference reducing character which is probably much easier to guide in practice.

We assume for the Chapters 5 and 6 that the considered signature contains the primitive
equality symbols \(=\alpha\) for all types \(\alpha\), and that the denotation of these symbol is fixed to the intended
semantical equality relations of appropriate type. Furthermore, the primitive substitution rule is
automatically extended, such that the new logical connectives \(=\alpha\) are now imitated as well.
Chapter 3

Higher-Order Model Existence

In this section we introduce abstract consistency properties and respective model existence theorems for the different semantical notions discussed in Chapter 2.1. These theorems have the following form, where \( s \in \{ \beta, R, \bar{R}, \bar{R}_b, \bar{R}_b \} \):

**Theorem (Model Existence):** For a given abstract consistency class \( \mathfrak{A} \), and a set \( H \in \mathfrak{A} \), there is a \( \Sigma \)-model \( \mathcal{M} \) of \( H \), such that \( \mathcal{M} \in \mathfrak{M} \).

The most important tools used in the proofs of the model existence theorems are the \( \Sigma \)-Hintikka sets. These sets are maximal elements in abstract consistency classes, and allow computations that resemble those in the considered semantical structures (e.g., \( \Sigma \)-Henkin models). These allow to construct \( s \)-valuations for the term structures that turn those into \( s \)-models.

The key step in the proof of the model existence theorems is an extension lemma, which guarantees a \( \Sigma \)-Hintikka set \( \mathcal{H} \) for any set \( H \) of sentences in \( \text{FL} \). Apart from this, the proofs for the model existence theorems are standard.

3.1 Abstract Consistency

Let us now review a few technicalities that we will need for the proofs of the model existence theorems.

**Definition 3.1 (Compactness).** Let \( \mathcal{C} \) be a class of sets.

1. \( \mathcal{C} \) is called **closed under subsets**, iff for all sets \( S \) and \( T \) the following condition holds: if \( S \subseteq T \) and \( T \in \mathcal{C} \), then \( S \in \mathcal{C} \).
2. \( \mathcal{C} \) is called **compact**, iff for every set \( S \) the following condition holds: \( S \in \mathcal{C} \), iff every finite subset of \( S \) is a member of \( \mathcal{C} \).

**Lemma 3.2.** If \( \mathcal{C} \) is compact, then \( \mathcal{C} \) is closed under subsets.

**Proof:** Suppose \( S \subseteq T \) and \( T \in \mathcal{C} \). Every finite subset \( A \) of \( S \) is a finite subset of \( T \), and since \( \mathcal{C} \) is compact, we know that \( A \in \mathcal{C} \). Thus \( S \in \mathcal{C} \). \( \square \)

**Definition 3.3 (Sufficiently Pure).** Let \( \Sigma \) be a signature and \( \mathcal{T} \) be a set of \( \Sigma \)-sentences. \( \mathcal{T} \) is called **sufficiently \( \Sigma \)-pure**, iff for each type \( \alpha \) there is a set of constants \( \mathcal{P}_\alpha \subseteq \mathcal{C}_\alpha \) with equal cardinality to \( \text{wff}_\alpha (\Sigma) \), such that the elements of \( \mathcal{P} \) do not occur in \( \mathcal{T} \).

We will always presuppose that sets of sets of sentences are sufficiently \( \Sigma \)-pure in order to have enough witness constants. This can be obtained in practice by enriching the signature with spurious constants. Another way would be to use specially marked variables (which may never be instantiated) as in [Koh94b].
3.1. ABSTRACT CONSISTENCY

Definition 3.4 (Properties for Abstract Consistency Classes). Let \( \Gamma_\Sigma \) be a class of sets of \( \Sigma \)-sentences. We need the following conditions, where \( A, B \in \text{cuff}_\Sigma(\Sigma) \) and \( F, G \in \text{cuff}_\alpha \rightarrow \beta(\Sigma) \):\(^1\)

\( \nabla_c \) If \( A \) is atomic, then \( A \notin \Phi \) or \( \neg A \notin \Phi \).

\( \nabla_n \) If \( \neg \neg A \in \Phi \), then \( \Phi * A \in \Gamma_\Sigma \).

\( \nabla_\beta \) If \( A \in \Phi \) and \( B \) is the \( \beta \)-normal form of \( A \), then \( B * \Phi \in \Gamma_\Sigma \).

\( \nabla_\eta \) If \( A \in \Phi \) and \( B \) is the \( \beta \eta \)-normal form of \( A \), then \( B * \Phi \in \Gamma_\Sigma \).

\( \nabla_v \) If \( A \vee B \in \Phi \), then \( \Phi * A \in \Gamma_\Sigma \) or \( \Phi * B \in \Gamma_\Sigma \).

\( \nabla_A \) If \( A \vee B \in \Phi \), then \( \Phi \cup \{ \neg A, \neg B \} \in \Gamma_\Sigma \).

\( \nabla_f \) If \( \Pi_\alpha F \in \Phi \), then \( \Phi * (F W) \in \Gamma_\Sigma \) for each \( W \in \text{cuff}_\alpha(\Sigma) \).

\( \nabla_\beta \) If \( \neg \Pi_\alpha F \in \Phi \), then \( \Phi * (\neg F w) \in \Gamma_\Sigma \) for any constant \( w \in \Sigma_\alpha \), which does not occur in \( \Phi \).

\( \nabla_\eta \) If \( (A \equiv \beta B) \in \Phi \), then \( \Phi \cup \{ A, B \} \in \Gamma_\Sigma \) or \( \Phi \cup \{ \neg A, B \} \in \Gamma_\Sigma \).

\( \nabla_\phi \) If \( (F \equiv \alpha \rightarrow \beta G) \in \Phi \), then \( \Phi * (F \equiv \beta G w) \in \Gamma_\Sigma \) for any constant \( w \in \Sigma_\alpha \), which does not occur in \( \Phi \).

(Additional abstract consistency conditions for primitive equality will be introduced later in Section 3.3.)

Remark 3.5. Note that for the connectives \( \vee, \Pi_\alpha \) there are two conditions – a positive and a negative one – given in the definition above, namely \( \nabla_\vee / \nabla_n \) for \( \vee \) and \( \nabla_\Pi / \nabla_\beta \) for \( \Pi_\alpha \). For \( \equiv^{\alpha \rightarrow \beta} \) the situation is different, as we need only conditions for the negative cases. The positive cases can be inferred at the level of Hintikka sets by expanding the Leibniz definition of equality (see the proofs of \( \nabla_\phi \) in lemma 3.15 and \( \nabla_\phi \) in lemma 3.17).

Definition 3.6 (Abstract Consistency Classes). Let \( \Sigma \) be a signature and \( \Gamma_\Sigma \) be a class of sets of \( \Sigma \)-sentences. Using the properties from the previous definition we introduce the following abstract consistency classes:

\( \text{Acc}_{\beta} \) If \( \nabla_c, \nabla_n, \nabla_\beta, \nabla_\Pi, \nabla_\alpha, \nabla_\eta, \nabla_v \) and \( \nabla_\phi \) are valid for \( \Gamma_\Sigma \), then \( \Gamma_\Sigma \) is called an abstract consistency class for \( \Sigma \)-models (\( \text{Acc}_\Sigma \)).

Based upon this definition we introduce the following specialised abstract consistency classes:

\( \text{Acc}_{\beta\beta}, \text{Acc}_{\beta\eta}, \text{Acc}_{\beta\Pi}, \text{Acc}_{\beta\alpha}, \text{Acc}_{\beta\eta\Pi}, \text{Acc}_{\beta\eta\alpha} \), where we indicate by indices which additional properties from \( \{ \nabla_\Pi, \nabla_\alpha, \nabla_\eta \} \) are required.

Sometimes we do not want to differentiate between the particular notions above. In this cases we simply speak of an abstract consistency class, with which we refer to an arbitrary but one in \( \{ \text{Acc}_{\beta}, \text{Acc}_{\beta\beta}, \text{Acc}_{\beta\eta}, \text{Acc}_{\beta\Pi}, \text{Acc}_{\beta\alpha}, \text{Acc}_{\beta\eta\Pi}, \text{Acc}_{\beta\eta\alpha} \} \).

Remark 3.7. Note that \( \text{Acc}_{\beta} \) corresponds to the abstract consistency property discussed by Andrews in [And71]. The only (technical) difference is that Andrews does not consider \( \alpha \)-conversion as built-into the logic but needs a condition similar to \( \nabla_\phi \) that requires \( \alpha \)-standardised forms to be abstract consistent.

Lemma 3.8 (Non-atomic Consistency). Let \( \Gamma_\Sigma \) be an abstract consistency class and \( A \in \text{cuff}_\Sigma(\Sigma) \), then for all \( \Phi \in \Gamma_\Sigma \) we have \( A \notin \Phi \) or \( \neg A \notin \Phi \).

\(^1\)In the following we will use \( \varphi * A \) as an abbreviation for \( \varphi \cup \{ A \} \).
Proof: Let \( A \in \text{wff}(\Sigma) \) and \( \Phi \in \Gamma_2 \), such that \( A \in \Phi \). By \( \nabla \beta \) we can prove that \( A \) is a \( \beta \)-normal form. So we prove the assertion by an induction over the structure of \( A \).

If \( A \) is atomic we get the assertion immediately by \( \nabla \alpha \). If \( A \) is not atomic, then its head must be a logical constant, therefore we can proceed by a case-analysis over the connectives and quantifiers.

Suppose \( A \) has the form \( \neg B \) and \( \{\neg B, \neg\neg B\} \subseteq \Phi \). By \( \nabla \alpha \) we know that \( \{\neg B, B\} \cup \Phi \in \Gamma_2 \), which contradicts the induction hypotheses. Now suppose \( A \) has the form \( B \lor C \) and \( \{B \lor C, \neg(B \lor C)\} \subseteq \Phi \). By \( \nabla \gamma \), \( \nabla \alpha \) we know that \( \{B \lor C, \neg(B \lor C), B, \neg C\} \cup \Phi \in \Gamma_2 \) or \( \{B \lor C, \neg(B \lor C), C, \neg B, \neg C\} \cup \Phi \in \Gamma_2 \). In both cases the contradiction is given by the induction hypotheses.

Suppose \( A \) has the form \( \Pi(\lambda X, B) \) and \( \{\Pi(\lambda X, B), \neg\Pi(\lambda X, B)\} \subseteq \Phi \). By \( \nabla \beta \), \( \nabla \gamma \) and \( \nabla \beta \) we know that \( \{\Pi(\lambda Y, B), \neg\Pi(\lambda Y, B), [W/Y]B, \neg[W/Y]B\} \cup \Phi \in \Gamma_2 \), which contradicts again the induction hypotheses.

In contrast to [And71], we work with saturated abstract consistency classes in order to obtain total \( \Sigma \)-valuations, which makes the proofs of the model existence theorem much simpler and, e.g., yields much more natural models.

Definition 3.9 (Saturated). We call an abstract consistency class \( \Gamma_2 \) atomically saturated, iff for all \( \Phi \in \Gamma_2 \) and for all atomic sentences \( A \in \text{cuff}_n(\Sigma) \), we have \( \Phi \ast A \in \Gamma_2 \) or \( \Phi \ast \neg A \in \Gamma_2 \). If this property holds for all sentences \( A \in \text{cuff}_n(\Sigma) \), then we call \( \Gamma_2 \) saturated.

Remark 3.10. Clearly, not all abstract consistency classes are saturated, since the empty set is one that is not, even if \( \Sigma \) is empty.

In the definition of abstract consistency class, we only had to require atomic consistency, i.e., that there are no atomic propositions that contradict each other in one abstract consistent set, to ensure consistency (see 3.8). The conjecture is that a similar theorem can be proven for saturatedness:

Conjecture: Let \( \Gamma_2 \) be an atomic saturated abstract consistency class. Then there exists a saturated abstract consistency class \( \Gamma_2' \), with \( \Gamma_2 \subseteq \Gamma_2' \), which is a subclass of \( \Gamma_2 \).

Such a result would be of practical importance, as it allows to reduce the problem of proving saturatedness of a given calculus to proving atomic saturatedness.

Lemma 3.11. Let \( \Gamma_2 \) be a saturated abstract consistency class, \( \Phi \in \Gamma_2 \) and \( A \) an atomic sentence. Then \( \Phi \ast (A \lor \neg A) \in \Gamma_2 \).

Proof: Since \( \Gamma_2 \) is saturated and \( \Phi \in \Gamma_2 \), we must have \( \Phi \ast (A \lor \neg A) \in \Gamma_2 \) or \( \Phi \ast \neg(A \lor \neg A) \in \Gamma_2 \). We prove the assertion by refuting the second alternative. If \( \Phi \ast \neg(A \lor \neg A) \in \Gamma_2 \), then \( \Phi \cup \{\neg(A \lor \neg A), \neg A, \neg\neg A, A\} \in \Gamma_2 \) by \( \nabla \alpha \) and \( \nabla \beta \). Since \( A \) is an atomic sentence we get a contradiction with lemma 3.8.

Lemma 3.12 (Compactness of abstract consistency classes). For each abstract consistency class \( \Gamma_2 \), there exists an abstract consistency class \( \Gamma_2' \) of the same type, such that \( \Gamma_2 \subseteq \Gamma_2' \), and \( \Gamma_2' \) is compact. Furthermore \( \Gamma_2 \) is saturated, iff \( \Gamma_2' \) is.

Proof: (following and extending [And86], proposition no. 2506)
We choose \( \Gamma_2' := \{\Phi \subseteq \text{cuff}_n(\Sigma) \mid \text{every finite subset of } \Phi \text{ is in } \Gamma_2 \} \). Now suppose that \( \Phi \in \Gamma_2 \). \( \Gamma_2 \) is closed under subsets, so every finite subset of \( \Phi \) is in \( \Gamma_2 \) and thus \( \Phi \in \Gamma_2' \). Hence \( \Gamma_2 \subseteq \Gamma_2' \).

Next let us show that each \( \Gamma_2' \) is compact. Suppose \( \Phi \in \Gamma_2' \) and \( \Psi \) is an arbitrary finite subset of \( \Phi \). By definition of \( \Gamma_2' \) all finite subsets of \( \Phi \) are in \( \Gamma_2 \) and therefore \( \Psi \in \Gamma_2 \). Thus all finite subsets of \( \Phi \) are in \( \Gamma_2' \) whenever \( \Phi \) is in \( \Gamma_2' \). On the other hand, suppose all finite subsets of \( \Phi \) are in \( \Gamma_2' \). Then by the definition of \( \Gamma_2' \) the finite subsets of \( \Phi \) are also in \( \Gamma_2 \), so \( \Phi \in \Gamma_2 \). Thus \( \Gamma_2 \) is compact.

Next we show that if \( \Gamma_2 \) satisfies \( \nabla_\ast \), then \( \Gamma_2' \) satisfies \( \nabla_\ast \), by considering the cases of definition 3.6. First note that by lemma 3.2 we have that \( \Gamma_2' \) is closed under subsets.
3.2. Hintikka Sets

Let $\Phi \in \Gamma_2'$ and suppose there is an atom $A$, such that $\{A, \neg A\} \subseteq \Phi$. Then $\{A, \neg A\} \in \Gamma_2$ contradicting $\nabla_c$.

Let $\Phi \in \Gamma_2'$, $\neg A \in \Phi$, $\Psi$ be any finite subset of $\Phi \ast A$ and $\Theta := (\Psi \setminus \{A\}) \ast \neg A$. $\Theta$ is a finite subset of $\Phi$, so $\Theta \in \Gamma_2$. Since $\Gamma_2$ is an abstract consistency class and $\neg A \in \Theta$, we get $\Theta \ast A \in \Gamma_2$ by $\nabla_a$. We know that $\Psi \subseteq \Theta \ast A$ and $\Gamma_2$ is closed under subsets, so $\Psi \in \Gamma_2$. Thus every finite subset $\Psi$ of $\Phi \ast A$ is in $\Gamma_2$ and therefore by definition $\Phi \ast A \in \Gamma_2'$.

$\nabla_b, \nabla_c, \nabla_d, \nabla_e, \nabla_f$ Analogous to $\nabla_a$.

Let $\Phi \in \Gamma_2'$ with $\neg A \ast B \in \Phi$ but $\Phi \cup \{A, \neg B\} \notin \Phi$ and $\Phi \cup \{\neg A, B\} \notin \Phi$. Then there exists finite subsets $\Phi_1$ and $\Phi_2$ of $\Phi$, such that $\Phi_1 \ast \{A, \neg B\} \in \Gamma_2'$ and $\Phi_2 \ast \{\neg A, B\} \notin \Gamma_2'$.

Now we choose $\Theta := \Phi_1 \cup \Phi_2 \ast (A \equiv B)$. Obviously $\Theta$ is a finite subset of $\Phi$ and therefore $\Theta \in \Gamma_2$. Since $\Gamma_2$ satisfies $\nabla_b$, we have that $\Theta \cup \{A, \neg B\} \in \Gamma_2'$ or $\Theta \cup \{\neg A, B\} \in \Gamma_2'$. From this and the fact that extensional abstract consistency classes are closed under subsets we get that $\Phi_1 \cup \{A, \neg B\} \in \Gamma_2'$ or $\Phi_2 \cup \{\neg A, B\} \in \Gamma_2'$, which contradicts our assumption.

For the proof that $\Gamma_2'$ is saturated, let $\Phi \in \Gamma_2'$, but neither $\Phi \ast A$ nor $\Phi \ast \neg A$ be in $\Gamma_2'$. Then there are finite subsets $\Phi^+ \ast \Phi^-$ of $\Phi$, such that $\Phi^+ \ast A \notin \Gamma_2'$ and $\Phi^- \ast \neg A \notin \Gamma_2'$ (since all finite subsets of $\Phi$ are in $\Gamma_2'$). As $\Psi := \Phi^+ \cup \Phi^-$ is a finite subset of $\Phi$, we have $\Psi \in \Gamma_2'$. Furthermore, $\Psi \ast A \in \Gamma_2'$ or $\Psi \ast \neg A \in \Gamma_2'$, because $\Gamma_2$ is closed under subsets, so $\Phi^+ \ast A \in \Gamma_2'$ or $\Phi^- \ast \neg A \in \Gamma_2'$. This is a contradiction, so we can conclude that if $\Phi \in \Gamma_2$, then $\Phi \ast A \in \Gamma_2'$ or $\Phi \ast \neg A \in \Gamma_2'$.

3.2 Hintikka Sets

Now we define Hintikka sets, which are maximal elements in an abstract consistency class. Hintikka sets connect syntax with semantics as they provide the basis for the model constructions in the model existence theorem 3.29.

**Definition 3.13 (\(\Sigma\)-Hintikka Set).** Let $\Gamma_2$ be an abstract consistency class, then a set $\mathcal{H}$ is called a $\Sigma$-Hintikka set for $\Gamma_2$, if it is maximal in $\Gamma_2$, i.e., iff for each sentence $D \in \text{cuff}_\Sigma(\mathcal{H})$, such that $\mathcal{H} \ast D \in \Gamma_2$, we already have $D \in \mathcal{H}$.

In the following we discuss properties of $\Sigma$-Hintikka sets. Since we have different types of abstract consistency classes, depending on the additional requirements $f, g$ and $h$, we have to discuss different Hintikka lemmata.

**Theorem 3.14 (Hintikka Lemma for \(\mathfrak{A}\mathfrak{C}\mathfrak{C}_\Sigma\)).** If $\Gamma_2$ is a saturated $\mathfrak{A}\mathfrak{C}\mathfrak{C}_\Sigma$ and $\mathcal{H}$ is maximal in $\Gamma_2$, then the following statements hold for all $A, B \in \text{cuff}_\Sigma(\mathcal{H})$, $F \in \text{cuff}_{\alpha \rightarrow_\Sigma}(\mathcal{H})$ and $C, D, E \in \text{cuff}_\Sigma(\mathcal{H})$:

- $\nabla_c^1 \ A \notin \mathcal{H}$ or $\neg A \notin \mathcal{H}$.
- $\nabla_c^2 \ A \in \mathcal{H}$, iff $\neg A \notin \mathcal{H}$.
- $\nabla_c^3 \ \neg A \in \mathcal{H}$, iff $A \notin \mathcal{H}$.
- $\nabla_c^4 \ (\neg \neg A) \in \mathcal{H}$, iff $A \in \mathcal{H}$.
CHAPTER 3. HIGHER-ORDER MODEL EXISTENCE

\( \forall \beta \)
If \( A \equiv_\beta B \), then \( A \in \mathcal{H} \), iff \( B \in \mathcal{H} \).

\( \forall \alpha \)
(\( A \lor B \)) \in \mathcal{H} \, iff \, A \in \mathcal{H} \ or \ B \in \mathcal{H}.

\( \forall \alpha \)
\( \neg (A \lor B) \in \mathcal{H} \, iff \, \neg A \in \mathcal{H} \ and \ \neg B \in \mathcal{H}.

\( \forall \) 
\( \Pi^\alpha F \in \mathcal{H} \, iff \, for \ each \ D \in \text{cuff}_\alpha(\Sigma) \ we \ have \ (F[D]) \in \mathcal{H}.

\( \forall \alpha \)
\( \neg \Pi^\alpha F \in \mathcal{H} \, iff \, there \ is \ a \ D \in \text{cuff}_\alpha(\Sigma), \ such \ that \ \neg(F[D]) \in \mathcal{H}.

\( \forall \alpha \)
\( A \vdash^\alpha A \in \mathcal{H} \)

\( \forall \alpha \)
If \( F[C]_p \in \mathcal{H} \ and \ C \models^\alpha D \in \mathcal{H}, \ then \ F[D]_p \in \mathcal{H}

\( \forall \alpha \)
\( C \models^\alpha D \in \mathcal{H}, \ iff \ D \models^\alpha C \in \mathcal{H}.

\( \forall \alpha \)
\( C \models^\alpha D \in \mathcal{H} \ and \ D \models^\alpha E \in \mathcal{H}, \ then \ C \models^\alpha E \in \mathcal{H}.

\( \forall \alpha \)
(\( A \lor \neg A \)) \in \mathcal{H} \ for \ any \ sentence \ A.

Proof:
\( \forall \alpha \)
By 3.8.

\( \forall \alpha \)
Both are direct consequences of the saturation of \( \Gamma_2 \) and \( \forall \).

\( \forall \alpha \)
If \( \neg A \in \mathcal{H} \), then \( H \vdash A \in \Gamma_2 \) by \( \forall \). The maximality of \( \mathcal{H} \) now gives us that \( A \in \mathcal{H} \). To obtain the converse, let us assume that \( A \in \mathcal{H} \). Then by \( \forall \) we know that \( \neg A \notin \mathcal{H} \) and by \( \forall \) we have \( \neg A \notin \mathcal{H} \).

\( \forall \beta \)
Suppose \( A \equiv_\beta B \). Since \( \beta \)-reduction is terminating and confluent there is unique \( C \), such that \( C \) is the \( \beta \)-normal-form of \( A \) and \( B \). Without loss of generality we show that if \( A \in \mathcal{H} \), then \( B \in \mathcal{H} \). For that we suppose that \( A \in \mathcal{H} \ but \ B \notin \mathcal{H} \). From the latter we get by \( \forall \) that \( \neg B \in \mathcal{H} \). Note that the \( \beta \)-normal-form of \( A \) is \( C \) and of \( \neg B \) is \( \neg C \). By \( \forall \) and the maximality of \( \mathcal{H} \) we know that \( \{C, \neg C\} \in \mathcal{H} \), which contradicts \( \forall \).

\( \forall \alpha \)
We get the first direction by \( \forall \) and the maximality of \( \mathcal{H} \). For the converse direction let us assume that \( A \in \mathcal{H} \) or \( B \in \mathcal{H} \) but \( (A \lor B) \notin \mathcal{H} \). Then by \( \forall \) we get \( \neg (A \lor B) \in \mathcal{H} \) and by the first direction of \( \forall \) we have \( \{A, \neg B\} \subseteq \mathcal{H} \) which contradicts the assumption with \( \forall \).

\( \forall \alpha \)
Analogous to the case; note that the argumentation is not circular. In both cases we use the forward direction of the counterpart to verify the backward direction, whereas forward directions are proven directly. The same holds for the proofs of \( \forall \) and \( \forall \) below.

\( \forall \alpha \)
Again, we get the first direction by \( \forall \) and the maximality of \( \mathcal{H} \). For the converse direction let us assume that \( (F[D]) \in \mathcal{H} \) for each \( D \in \text{cuff}_\alpha(\Sigma) \), but \( \Pi^\alpha F \notin \mathcal{H} \). Then \( \neg \Pi^\alpha F \in \mathcal{H} \) by \( \forall \) \( \forall \) and by the first direction of \( \forall \) there is \( D \in \text{cuff}_\alpha(\Sigma) \), such that \( \neg (F[D]) \in \mathcal{H} \) which is a contradiction.

\( \forall \alpha \)
Analogous to \( \forall \).

\( \forall \alpha \)
Suppose \( A \vdash^\alpha A \notin \mathcal{H} \). By \( \forall \), the definition of \( \vdash^\alpha \), \( \forall \), and \( \forall \) we have \( \neg (A \lor Q \lor A) \in \mathcal{H} \) for a \( Q \in \text{cuff}_\beta \to \alpha(\Sigma) \). Applying \( \forall \) contradicts \( \forall \).

\( \forall \alpha \)
Suppose \( F[C]_p \in \mathcal{H} \) and \( C \models D \in \mathcal{H} \). From the latter we obtain \( (\lambda \alpha \top P \lor P \top D)(X, F[X]_p) \in \mathcal{H} \) by the definition of \( \models \) and \( \forall \). Note that \( X \) is free for \( F[Y]_p \) so we have \( \neg(F[C]_p \top F[D]_p) \in \mathcal{H} \) by \( \forall \). From this we conclude with \( \forall \) that \( \neg F[C]_p \in \mathcal{H} \) or \( F[D]_p \in \mathcal{H} \). Since the first option contradicts our assumption with \( \forall \), it must be the case that \( F[D]_p \in \mathcal{H} \).
3.2. HINTIKKA SETS

\[ \neg \neg \alpha \] By \( \neg \neg \) and \( \neg \).

\[ \neg \neg \beta \] By \( \neg \) and \( \neg \).

\( \mathcal{H} \) Saturation of \( \Gamma_\Sigma \) and maximality of \( \mathcal{H} \) entails that \( A \in \mathcal{H} \) or \( \neg A \in \mathcal{H} \). We now get the assertion by \( \neg \).

Depending on the kind of abstract consistency class we are considering, Hintikka sets have different properties. We discuss this different properties in the Hintikka lemmata below.

Theorem 3.15 (Hintikka Lemma for \( \mathcal{A}_{cc} \)). If \( \Gamma_\Sigma \) is a saturated \( \mathcal{A}_{cc} \) and \( \mathcal{H} \) is maximal in \( \Gamma_\Sigma \), then for all \( A, B, C \in \text{cwff}_\alpha(\Sigma) \):

\[ \mathcal{H} \text{ if } A \equiv \beta B, \text{ then } A \in \mathcal{H} \text{ iff } B \in \mathcal{H}. \]

Proof: Analogous to \( \neg \beta \) in Theorem 3.14

Theorem 3.16 (Hintikka Lemma for \( \mathcal{A}_{cc} \)). If \( \Gamma_\Sigma \) is a saturated \( \mathcal{A}_{cc} \) and \( \mathcal{H} \) is maximal in \( \Gamma_\Sigma \), then for all \( C \in \text{cwff}_\alpha(\Sigma) \), and \( F, G \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma) \):

\[ \neg (F \vdash_{\alpha \rightarrow \beta} G) \in \mathcal{H}, \text{ iff there is a } C \in \text{cwff}_\alpha(\Sigma), \text{ such that } \neg (F \vdash_{\alpha \rightarrow \beta} G \ C) \in \mathcal{H}. \]

\[ F \vdash_{\alpha \rightarrow \beta} G \in \mathcal{H}, \text{ iff } F \ C \vdash_{\alpha \rightarrow \beta} G \ C \in \mathcal{H} \text{ for all } C \in \text{cwff}_\alpha(\Sigma) \]

Proof:

We get the first direction by the definition of \( \equiv \), \( \vdash \), and the maximality of \( \mathcal{H} \). For the converse let us suppose that \( \neg (F \vdash_{\alpha \rightarrow \beta} G \ C) \in \mathcal{H} \) but \( \neg (F \vdash G) \notin \mathcal{H} \). From the latter we know by \( \vdash \), that \( F \vdash G \in \mathcal{H} \) and by \( \neg \vdash \) we have that \( \neg (G \ C \vdash_{\alpha \rightarrow \beta} G \ C) \in \mathcal{H} \) which contradicts \( \vdash \) and \( \vdash \).

Suppose \( F \vdash G \in \mathcal{H} \) but \( F \ C \vdash_{\alpha \rightarrow \beta} G \ C \notin \mathcal{H} \), which means by \( \vdash \), that \( \neg (F \vdash_{\alpha \rightarrow \beta} G \ C) \in \mathcal{H} \). From this we get by the definition of \( \vdash \), \( \neg \vdash \) and \( \vdash \), that \( \neg (\neg Q (F \ C) \lor Q (G \ C)) \in \mathcal{H} \) for some \( Q \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma) \). On the other hand we know from \( F \vdash G \in \mathcal{H} \) by the definition of \( \vdash \) and \( \vdash \) that \( (\lambda X_{\alpha \rightarrow \beta}(\Sigma) \rightarrow P \lor P \lor G) (\lambda X_{\alpha \rightarrow \beta} \in \mathcal{H}) \), and hence by \( \vdash \) that \( \neg (\lambda X_{\alpha \rightarrow \beta}(\Sigma) \rightarrow P \lor P \lor G) (\lambda X_{\alpha \rightarrow \beta} \in \mathcal{H}) \), which contradicts \( \vdash \). For the converse assume that \( F \ C \vdash_{\alpha \rightarrow \beta} G \ C \in \mathcal{H} \) for all \( C \in \mathcal{H} \) but \( F \vdash G \notin \mathcal{H} \). We get by \( \vdash \) that \( \neg (F \vdash G) \in \mathcal{H} \) which contradicts the assumption with \( \vdash \) and \( \vdash \).

Theorem 3.17 (Hintikka Lemma for \( \mathcal{A}_{cc} \)). If \( \Gamma_\Sigma \) is a saturated \( \mathcal{A}_{cc} \) and \( \mathcal{H} \) is maximal in \( \Gamma_\Sigma \), then for all \( A, B \in \text{cwff}_\alpha(\Sigma) \):

\[ \neg (A \vdash_{\alpha} B) \in \mathcal{H}, \text{ iff } \{ \neg A, B \} \subseteq \mathcal{H} \text{ or } \{ A, \neg B \} \subseteq \mathcal{H}. \]

\[ (A \vdash_{\alpha} B) \in \mathcal{H}, \text{ iff } \{ A, B \} \subseteq \mathcal{H} \text{ or } \{ \neg A, \neg B \} \subseteq \mathcal{H}. \]

\[ (A \equiv_{\alpha} B) \in \mathcal{H}, \text{ iff } (A \vdash_{\alpha} B) \in \mathcal{H}. \]

\[ \text{Either } A \vdash_{\alpha} B \in \mathcal{H} \text{ or } A \equiv_{\alpha} \neg B \in \mathcal{H}. \]

\[ \neg (T_a \vdash_{\alpha} F_o) \in \mathcal{H}, \text{ if } T_a \text{ and } F_o \text{ are defined as in lemma 2.25}. \]

\[ \text{Either } A \vdash_{\alpha} T_a \in \mathcal{H} \text{ or } A \equiv_{\alpha} F_o \in \mathcal{H}. \]

Proof:
We get the first direction by the definition of $\equiv$, $\nabla_b^1$ and the maximality of $\mathcal{H}$. Now assume that $\{\neg A, B\} \subseteq \mathcal{H}$ or $\{A, \neg B\} \subseteq \mathcal{H}$ but $\neg (A \equiv B) \notin \mathcal{H}$. From the latter we know by the definition of $\equiv$ and $\nabla_a^1$ that $\{(\lambda x \cdot P)(\lambda x \cdot P)B(\lambda X \cdot \neg X)(\lambda x \cdot \neg P)A \vee P B)(\lambda X \cdot \neg X)\} \subseteq \mathcal{H}$ and by $\nabla_b^1$ and $\nabla_a$ that one of $\{\neg A, \neg \neg A\}$, $\{B, \neg B\}$, $\{\neg A, \neg B\}$ or $\{B, \neg \neg A\}$ must be a subset of $\mathcal{H}$. All four cases contradict $\nabla_c^3$ together with the assumption.

Since $\mathcal{E}_2$ is saturated we have $A \in \mathcal{H}$ or $\neg A \in \mathcal{H}$. From this we easily get the first direction by $\nabla_a^3$. For the converse suppose that $\{A, B\} \in \mathcal{H}$ or $\{\neg A, \neg B\} \in \mathcal{H}$ but $(A \equiv B) \notin \mathcal{H}$ which means by $\nabla_a^3$ that $\neg (A \equiv B) \in \mathcal{H}$. By $\nabla_b^1$ we have $\{A, B\} \in \mathcal{H}$ or $\{A, \neg B\} \in \mathcal{H}$.

In each of the four cases the contradiction follows by $\nabla_c^3$.

If we assume $(A \iff B) \in \mathcal{H}$, then by the definition of $\iff$ and $\nabla_a$ we have $\{\neg A \vee B, A \vee \neg B\} \subseteq \mathcal{H}$, and by $\nabla_a$ that $\{\neg A, A\} \subseteq \mathcal{H}$ or $\{A, \neg B\} \subseteq \mathcal{H}$ or $\{\neg A, \neg B\} \subseteq \mathcal{H}$ or $\{A, B\} \subseteq \mathcal{H}$.

Note that the first two alternatives are impossible because of $\nabla_c^3$. Now we assume that $A \equiv B \notin \mathcal{H}$ from which we obtain by $\nabla_c^3$ and $\nabla_b$ that $\{\neg A, B\} \subseteq \mathcal{H}$ or $\{A, \neg B\} \subseteq \mathcal{H}$. We have to consider four cases and in each case we get a contradiction with $\nabla_c^3$.

Assume that $A \equiv B \notin \mathcal{H}$ and $A \equiv \neg B \notin \mathcal{H}$. By $\nabla_a$ we have $\neg (A \equiv B) \in \mathcal{H}$ and $\neg (A \equiv \neg B) \in \mathcal{H}$, and by $\nabla_b^1$ we get from the former that $\{\neg A, B\} \subseteq \mathcal{H}$ or $\{A, \neg B\} \subseteq \mathcal{H}$ and from the latter that $\{A, \neg B\} \subseteq \mathcal{H}$ or $\{\neg A, \neg B\} \subseteq \mathcal{H}$. We have to consider four cases and in each we get a contradiction with $\nabla_c^3$.

Analogously we can show with $\nabla_b^1$ that $A \equiv B \in \mathcal{H}$ and $A \equiv \neg B \in \mathcal{H}$ leads to a contradiction.

From $\nabla_a$ we know that $T_o \in \mathcal{H}$. Hence by $\nabla_a^2$ and $\nabla_a^3$ that $\neg F_o \in \mathcal{H}$ and finally by $\nabla_b$ we get $\neg (T_o =^o F_o) \in \mathcal{H}$.

Follows immediately from $\nabla_a$.

\[ \square \]

### 3.3 Primitive Equality

We now define abstract consistency properties for primitive equality. For this we have different options, e.g., we could introduce primitive equality by postulating $=$ to be a functional congruence relation or alternatively we could state properties connecting $=$ with $\equiv$.

Our concrete choice, namely a property postulating reflexivity and substitutivity of $=$, is motivated from a practical point of view, as we believe that reflexivity and substitutivity are more easy to verify in practical applications.

**Definition 3.18 (Abstract Consistency with Primitive Equality).** Let $\Sigma$ be signature and let $\mathcal{E}_2$ be a $\mathcal{A}_\beta$, then we define the following condition, where $\Phi \in \mathcal{E}_2$:

$\nabla_b^1$ $\neg (A =^0 A) \notin \Phi.$

$\nabla_b^2$ If $F[A]_p \in \Phi$ and $A = B \in \Phi$, then $\Phi \ast F[B]_p \in \mathcal{E}_2$.

Using this properties we introduce the abstract consistency classes $\mathcal{A}_{\beta}$ and $\mathcal{A}_{\beta \beta}$ based upon the definition of $\mathcal{A}_\beta$.

Instead of using reflexivity and substitution principles we can also introduce the following alternative definition for abstract consistency with primitive equality:

**Definition 3.19 (Alternative Abstract Consistency with Primitive Equality).** Let $\Sigma$ be signature and let $\mathcal{E}_2$ be a $\mathcal{A}_\beta$, then we define the following condition, where $\Phi \in \mathcal{E}_2$:

$\nabla_b^1$ $\neg (A =^0 A) \notin \Phi.$

$\nabla_b^2$ If $F[A]_p \in \Phi$ and $A = B \in \Phi$, then $\Phi \ast F[B]_p \in \mathcal{E}_2$.
3.3. PRIMITIVE EQUALITY

\[ \nabla \neg \alpha \] If \( \neg(A = B) \in \Phi \), then \( \Phi \ast \neg(A \neq B) \in \Gamma \).

\[ \nabla = \alpha \] If \( \neg(A = B) \in \Phi \), then \( \Phi \ast \neg(A = B) \in \Gamma \).

Using these properties instead of \( \nabla \neg \alpha \) and \( \nabla = \alpha \) we can introduce analogs to the abstract consistency classes \( \mathfrak{C} \mathfrak{C} \mathfrak{C}_\beta \) and \( \mathfrak{C} \mathfrak{C} \mathfrak{C}_\beta \) above.

Remark 3.20. Just as in the case with Leibniz equality, we can extend a abstract consistency class with primitive equality so that it is compact.

**Proof:** We proceed just as in the proof of Lemma 3.12 but check the cases for \( \nabla \neg \alpha \) and \( \nabla = \alpha \).

For \( \nabla \neg \alpha \) let \( \Phi \in \Gamma \), and suppose there is an \( A \in c \) with \( \neg(A = A) \in \Phi \). Then \( \{ \neg(A = A) \} \in \Gamma \), contradicting \( \nabla \neg \alpha \).

For \( \nabla = \alpha \) let \( \Phi \in \Gamma \), \( \{ \Phi[B] \} \subseteq \Phi \), \( \Psi \) be any finite subset of \( \Phi \ast \Phi[B] \) and \( \Theta := (\Psi \setminus \{ \Phi[B] \}) \cup \{ \Phi[A] \} \). \( \Theta \) is a finite subset of \( \Phi \), so \( \Theta \in \Gamma \). Since \( \Gamma \) is an \( \mathfrak{C} \mathfrak{C} \mathfrak{C}_\beta \) and \( \{ \Phi[B] \} \subseteq \Theta \), we get \( \Theta \ast \Phi[B] \in \Gamma \) by \( \nabla = \alpha \). We know that \( \Psi \subseteq \Theta \ast \Phi[B] \) and \( \Gamma \) is closed under subsets, so \( \Psi \in \Gamma \). Thus every finite subset \( \Psi \) of \( \Phi \ast \Phi[B] \) is in \( \Gamma \) and therefore by definition \( \Phi \ast \Phi[B] \in \Gamma \).

The next lemma shows the connection between Leibniz equality and primitive equality for \( \mathfrak{C} \mathfrak{C} \mathfrak{C}_\beta \).

Lemma 3.21 (Leibniz versus Primitive Equality). Let \( \Gamma \) be a saturated \( \mathfrak{C} \mathfrak{C} \mathfrak{C}_\beta \). For all \( \Phi \in \Gamma \), all \( A, B \in \Psi \), and \( F, G \in \Psi \), the following holds:

1. If \( \neg(A = B) \in \Phi \), then \( \Phi \ast \neg(A = B) \in \Gamma \).
2. If \( A = B \in \Phi \), then \( \Phi \ast \neg(A = B) \in \Gamma \).
3. If \( A = B \in \Phi \), then \( \Phi \ast A = B \in \Gamma \).
4. If \( A \neq B \in \Phi \), then \( \Phi \ast A \neq B \in \Gamma \).
5. If \( \neg(F = B \Rightarrow G) \in \Phi \), then \( \Phi \ast \neg(F \Rightarrow G) \in \Gamma \), for any constant \( w \in \mathcal{C}_\alpha \), which does not occur in \( \Phi \).

**Proof:**

1. Suppose \( \neg(A \neq \alpha) \in \Phi \), but \( \Phi \ast \neg(A \neq \alpha) \notin \Gamma \). Since \( \Gamma \) is saturated, we have \( \Phi \ast A \neq \alpha B \notin \Gamma \), and by \( \nabla \neg \alpha \), that \( \Phi \ast A = \alpha B \Rightarrow (B \neq \alpha) \in \Gamma \). From the definition of \( \neq \), \( A \neq \alpha B \Rightarrow (B \neq \alpha) \) is in \( \nabla \neg \alpha \) for any constant \( p \in \mathcal{C}_\alpha \). From this we get the contradiction with \( \nabla \neg \alpha \) and Lemma 3.8.

2. Suppose \( \neg(A = \alpha) \in \Phi \), but \( \Phi \ast \neg(A = \alpha) \notin \Gamma \). Since \( \Gamma \) is saturated, we have \( \Phi \ast A = \alpha B \in \Gamma \), and by definition of \( = \), \( \nabla = \alpha \) and the subset closure of \( \Gamma \), that \( \Phi \ast \lambda P \rightarrow \neg P \leftarrow A \in \Gamma \). By \( \nabla = \alpha \) and the subset closure of \( \Gamma \), we finally get that \( \Phi \ast \neg(A = \alpha) \in \Gamma \) or \( \Phi \ast A = \alpha B \in \Gamma \). The former is contradictory with \( \nabla = \alpha \) and Lemma 3.8, and the latter with the assumption \( \neg(A = \alpha) \in \Phi \) and Lemma 3.8.

3. Suppose \( A \neq \alpha B \in \Phi \), but \( \Phi \ast A \neq \alpha B \notin \Gamma \). Since \( \Gamma \) is saturated, we have \( \Phi \ast \neg(A = \alpha) \neq \alpha B \in \Gamma \), and by (2) and the subset closure of \( \Gamma \), that \( \Phi \ast \neg(A \neq \alpha B) \in \Gamma \), which contradicts the assumption with Lemma 3.8.

4. Analogous to (3) with (1).

5. From \( \neg(F \Rightarrow G) \in \Phi \), we can derive with (2), (1) and the subset closure of \( \Gamma \), that 

\[ \Phi \ast \neg(F \Rightarrow G) \in \Gamma \].

\( \square \)
Remark 3.22. Lemma 3.21 again shows that in an \( \mathcal{M} \in \mathfrak{M}_{\beta} \) the symbol \( = \) defines the same relation as \( \equiv \), namely a functional congruence relation modulo \( v \). And if we are considering an \( \mathcal{M} \in \mathfrak{M}_{\beta} \) then both describe the equality relation. This shows that the conditions \( \nabla^e_i \) and \( \nabla^e_g \) are sufficient for this purpose. We could alternatively introduce primitive equality by requiring the statements 1. and 2. of lemma 3.21, i.e., as suggested in definition 3.19.

We now discuss two new Hintikka lemmata, which take the logical nature of \( = \) into account.

**Theorem 3.23 (Hintikka Lemma for \( \mathfrak{A}(\mathfrak{M}_{\beta}) \)).** If \( \Gamma_\Sigma \) is a saturated \( \mathfrak{A}(\mathfrak{M}_{\beta}) \) and \( \mathcal{H} \) is maximal in \( \Gamma_\Sigma \), then the following statements hold for all \( A, B, C \in \text{wff}_a(\Sigma) \), \( F, G \in \text{wff}_a(\Sigma) \) and \( D, E \in \text{wff}_a(\Sigma) \):

\[ \nabla^e = (A =^\alpha A) \in \mathcal{H}. \]
\[ \nabla^e = \text{If } D[A]_\beta \in \mathcal{H} \text{ and } A =^\alpha B \in \mathcal{H}, \text{ then } D[B]_\beta \in \mathcal{H}. \]
\[ \nabla^g = A =^\alpha B \in \mathcal{H}, \text{ iff } (B =^\alpha A) \in \mathcal{H}. \]
\[ \nabla^f = A =^\alpha B \in \mathcal{H} \text{ and } B =^\alpha C \in \mathcal{H}, \text{ then } A =^\alpha C \in \mathcal{H}. \]
\[ \nabla^i = \lnot (F =^\alpha \beta G) \in \mathcal{H}, \text{ iff there is a } C \in \text{wff}_a(\Sigma), \text{ such that } \lnot (F \ C =^\beta G \ C) \in \mathcal{H}. \]
\[ \nabla^j = F =^\alpha \beta G \in \mathcal{H}, \text{ iff } F \ C =^\beta G \ C \in \mathcal{H} \text{ for all } C \in \text{wff}_a(\Sigma). \]
\[ \nabla^k = \lnot (A =^\alpha B) \in \mathcal{H}, \text{ iff } \lnot (A =^\alpha B) \in \mathcal{H}. \]

**Proof:**

\[ \nabla^e \text{ Follows by } \nabla^e_i \text{ and } \nabla^e_g. \]
\[ \nabla^e \text{ By maximality of } \mathcal{H} \text{ and } \nabla^e_i. \]
\[ \nabla^g \text{, } \nabla^f \text{ By } \nabla^e \text{ and } \nabla^e_i. \]
\[ \nabla^j \text{ By maximality of } \mathcal{H}, \text{ 3.21(3).} \text{ and 3.21(4).} \]
\[ \nabla^k \text{ By maximality of } \mathcal{H}, \text{ 3.21(1).} \text{ and 3.21(2).} \]
\[ \nabla^i \text{ Follows from } \nabla^e_q \text{ with } \nabla^i. \]
\[ \nabla^l \text{ Follows from } \nabla^e_q \text{ with } \nabla^l. \]

**Theorem 3.24 (Hintikka Lemma for \( \mathfrak{A}(\mathfrak{M}_{\beta}) \)).** If \( \Gamma_\Sigma \) is a saturated \( \mathfrak{A}(\mathfrak{M}_{\beta}) \) and \( \mathcal{H} \) is maximal in \( \Gamma_\Sigma \), then for all \( A, B \in \text{wff}_a(\Sigma) \):

\[ \nabla^i = \lnot (A =^\alpha B) \in \mathcal{H}, \text{ iff } \{ \lnot A, B \} \subseteq \mathcal{H} \text{ or } \{ A, \lnot B \} \subseteq \mathcal{H}. \]
\[ \nabla^j = A =^\alpha B \in \mathcal{H}, \text{ iff } \{ A, B \} \subseteq \mathcal{H} \text{ or } \{ \lnot A, \lnot B \} \subseteq \mathcal{H}. \]
\[ \nabla^k = A \leftrightarrow B \in \mathcal{H}, \text{ iff } A = B \in \mathcal{H}. \]
\[ \nabla^l = \text{Either } A = B \in \mathcal{H} \text{ or } A = \lnot B \in \mathcal{H}. \]

**Proof:** The statements follow direct from their counterparts \( \nabla^b = \nabla^b_i \) in lemma 3.17 with the help of \( \nabla^b_i \) and \( \nabla^b_i \).
3.4 Model Existence

We shall now present the proof of the abstract extension lemma, which will nearly immediately yield the model existence theorems. For the proof we adapt the construction of Henkin’s completeness proof from [Hen50, Hen96].

**Theorem 3.25 (Abstract Extension Lemma).** Let $\Sigma$ be a signature, $\Gamma_\Sigma$ be a compact abstract consistency class and let $H \in \Gamma_\Sigma$ be sufficiently $\Sigma$-pure. Then there exists a $\Sigma$-Hintikka set $H$ for $\Gamma_\Sigma$, such that $H \subseteq H$.

**Proof:** We construct $H$ by inductively constructing a sequence of sets $H^i$ such that $H^i \in \Gamma_\Sigma$. Then the $\Sigma$-Hintikka set is $H := \bigcup_{i \in \mathbb{N}} H^i \in \Gamma_\Sigma$.

Let $A_1, A_2, \ldots$ be an enumeration of $\text{wff}(\Sigma)$. We define $H^0 := H$ and the set $H^{n+1}$ according to the following table 3.1. Since the construction is uniform for all kinds of abstract consistency classes $H^{n+1}$ depends on the respective kind of abstract consistency class $\Gamma_\Sigma$ we are interested in and in the properties of $A_n$ with respect to this $\Gamma_\Sigma$. *(How to read the table: Assume $\Gamma_\Sigma$ is an an $\mathbb{A}_{\Sigma_\beta}$ and $A_n$ is of form $\neg(\mathbb{F}^{\alpha \rightarrow \beta} \mathbb{G})$. The table defines $H^{n+1}$ to be $H^n * A_n * \neg(\mathbb{F} w^{\beta} \mathbb{G} w)$ for a fresh $w \in C_n$ in case $A_n \in \Gamma_\Sigma$ and $H^n$ otherwise.)*

<table>
<thead>
<tr>
<th>$H^{n+1}$</th>
<th>$\mathbb{A}<em>{\Sigma</em>\beta}/\mathbb{A}<em>{\Sigma</em>\beta}$</th>
<th>$\mathbb{A}<em>{\Sigma</em>\beta}/\mathbb{A}<em>{\Sigma</em>\beta}$</th>
<th>$\mathbb{A}<em>{\Sigma</em>\beta}/\mathbb{A}<em>{\Sigma</em>\beta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H^n * A_n \notin \Gamma_\Sigma$</td>
<td>$H^n$</td>
<td>$H^n$</td>
<td>$H^n$</td>
</tr>
<tr>
<td>$H^n * A_n \in \Gamma_\Sigma$ and $A_n$ of form $\neg \Pi^\alpha B$</td>
<td>$H^n * A_n * \neg(\mathbb{B} w)$</td>
<td>$H^n * A_n * \neg(\mathbb{B} w)$</td>
<td>$H^n * A_n * \neg(\mathbb{B} w)$</td>
</tr>
<tr>
<td>$A_n$ of form $\neg(\mathbb{F}^{\alpha \rightarrow \beta} \mathbb{G})$</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
</tr>
<tr>
<td>$A_n$ of form $\neg(\mathbb{F}^{\alpha \rightarrow \beta} \mathbb{G})$</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
</tr>
<tr>
<td>$A_n$ of other form</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
<td>$H^n * A_n$</td>
</tr>
</tbody>
</table>

Figure 3.1: Hintikka set construction: How to construct sets $H^{n+1}$ from $H^n$

Next we show by induction, that $H^n \in \Gamma_\Sigma$ for all $n \in \mathbb{N}$. The base case holds by construction (for all kinds of abstract consistency classes). So let $H^0 \in \Gamma_\Sigma$. We have to show that $H^{n+1} \in \Gamma_\Sigma$. This is trivial in case $H^n * A_n \notin \Gamma_\Sigma$ (again for all abstract consistency classes). In case $H^n * A_n \in \Gamma_\Sigma$ we have to consider four sub-cases:

1. If $A_n$ is of form $\neg \Pi^\alpha B$, then we get the conclusion trivially by $\nabla_\alpha$ (for all cases).
2. If $A_n$ is of form $\neg(\mathbb{F}^{\alpha \rightarrow \beta} \mathbb{G})$ the conclusion is either trivial (by $\nabla_\alpha$ in case of $\mathbb{A}_{\Sigma_\beta}$, $\mathbb{A}_{\Sigma_\beta}$, $\mathbb{A}_{\Sigma_\beta}$, or $\mathbb{A}_{\Sigma_\beta}$), or follows by $\nabla_\beta$ in case of $\mathbb{A}_{\Sigma_\beta}$, $\mathbb{A}_{\Sigma_\beta}$, or $\mathbb{A}_{\Sigma_\beta}$, or follows by $\nabla_\beta$ and Lemma 3.21 (1), 3.21 (5), and 3.21 (2) in case of $\mathbb{A}_{\Sigma_\beta}$ or $\mathbb{A}_{\Sigma_\beta}$.
3. If $A_n$ is of form $\neg(\mathbb{F}^{\alpha \rightarrow \beta} \mathbb{G})$ the conclusion is either trivial (by $\nabla_\alpha$ in case of an $\mathbb{A}_{\Sigma_\beta}$, $\mathbb{A}_{\Sigma_\beta}$, $\mathbb{A}_{\Sigma_\beta}$, or $\mathbb{A}_{\Sigma_\beta}$, or follows by 3.21 (5).
4. If $A_n$ is of any other form, then the conclusion is trivial (for all cases).

Since $\Gamma_\Sigma$ is compact, we also have $H \in \Gamma_\Sigma$.

Now we know that our inductively defined set $H$ is indeed $\in \Gamma_\Sigma$ and that $H \subseteq H$. It only remains to show that $H$ is maximal in $\Gamma_\Sigma$. So let $A_n \in \text{wff}(\Sigma)$ be the n-th sentence from the above sequence, such that $H * A_n \in \Gamma_\Sigma$. Since $H$ is closed under subsets we know that $H^n * A_n \in \Gamma_\Sigma$. By definition of $H^{n+1}$ we conclude that $A_n \in H^{n+1}$ and hence $A_n \in H$. □

Next we define two congruence relations which we need in the model existence theorems below in order to build quotient models.
Definition 3.26 (Congruence Relations $\sim_{\mathcal{H}}$ and $\sim_{\mathcal{H}_1}$). Let $\Gamma_{\mathcal{H}}$ be an abstract consistency class and $\mathcal{H}$ be a Hintikka set for $\Gamma_{\mathcal{H}}$. For all $A, B \in \text{wff}(\Sigma)$ we define:

$$A \sim_{\mathcal{H}} B \iff A \equiv B \in \mathcal{H}.$$  

$$A \sim_{\mathcal{H}_1} B \iff \begin{cases} A \equiv B & \text{if } \gamma \equiv \iota \\ \{A, B\} \in \mathcal{H} \text{ or } \{A, B\} \cap \mathcal{H} \equiv \emptyset & \text{if } \gamma \equiv o \\ AC \sim_{\mathcal{H}} BC \text{ for all } C \in \text{wff}_{\mathcal{H}}(\Sigma) & \text{if } \gamma \equiv \alpha \rightarrow \beta \end{cases}$$

Lemma 3.27 (Functional Congruence Relations). Let $\Gamma_{\mathcal{H}}$ be an abstract consistency class and $\mathcal{H}$ be a Hintikka set for $\Gamma_{\mathcal{H}}$. Then $\sim_{\mathcal{H}}$ is a functional congruence relation, if $\Gamma_{\mathcal{H}}$ is a $\mathcal{AC}_2$ and $\sim_{\mathcal{H}_1}$ is a functional congruence relation, if $\Gamma_{\mathcal{H}}$ is an $\mathcal{AC}_{3\mathbb{B}}$.

Proof: $\sim_{\mathcal{H}}$ is a functional congruence relation by $\nabla_{\mathcal{H}}^+, \nabla_{\mathcal{H}}^{\#}, \nabla_{\mathcal{H}}^*, \nabla_{\mathcal{H}}^+$, which are valid in case $\Gamma_{\mathcal{H}}$ is an $\mathcal{AC}_{3\mathbb{B}}$.

Note that $\sim_{\mathcal{H}}$ is a functional congruence by construction.

Remark 3.28. Note that in Lemma 2.37 $\text{EXT}^\alpha_{\mathcal{H}}$ does not hold for $\approx$ and hence $\sim_{\mathcal{H}}$ is not a functional congruence in case $\Gamma_{\mathcal{H}}$ is not an $\mathcal{AC}_{3\mathbb{B}}$. Hence $\sim_{\mathcal{H}}$ is unsuitable for the model construction of an $\mathcal{M} \in \mathcal{M}_{3\mathbb{B}}$ (or $\mathcal{M} \in \mathcal{M}_{3\mathbb{B}}$) from a given $\mathcal{M}' \in \mathcal{M}_{\beta}$ (or $\mathcal{M}' \in \mathcal{M}_{\beta}$) as demonstrated below but fits well for the construction of an $\mathcal{M} \in \mathcal{M}_{3\mathbb{B}}$. Fortunately the relation $\sim_{\mathcal{H}}$ is already a functional congruence in case $\Gamma_{\mathcal{H}}$ is an $\mathcal{AC}_{3\mathbb{B}}$.

We now use the $\Sigma$-Hintikka sets, guaranteed by lemma 3.25, to construct a $\Sigma$-valuation for the $\Sigma$-term structure that turns it into the desired model $\mathcal{M}$.

Theorem 3.29 (Model Existence Theorem). Let $\Gamma_{\mathcal{H}}$ be an saturated $\mathcal{AC}$ and let $H \in \Gamma_{\mathcal{H}}$ be a sufficiently $\Sigma$-pure set of sentences. For all $i \in \{\beta, \beta, \beta, \beta, \beta, \beta, \beta\}$ (cf. Def. 2.28, 2.31 and 2.39) we have: If $\Gamma_{\mathcal{H}}$ is an $\mathcal{AC}_{\beta}$ (cf. Def. 3.6 and 3.18), then there exists a countable model in $\mathcal{M} \in \mathcal{M}_{\beta}$ that satisfies $H$.

Proof: Let $\Gamma_{\mathcal{H}}$ be an abstract consistency class. We can assume without loss of generality (see lemma 3.12) that $\Gamma_{\mathcal{H}}$ is compact, so the preconditions of 3.25 are met, and therefore there exists a $\Sigma$-Hintikka set $\mathcal{H} \subseteq \text{wff}_{\Sigma}(\Sigma)$ for $\Gamma_{\mathcal{H}}$ such that $H \subseteq \mathcal{H}$.

Now, for each different kind of abstract consistency class, we will construct a countable model $\mathcal{M}^\beta$ of the corresponding type. These model constructions closely reflect the relations of the different model types as discussed in Chapter 2.1 and shown in figures 1.1 and 2.1. We start with the construction of an $\mathcal{M}^\beta \in \mathcal{M}_{\beta}$ and an $\mathcal{M}^\beta \in \mathcal{M}_{\beta}$ based upon the non-functional termstructure $TS(\Sigma)^\beta$ and the functional $TS(\Sigma)^{\beta\beta}$. The remaining model constructions are then based upon these two basic constructions.

$\mathcal{M}_{\beta}$ Let $\Gamma_{\mathcal{H}}$ be an $\mathcal{AC}_{\beta}$. Given the $\Sigma$-Hintikka set $\mathcal{H}$ with $H \subseteq \mathcal{H}$ from above, we choose $\nu(C) := \mathcal{T}$, iff $C \in \mathcal{H}$. Note that we have $\nu(C) := \mathcal{T}$, iff $\neg C \in \mathcal{H}$ by $\nabla_{\mathcal{H}}^2$. By $\nabla_{\mathcal{H}}^2$ we know that $\nu$ is well-defined on $\text{wff}(\Sigma)^{\beta\beta}$ and by $\nabla_{\mathcal{H}}^2, \nabla_{\mathcal{H}}^2$ we have that $\nu$ is a total function on $TS(\Sigma)^{\beta\beta}$.

Furthermore by $\nabla_{\mathcal{H}}^2, \nabla_{\mathcal{H}}^2, \nabla_{\mathcal{H}}^2$ and $\nabla_{\mathcal{H}}^2$ we have that $\nu$ is a $\Sigma$-valuation of the $\Sigma$-term structure $TS(\Sigma)^{\beta\beta}$ and thus $\mathcal{M}^\beta := (TS(\Sigma)^{\beta\beta}, \nu)$ is a $\Sigma$-model by construction. We have $\mathcal{M}^\beta \models H$, since $H \subseteq \mathcal{H}$. Note that $\mathcal{M}^\beta$ is indeed countable, since the sets of well-typed formulae are countable.

$\mathcal{M}_{\beta\beta}$ Let $\Gamma_{\mathcal{H}}$ be an $\mathcal{AC}_{3\mathbb{B}}$ and hence also an $\mathcal{AC}_{3\mathbb{B}}$. Analogous to the previous case we construct the countable $\Sigma$-model $\mathcal{M}^\beta := (TS(\Sigma)^{\beta\beta}, \nu)$ with $\mathcal{M}^\beta \models H$. Note that in this case $\nu$ is well-defined on $TS(\Sigma)^{\beta\beta}$ because of $\nabla^\beta$. By lemma 2.15 we know that $\mathcal{M}^\beta$ is functional and hence $\mathcal{M}^\beta$ is in $\mathcal{M}_{\beta\beta}$. 


3.4. MODEL EXISTENCE

We proceed with the construction of an $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ and $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ based upon the previous construction of an $\mathcal{M}^H_1 \in \mathbb{M}_\beta$ and accordingly of an $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ and an $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ based upon an $\mathcal{M}^H_2 \in \mathbb{M}_{\beta_4}$. Thus we start out with a countable $\Sigma$-model $\mathcal{M}_1^H := (TS(\Sigma)^{\beta_1}, \nu)$ or $\mathcal{M}_2^H := (TS(\Sigma)^{\beta_2}, \nu)$, such that $\mathcal{M}_i^H \models H$, for $i = 1, 2$. Property $q$ is easy to verify, as it follows from the properties discussed in the Hintikka-lemmata 3.14 and 3.16.

$\mathbb{M}_{\beta_4}$ Let $H_2$ be an $\mathcal{A}cc_{\beta_4}$. From $\nabla^x, \nabla^{y_1}, \nabla^x_4, \nabla^u$ and $\nabla^v_4$ we can derive that $\Delta^\alpha$ is indeed the $q^\alpha$ required by property $q$ and hence $\mathcal{M}^H$ is a countable $\Sigma$-model in $\mathbb{M}_{\beta_4}$.

To verify property $b$ instead we have to construct an $\mathcal{M}^H$ from $\mathcal{M}^H$ ($i = 1, 2$) by reducing the set of truth values to $\{T, F\}$, which can be done with the help of a functional congruence relation.

$\mathbb{M}_{\beta_4}, \mathbb{M}_{\beta_4}$ Let $H_2$ be an $\mathcal{A}cc_{\beta_4}$ or $\mathcal{A}cc_{\beta_4}$. By lemma 3.27, $\nabla^x_4$ and $\nabla^c_4$ we can show that the relation $\sim_H$ defined in 3.25 is a functional $\Sigma$-congruence for $\mathcal{M}^H$ and thus, by lemma 2.31, the quotient structure $\mathcal{M}^H/\sim_H$ is a functional $\Sigma$-model that satisfies $H$. From $\nabla^x_4, \nabla^c_4$ and the choice of $\nu$ we conclude that $\sim_H$ has exactly two equivalence classes on $TS_\nu(\Sigma)^{\beta_4}$. Thus we have $D_\nu \equiv \{T \models [T]_\sim, F \models [F]_\sim\}$, if we define $T_\nu$ and $F_\nu$ as in lemma 2.25. Using $\nabla^x_4$ and $\nabla^c_4$ we further get that $\nu$ is the identity relation. Finally note that $\mathcal{M}^H/\sim_H$ is countable since $\mathcal{M}^H$ is.

We finish the constructions for the cases without a primitive notion of equality with the construction of a $\Sigma$-Henkin model $\mathcal{N}^H \in \mathbb{M}_{\beta_4}$ in case we are considering an $\mathcal{A}cc_{\beta_4}$.

We start with $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ guaranteed by the discussion above. Analogous to the construction of an $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$, we make use of a functional congruence relation in order to construct a quotient model which fulfills property $b$. But instead of the relation $\sim_H$ we had to use before, we apply the simpler relation $\sim_H$ which is a functional congruence relation for the elements of $\mathbb{M}_{\beta_4}$.

$\mathbb{M}_{\beta_4}$ By lemma 3.27 and $\nabla^x_4$, $\nabla^c_4$ we know that the relation $\sim_H$ is a functional $\Sigma$-congruence for $\mathcal{M}^H$, so the quotient structure $\mathcal{N}^H := \mathcal{M}^H/\sim_H \in \mathbb{M}_{\beta_4}$ with $\mathcal{N}^H \models H$ by lemma 2.34.

Now we can conclude that $D_\nu \equiv \{T, F\}$ with the same argumentation as in the $\mathbb{M}_{\beta_4}$ case.

It remains to discuss the cases with primitive equality and we start with the $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$, resp. $\mathcal{M}^H \in \mathbb{M}_{\beta_4}$ from above.

$\mathbb{M}_{\beta_4}$ Let $H_2$ be an $\mathcal{A}cc_{\beta_4}$ and hence an $\mathcal{A}cc_{\beta_4}$. We construct the countable $\mathcal{E}^H \in \mathbb{M}_{\beta_4}$ with $\mathcal{E}^H \models H$ as discussed above. It remains to show that property $e$ is valid for $\mathcal{E}^H$ which follows from property $q$ by $\nabla^x_4$ and $\nabla^c_4$.

$\mathbb{M}_{\beta_4}$ This case is analogous: Let $H_2$ be an $\mathcal{A}cc_{\beta_4}$ and hence an $\mathcal{A}cc_{\beta_4}$. We construct a countable $\mathcal{E}^H := (TS(\Sigma)^{\beta_4}, \nu) \in \mathbb{M}$ with $\mathcal{E}^H \models H$. Again property $e$ is valid for $\mathcal{E}^H$ by property $q$, $\nabla^x_4$ and $\nabla^c_4$. 

$\square$
Chapter 4

Extensional Higher-Order Resolution: \( \mathcal{ER} \)

In this chapter we introduce the calculus \( \mathcal{ER} \) for extensional higher-order resolution. The key idea of \( \mathcal{ER} \) is to integrate the search for unifiers and for refutations on the same level, i.e., we allow for recursive calls to the refutation process from within higher-order unification and vice versa. \( \mathcal{ER} \) is Henkin complete without additional axioms. In Section 4.1 we shall first discuss calculus \( \mathcal{ER} \) and illustrate the connections, modifications and extensions with respect to the underlying calculus \( \mathcal{HORES} \) as introduced in [Koh94b]. We then formally introduce calculus \( \mathcal{ER} \) in Section 4.2 and define \( \mathcal{ER}_f \) and \( \mathcal{ER}_b \), which generalise \( \mathcal{ER} \) by unfolding clause normalisation derivations and additionally providing the \textit{FlexH} unification rule. In Section 4.3 we present a simplified proof (compared to the technique employed in [Koh94b]) of a lifting lemma for the generalised calculus \( \mathcal{ER}_b \) before we examine Henkin completeness of \( \mathcal{ER}_b \) in Section 4.4. As we are actually more interested in calculus \( \mathcal{ER} \) than in \( \mathcal{ER}_b \) or \( \mathcal{ER}_f \), we discuss the equivalence of these three calculi in Section 4.5.

4.1 A Review of \( \mathcal{HORES} \) and \( \mathcal{ER} \)

Traditional first-order resolution [Rob65] can be seen as a two layered approach, where the overall search for a refutation (based on the resolution rule \textit{resolve} and the factorisation rule \textit{factorise}) is performed at a layer above: this layer passes subproblems to the lower layer, such as the initial clause normalisation process or the intermediate unification problems. An important fact is that all the side computations performed at the lower layer are decidable. First-order unification is the main engine of first-order resolution, it is in a sense a filter in the refutation search in order to separate inappropriate clauses from the search space and to compute most general representations for all suitable variable instantiations for the appropriate ones.

In our higher-order setting clause normalisation remains uncritical and the set of clauses \( \text{CNF}(\Phi) \) for a given set of higher-order formulæ \( \Phi \) can easily be computed with the clause normalisation rules as follows: Initially all formulæ \( A \in \Phi \) are replaced by pre-clause \( [A_{\lambda\kappa}]^\kappa \). Then the clause normalisation rules are exhaustively applied to \( \Phi \). Thus, in calculus \( \mathcal{ER} \) clause normalisation does not cause any decidability problems and can still be employed as a side computation (evoked by rule \textit{Cnf} of Figure 4.2) whenever it appears to be appropriate.

In contrast however to clause normalisation, higher-order unification is undecidable [SG89, Sny91] and can thus no longer be employed as a side computation like in the traditional first-order setting. Huet solved the undecidability problem in [Hue72, Hue73a] by delaying unification in his original constraint resolution approach instead of employing it as a filter. This is somewhat unrealistic in practice, as the filter effect is now delayed until the end too, and there are just too many candidates which fail only at the end of the computation. Therefore Kohlhase allows within his sorted variant of Huet’s resolution calculus \( \mathcal{HORES} \) (see [Koh94b]) for eager unification, i.e.,
the unification filter is (inspite of its theoretical undecidability) employed as early as possible during the refutation process. This is realistic as undecidable unification problems do rarely occur in practice.

The unsorted variant of \textit{HORES} provides the basis for our calculus \textit{ER} (as well as for \textit{EP} and \textit{ERIE}). The first modification of calculus \textit{ER} with respect to \textit{HORES} concerns Skolemisation: the Skolemisation technique employed by \textit{HORES} is not sound. \textit{HORES} adds special variable conditions to each clause in which the binding restrictions obtained from the Skolemisation steps are encoded. Consequently, after modifications of a clause (like the renaming of free variables) these variable conditions have to be updated as well. As the employed variable conditions and the updating mechanisms are not strong enough to prevent \textit{HORES} from proving obviously invalid statements like \textit{Each function has a fix-point} (\(\forall F_a \rightarrow \exists X_a. (F X) = X\)), we employ in calculus \textit{ER} traditional Skolemisation again. More precisely, we use Miller’s sound approach for higher-order Skolemisation [Mi83]. Another, more notational modification belongs to the encoding of unification constraints which we uniformly present as negated equations in all calculi introduced in this thesis. The most important modification of \textit{ER} over \textit{HORES} is that we add new extensionality rules to the calculus in order to reach Henkin completeness without the need for extensionality axioms. The rules of \textit{HORES} that are directly reflected in \textit{ER} are the clause normalisation rules presented in Figure 4.1 and basically the resolution and the pre-unification rules as stated in Figure 4.2 — except for the extensionality rules \textit{Equiv} and \textit{Leib}, and partly the extensionality rule \textit{Func}.

The rules of calculus \textit{ER} can be divided into the following three groups: clause normalisation rules, resolution rules, and extensional higher-order pre-unification rules. The set of clause normalisation rules are displayed in Figure 4.1 and the resolution and unification rules of \textit{ER} in Figure 4.2. For all rules we assume commutativity of \(\lor\), symmetry of \(\equiv\), and associativity of \(\lor\). Furthermore, we assume that the literals of our clauses are always kept in head-normal form. Consequently we suppose that the newly generated or modified clauses are always immediately transformed into head-normal form.¹

We now discuss the rules of \textit{ER} in detail and start with those that are directly imported from \textit{HORES}. We then describe extensionality rules that are new in \textit{ER} and which make \textit{ER} Henkin complete.

¹For the formal proofs in this chapter we do not assume that \textit{ER} automatically takes idempotency of \(\lor\) into account and does not automatically factorise identical literals in the clause normalisation process, since this saves our argumentations. In practice, however, one is certainly interested to optimise clause normalisation as far as possible.
**Chapter 4. Extensional Higher-Order Resolution: ER**

**Clause Normalisation:** (defined for arbitrary clauses)

\[
\frac{D \land C \in \mathcal{C} \land F(D)}{C} \quad \text{Cnf}
\]

**Resolution:** (defined on proper clauses only)

\[
\frac{[A]^{\alpha} \lor C \land [B]^{\beta} \lor D \land \alpha \neq \beta}{C \lor D \lor [A = B]^{F}} \quad \text{Res}
\]

\[
\frac{[A]^{\alpha} \lor [B]^{\alpha} \lor C \land \alpha \in \{T, F\}}{[A]^{\alpha} \lor C \lor [A = B]^{F}} \quad \text{Fac}
\]

\[
\frac{[Q \gamma U^{x}]^{\alpha} \land C \land P \in \mathcal{AB}_{\gamma}^{\alpha}[\{\gamma \lambda, \nu \mid \Pi \in \beta \in T\}] \land \alpha \in \{T, F\}}{[Q \gamma U^{x}]^{\alpha} \lor C \lor [Q \gamma P]^{F}} \quad \text{Pri}
\]

**Extensional (Pre-)Unification:** (defined for arbitrary clauses)

\[
\frac{C \lor [M_{a \Rightarrow B} = N_{a \Rightarrow B}]^{F} \land s_{a} \text{ Skolem term for this clause}}{C \lor [M s = N s]^{F}} \quad \text{Func}
\]

\[
\frac{C \lor [A_{a \Rightarrow B} = C_{a} = B_{a \Rightarrow D} = D_{a}]^{F}}{C \lor [A = B]^{F} \lor [C = D]^{F}} \quad \text{Dec}
\]

\[
\frac{C \lor [A = A]^{F}}{C} \quad \text{Triv}
\]

\[
\frac{C \lor [X = A]^{F} \land X \notin \text{free}(A)}{C[A / X]} \quad \text{Subst}
\]

\[
\frac{C \lor [F_{\gamma} U_{n} = h \\nabla m]^{F} \land G \in \mathcal{AB}_{\gamma}^{h}}{C \lor [F = G]^{F} \lor [F \equiv = h \\nabla m]^{F}} \quad \text{FlexRigid}
\]

\[
\frac{C \lor [M_{a} = N_{a}]^{F}}{C \lor [M_{a} \leftrightarrow N_{a}]^{F}} \quad \text{Equiv}
\]

\[
\frac{C \lor [M_{a} = N_{a}]^{F}}{C \lor [P \forall a \cdot P \Rightarrow M \Rightarrow P]^{F}} \quad \text{Leib}
\]

\[\mathcal{AB}_{\gamma}\] is the set of partial bindings of type \(\gamma\) for head \(h\) as defined in [SG89]

Figure 4.2: The Extensional Higher-Order Resolution calculus \(\mathcal{ER}\)
4.1. A REVIEW OF HORES AND ER

The higher-order resolution rule Res and factorisation rule Fac employed in $\mathcal{ER}$ and $\mathcal{HORES}$ obviously differ from their first-order counterparts. Instead of using unification as a filter, which checks the rules applicability and even computes a most general representation of all suitable variable instantiations justifying the particular resolution or factorisation steps, they add — as suggested by Huet — respective unification constraints to the generated clauses and generally delay the application of unification. Consequently, the search space thereby explodes as any two literals with contrary polarities can be resolved upon and any two literals in a clause with identical polarities can be factorised.

In order to avoid this search space explosion $\mathcal{ER}$ and $\mathcal{HORES}$ allow for eager pre-unification. This is realised by rule Subst which propagates (partly) solved unification constraints back to the other literals of the same clause. Thus, the idea is not to delay unification in $\mathcal{ER}$ till an empty clause is derived, but to employ it early and parallel to the overall refutation search, thereby partly regaining the filter properties of unification. Clearly, as higher-order unification and preunification is generally undecidable we still cannot prematurely decide all unification problems — but many.

After applying rule Subst (or analogously the extensionality rules Equiv or Leib as introduced below), clause normalisation may become necessary in order to obtain proper clauses again. This is due to the fact that instantiating predicate variables at head positions of some literals, i.e., flexible literal heads, may lead to pre-clauses instead of proper ones. The clause normalisation process is evoked by the application of rule Cnf. It performs exhaustive $\mathcal{CNF}$-derivations from a pre-clause $\mathcal{D}$ to a proper clause $\mathcal{C} \in \mathcal{CNF}(\mathcal{D})$ according to the rules presented in Figure 4.1. Thus, the whole derivation according to the $\mathcal{CNF}$-rules is hidden inside the single rule application of Cnf in calculus $\mathcal{ER}$.

It is well known for higher-order resolution that a primitive substitution rule (as suggested in $\text{[And89]}$) or a splitting rule (see $\text{[Hue72]}$) is needed, as unification is too weak to compute all necessary instantiations for flexible literal heads.

The primitive substitution rule Prim provided by calculus $\mathcal{ER}$ is a variant of the rule suggested in $\text{[And89]}$ and is conceptually simpler than Huet’s splitting rule. Rule Prim allows to instantiate flexible heads of the literals by a partial binding that imitates a logical constant. The important role of this rule can be illustrated by the example $\exists X_1 \exists Y_1 X_1 \land Y_1$ which is obviously a theorem with respect to Henkin semantics. By negation and clause normalisation we obtain the two unit clauses $[X]^F$ and $[Y]^F$. Both clauses consist of exactly one negated literal with a flexible head. Neither resolution nor factorisation is applicable and thus without the primitive substitution rule we cannot find a refutation. The application of Prim with the partial binding $\{X \leftarrow \neg X_1\}$ on clause $[X]^F$ results after clause normalisation with rule Cnf in $[X]^F$. Thus, by applying rule Prim we add important but missing logical structure to our clauses, such that a refutation with the other calculus rules becomes possible.

$\mathcal{HORES}$ is not complete with respect to Henkin semantics. The problem is that despite the primitive substitution rule Prim, which in some sense supports higher-order unification algorithm in computing instantiations of variables\footnote{Clause normalisation removes logical structure from the input formulas and translates them into the clause structure. The primitive substitution rule on the other hand can always introduce new logical structure for flexible literal heads, which cannot be computed by unification. In this sense primitive substitution supports higher-order unification in a refutation approach.}, the unification rules are still too weak to handle the extensionality principles sufficiently. More precisely, higher-order unification as employed in $\mathcal{HORES}$ or in Huet’s original approach is a pure syntactically oriented algorithm for unifying terms. But for reaching Henkin completeness we need unification with respect to the theory defined by the extensionality principles, as we are interested to unify terms like $A_{\alpha} \land B_{\alpha}$ and $B_{\alpha} \land A_{\alpha}$ or even $\lambda X_{\alpha} A_{\alpha^\to} X \land B_{\alpha^\to} X$ and $\lambda X_{\beta} B_{\beta^\to} X \land A_{\beta^\to} X$.

Clearly, $\mathcal{HORES}$ as well as the traditional approaches $\text{[And71, Hue72]}$ can be made Henkin complete by adding the extensionality axioms to the search space, which is unfeasible in practice.

As a solution to this problem, the calculus $\mathcal{ER}$ adds the three new extensionality rules $\text{Leib}$, $\text{Equiv}$, and $\text{Func}$ to the unification rules and thereby avoids the extensionality axioms in the search space. Rule Leib (see Figure 4.2) simply instantiates the equality symbol within unification
constraints by its Leibniz definition and \textit{Equiv} reflects the extensionality property for truth values in a negative way: if two formulae are not equal, then they are also not equivalent. Rule \textit{Func} analogously reflects functional extensionality: if two functions are not equal then there exists a witness \( s_\alpha \) on which these functions differ. To ensure soundness, \( s_\alpha \) has to be a new Skolem term that contains all the free variables occurring in the given clause. Why is rule \textit{Func} presented as a new extensionality rule but also as a usual unification rule? The reason is that the pre-unification rules \( \alpha \) and \( \eta \) as presented in \textit{HORES} already partially realise the negative aspect of the functional extensionality principle:

\[
C \lor [(\lambda x_{\alpha}, A) = (\lambda y_{\alpha}, B)]^F \\
\alpha
\]

\[
C \lor [A_{(s/x)} = B_{(s/y)}]^F
\]

\[
C \lor [(\lambda x_{\alpha}, A) = B]^F \\
\eta
\]

\[
C \lor [A_{(s/x)} = (B_s)]^F
\]

Note that the purely type information based rule \textit{Func} extends and generalises these two rules and thus rule \textit{Func} has the following two meanings in our calculus: If one or both unification terms is a \( \lambda \)-abstraction, it works like the traditional \( \alpha \)- and \( \eta \)-rules as, e.g., used in \textit{HORES} and [BK98a]. If on the other hand neither of the unification terms is a \( \lambda \)-abstraction term, \textit{Func} realises the functional extensionality principle.

In cooperation the new extensionality rules connect the unification part of our calculus with the resolution part by allowing for recursive calls of the overall refutation process from within higher-order unification.

We want to point out that none of the three new extensionality rules introduces any flexible literal and even better, they introduce no new free variable at all; even if they heavily increase the search space for refutations, they behave much better — as the experiments (see Section 7.4) with the LEO theorem prover [BK98b, Ben97] showed — than the addition of extensionality axioms in the traditional approaches, which introduces many flexible literals in the refutation process.

One important aspect that is illustrated by the examples in this thesis as well as the examples discussed in [BK98a] is that eager pre-unification becomes essential in \( \mathcal{ER} \) and many proofs cannot be found when delaying the unification process until the end.

Another important modification of \( \mathcal{ER} \) with respect to \textit{HORES} concerns the encoding of unification constraints. In \( \mathcal{ER} \), they are encoded as negated equational literals. In this sense a clause \( C \lor [L_1 = R_1]^F \lor \ldots \lor [L^n = R^n]^F \) can also be read as the implication \( [L_1 = R_1]^F \land \ldots \land [L^n = R^n]^F \rightarrow C \), i.e., the unification constraints describe the conditions under which clause \( C \) holds. Consequently the question argues whether or not resolution and factorisation rules are allowed to be applied on these unification constraints, which look like ordinary literals. In order to obtain a Henkin complete calculus this is not necessary — as the completeness proofs in [BK98a, BK97b] and the alternative one in this thesis show. Consequently the unification constraints do not necessarily have to be encoded as negated equational literals, any other form will work as well. But the encoding of unification constraints as negated equational literals becomes essential for the extensional higher-order RUE-resolution calculus presented in Chapter 6 and the extensional higher-order paramodulation calculus presented in Chapter 5.

We briefly sum up the particular modifications with respect to [BK98a]:

- The unification rules \( \alpha \) and \( \eta \) employed in [BK98a] are avoided as they are subsumed by rule \textit{Func}.

- Instead of employing clause normalisation in the definitions of rule \textit{Subst} and the extensionality rules \textit{Equiv} and \textit{Leib} we add the extra clause normalisation rule \textit{Cnf} to the calculus.

- We slightly modify the decomposition rule \textit{Dec}. This modification is illustrated in detail by Example \textit{E}^{Dec} in Subsection 8.2.

We will present in this chapter an alternative proof for the Henkin completeness of calculus \( \mathcal{ER} \) to the one given in [BK98a, BK97b]. The motivation for this new proof is threefold:
4.2. BASIC DEFINITIONS

- We have slightly modified the calculus $\mathcal{ER}$ in this thesis.
- The completeness proofs of the new, further extended calculi $\mathcal{EP}$ (extensional higher-order paramodulation) and $\mathcal{ERUE}$ (extensional higher-order RUE-Resolution) are carried out analogously, such that many lemmata can be either directly reused or with minor modifications.
- The lifting lemma in the completeness proofs in [BK98a, BK97b] builds upon an argument also employed in [Koh94b] which uses a quite complicated notion of clause isomorphisms susceptible to errors. In this thesis we present a lifting argument that omits the notion of clause isomorphisms. This is possible as we analyse a generalised resolution calculus $\mathcal{ER}_c$ instead of $\mathcal{ER}$. This enriched calculus additionally employs the instantiation guessing $\mathit{FlexFlex}$-rule (see Figure 4.3) and applies the single clause normalisation rules instead of grouping them into exhaustive clause normalisation chains with rule $\mathit{Cnf}$. Consequently, in Subsection 4.5 we will discuss the theorem equivalence between $\mathcal{ER}_c$ and $\mathcal{ER}$.

An important convention for this and the following chapters concerns $\alpha$-equality of clauses and the arity of Skolem terms:

**Remark 4.1 (Equality of clauses).** In resolution based theorem proving one usually assumes all clauses to be variable disjoint. In practice this is achieved by automatically renaming the variables within each newly generated clause. In this thesis we implicitly use this convention, too.

Another implicit convention concerns the Skolem terms. We briefly illustrate this aspect by an example. Assume that the following pre-clause is given:

$$C_1 : \forall X_i. p_{i \to o} X_i Y_i \top \lor \forall Z_i. q_{i \to o} Z \top^F$$

Clause normalisation either leads to

$$C_2 : [p_{i \to o} X_i Y_i \top \lor [q_{i \to o} (s_{1 \to i}^1 Y_i)]^F \quad \text{or to} \quad C_3 : [p_{i \to o} X_i Y_i \top \lor [q_{i \to o} (s_{1 \to i}^2 X_i Y_i)]^F$$

where $(s_{1 \to i}^1 Y_i)$ and $(s_{1 \to i}^2 X_i Y_i)$ are new Skolem terms. The first clause $C_2$ is the result of applying rule $\Pi^F$ first and $\Pi^F$ to the result, whereas the second clause $C_3$ is the result of applying first $\Pi^F$ and then $\Pi^F$. Both results differ with respect to the arity of the new Skolem terms. It is well known for refinement approaches that each refinement using only one of these clauses can be analogously carried out with the other one. For a discussion of this Skolemisation aspect in the context of sequent calculus we refer to [AMS98]. In the following we will therefore ignore the different arities of Skolem terms caused by switching the order of single applications of clause normalisation rules (switching the order of clause normalisation rules will be employed in some of the proofs in this thesis).

4.2 Basic Definitions

Instead of a direct proof of Henkin completeness for $\mathcal{ER}_c$ we first analyse the slightly enriched calculus $\mathcal{ER}_c$. Aside from the unfolding of exhaustive clause normalisation derivations this calculus provides the well known $\mathit{FlexFlex}$-unification rule displayed in Figure 4.3. In case a clause contains a $\mathit{flex-flex}$-unification constraint this rule allows to guess an instantiation for one of the flexible heads such that the unification process can proceed with its eager unification attempts. It was already pointed out by Huet [Hue72] that in practice we are interested in avoiding this possibly infinitely branching rule (there may be infinitely many constants in the signature) and it turned out that within a refinement approach one can in fact avoid the $\mathit{FlexFlex}$-rule and delay the operations on $\mathit{flex-flex}$-constraints until one of the head variables gets bound. However, employing this additional rule within the lifting lemma cases the proofs as it turns our eager pre-unification approach into an eager unification approach. This allows to omit the clause isomorphisms that are needed in the respective proofs in [Koh94b]. The motivation for the unfolding of exhaustive clause normalisation derivation by rejecting rule $\mathit{Cnf}$ and lifting the clause normalisation rules to calculus level is analogous: We want to close the proofs in this section and especially the analogous
\[
\frac{C \cup [F \sigma_\alpha] \quad \Gamma \vdash_{HF} \Phi \quad \Gamma \vdash H \quad \forall \alpha \quad G \in \mathcal{A}B_{\forall \alpha} \quad \text{for a } h \in C}{C \cup [F] \quad \Gamma \vdash_{HF} \Phi \quad \Gamma \vdash [F = G] \quad \text{FlexFlec}}
\]
Figure 4.3: The FlexFlec unification rule

but slightly more complicated ones for the extensional higher-order paramodulation calculus $\mathcal{EP}$ and extensional higher-order RUE-resolution calculus $\mathcal{ERUE}$ in the following sections.

A formal proof for the admissibility of rule FlexFlec has not been carried out yet, but evidence is given by [BK98a] (the completeness proof presented there avoids rule FlexFlec but admittedly lacks a bit of transparency and clarity within the lifting argument) and the case studies with the LEO prover [BK98b]. Actually, there is no example known to the author that requires the application of rule FlexFlec.

**Definition 4.2 (Clause Normalisation).** The calculus $\mathcal{CNF}$ consists of the clause normalisation rules displayed in Figure 4.1. We assume that the result of each rule application is transformed into head-normal form. By applying these rules exhaustively to a pre-clause $[A]^{\alpha}$ ($\alpha \in \{T, F\}$) one can derive the set $\mathcal{CNF}([A]^{\alpha})$ of proper clauses derivable from $[A]^{\alpha}$.

**Lemma 4.3 (Soundness of $\mathcal{CNF}$).** The rules $\mathcal{CNF}(\Pi F)$ preserve validity and the rules $\Pi F$ preserves satisfiability with respect to Henkin semantics.

**Proof:** The proofs for the rules in $\mathcal{CNF}(\Pi F)$ are analogous to the first-order case and Skolemisation has been corrected for higher-order logic by Miller in [Mil83, Mil91, Mil92]. □

**Definition 4.4 (Unification).** We define the following two calculi for higher-order unification and higher-order pre-unification:

- $\mathcal{UNT}$: The calculus $\mathcal{UNT}$ consists of the pre-unification rules Try, Func, Dec, FlexRigid and Subst as presented in Fig 4.2. We assume that the result of each rule application is transformed into head-normal form. These rules form a quite close variant of the higher-order pre-unification calculus discussed in [Koh94b] and [SG89, Sny91]. We already mentioned that rule Func, which simply applies the functional extensionality principle, subsumes the rules $\alpha$ and $\eta$ used in [BK98a] and [Koh94b].

- $\mathcal{UNT}_2$: The calculus $\mathcal{UNT}_2$ is defined as $\mathcal{UNT} \cup \{\text{FlexFlec}\}$.

**Theorem 4.5 (Higher-Order Unification and Pre-Unification).**

- **Soundness** $\mathcal{UNT}_2$ (resp. $\mathcal{UNT}$) is a sound calculus for higher-order unification (resp. higher-order pre-unification). More precisely, for each derivation $\Delta : E \equiv [L_1 = R_1]^{F} \lor \ldots \lor [L_n = R_n]^{F} \vdash_{\mathcal{UNT}_2} E'$ (resp. $\Delta : E \vdash_{\mathcal{UNT}} E'$), where $E'$ is in solved form and corresponds with unifier (pre-unifier) $\sigma$, holds that $\sigma$ is a unifier (resp. pre-unifier) of $\{L_1 = \overline{1} R_1, \ldots, L_n = \overline{1} R_n\}$.

- **Completeness** $\mathcal{UNT}_2$ (resp. $\mathcal{UNT}$) is a complete calculus for higher-order unification (resp. higher-order pre-unification). More precisely, for each higher-order unification problem $\{L_1 = \overline{1} R_1, \ldots, L_n = \overline{1} R_n\}$ with unifier (resp. pre-unifier) $\sigma$, there exists a derivation $\Delta : E \equiv [L_1 = R_1]^{F} \lor \ldots \lor [L_n = R_n]^{F} \vdash_{\mathcal{UNT}_2} E'$ (resp. $\Delta : E \vdash_{\mathcal{UNT}} E'$), such that $E'$ is in solved form and corresponds with unifier (resp. pre-unifier) $\sigma$.

---

3Remember the special definition of head-normal form for unification constraints as given in Chapter 2.1.

4Note that this is the main difference to the pre-unification rules presented in [SG89] which presupposes that all results are reduced to $\beta_n$-normal form.
4.2. BASIC DEFINITIONS

Proof: We will not present a formal proof here and instead refer to [Koh94b]. The only difference of our rules to the respective ones used in [Koh94b] is, that the latter consider sorts as well and employ a extra-logical form of Skolemisation. Note that our set of rules furthermore only slightly modifies the set of unification rules discussed in [SG89, Sny91].

The first complete set of transformations for higher-order unification was defined in [Pie73, Hue73a] and undecidability of higher-order unification was first discussed in [Hue73b]. Huet then introduced higher-order pre-unification in [Hue75]. For a modern presentation of higher-order unification and pre-unification we refer to [SG89, Sny91]. Sorted higher-order unification and pre-unification is discussed in [Koh94b].

As the calculus $\mathcal{H}$ realises higher-order unification and as rule Subst allows to propagate solutions back to the non-unification constraints of a clause we get the following corollary.

Corollary 4.6 (Higher-Order Unification). Let $\mathcal{C} \cup E$ be a clause with unification constraints $E$. Then for each unifier $\sigma$ of $E$ we have that $\mathcal{C} \cup E \cup \cup \cup \mathcal{C}_\sigma$.

Definition 4.7 (Extensional Higher-Order Resolution).

$\mathcal{ER}$ The calculus $\mathcal{ER}$ consists of the following inference rules displayed in Figure 4.2, i.e.,

$\mathcal{ER} := \{ Cnf, Res, Fac, Prim \} \cup \{ Lib, Equr \}$

$\mathcal{ER}_f$ The extension $\mathcal{ER}_f$ of calculus $\mathcal{ER}$ that employs full higher-order unification instead of higher-order pre-unification is defined as $\mathcal{ER}_f := \mathcal{ER} \cup \{ FlexFlex \}$.

$\mathcal{ER}_{fE}$ The calculus $\mathcal{ER}_{fE}$ that employs stepwise instead of exhaustive clause normalisation is defined by $\mathcal{ER}_{fE} := (\mathcal{ER} \setminus \{ Cnf \}) \cup \{ Cnf \}$.

A set of formulae $\Phi$ is refutable in calculus $R \in \{ \mathcal{ER}, \mathcal{ER}_f, \mathcal{ER}_{fE} \}$, iff there is a derivation $\Delta : \Phi_R \vdash_R \Box$, where $\Phi_R := \{ [F]_R \}| F \in \Phi \}$ is the set obtained from $\Phi$ by simple pre-classification. We remark again, that unification constraints are treated as special literals, which are only accessible to the unification rules.

Remark 4.8 (General Higher-Order E-Unification). The extensional higher-order resolution calculus $\mathcal{ER}$ ($\mathcal{ER}_f$ or $\mathcal{ER}_{fE}$) can also be viewed as a test calculus for general higher-order $E$-unifiability ($E$-unifiability): Assume an arbitrary set of equations $E_1 \ldots E_n$, describing a theory $E$ and an $E$-unification problem $T_1 = T_2$ is given. If we pass clauses $[E_1]^T \ldots [E_n]^T$ and $[T_1 = T_2]^T$ as an input problem to our calculus, then the calculus $\mathcal{ER}$ tests if the unification constraint $[T_1 = T_2]^T$ is solvable with respect to theory $E$ enriched by the extensionality properties. The overall answer substitution computed by the refutation is obviously also an answer substitution to our $E$-unification problem.

Theorem 4.9 (Soundness of Extensional Higher-Order Resolution).
The calculi $\mathcal{ER}$, $\mathcal{ER}_f$ and $\mathcal{ER}_{fE}$ are sound for Henkin semantics, i.e., let $\Phi$ be a set of formulae, such that $\Phi \vdash_R \Box$, then $\Phi$ is unsatisfiable with respect to Henkin semantics (and consequently standard semantics).

Proof: We already know by lemma 4.3 that the rules in $\mathcal{CNF}$ either preserve validity or satisfiability with respect to Henkin models and thus the latter also holds for compound rule Cnf. The extensionality principles are valid in Henkin models (see Lemmata 2.37 and 2.43) and thus the extensionality rules Lib and Equr preserve validity wrt. Henkin semantics, whereas rule Func only preserves satisfiability as it immediately applies Skolemisation after employing the functional extensionality principle. All remaining unification rules as well as the resolution rules Res, Fac, and Prim can easily be shown to preserve validity with respect to Henkin models. Now the assertion follows from the well known result that preservation of satisfiability ensures soundness within a refutation approach.

The following lemma states that each non-proper clause $\mathcal{C}$ in a clause set $\Phi$ can be replaced by its corresponding set of proper clauses $\mathcal{CNF}(\mathcal{C})$, without affecting the set of derivable proper clauses. We want to remark that this lemma implicitly employs the convention of Remark 4.1.
This lemma is applied in the completeness proof of calculus $\mathcal{ER}_c$, more precisely, in Lemma 4.15, which is then applied in the completeness proof to verify the consistency properties $\nabla_\nu$ and $\nabla_\theta$, and to show saturation. In the completeness proofs for $\mathcal{ER}_c$ and $\mathcal{ER}c_c$, analog of this lemma will be additionally employed to verify the abstract consistency property $\nabla_i'$ (again indirect within the proofs of respective lemmata). The Lemma can be proven for all three calculi, but we will need it only for calculus $\mathcal{ER}_c$.

**Lemma 4.10 (Proper Derivations).** For each clause set $\Phi$, clause $\mathcal{C}$, and proper clause $\mathcal{C}'$, such that $\Phi * \mathcal{C} \vdash_{\mathcal{ER}_c} \mathcal{C}'$, we have $\Phi \cup \mathcal{CNF}(\mathcal{C}) \vdash_{\mathcal{ER}_c} \mathcal{C}'$.

**Proof:** The proof is by induction on the length of the derivations $\Phi * \mathcal{C} \vdash_{\mathcal{ER}_c} \mathcal{C}'$. In the base case ($n \equiv 0$) we know that $\mathcal{C}' \in \Phi$ as $\mathcal{C}'$ must be different from $\mathcal{C}$. Thus, the assertion holds trivially. In the induction step ($n > 0$) we consider the first step in derivation $\Phi * \mathcal{C} \vdash_{\mathcal{ER}_c} \mathcal{C}'$, where $\mathcal{D}$ is a clause. If $\mathcal{C}$ is not a premise clause for the application of rule $r$ then the assertion follows immediately by induction hypotheses. Thus, let us assume that $\mathcal{C}$ is a premise clause for the application of rule $r$. By induction hypotheses applied to clause $\mathcal{D}$ we get that $(\Phi * \mathcal{C}) \cup \mathcal{CNF}(\mathcal{D}) \vdash_{\mathcal{ER}_c} \mathcal{C}'$. As the resolution rules Res, Prim, Fac are only defined on proper clauses, we know that the only rules that are possible for $r$ are the unification rules in $\text{UN}_\mathcal{F}$ and the rule $\text{CNf}$. It is easy to check that in all those cases we have that $\mathcal{CNF}(\mathcal{C}) \vdash_{\mathcal{ER}_c} \mathcal{CNF}(\mathcal{D})$. And thus, we finally get that $\Phi \cup \mathcal{CNF}(\mathcal{C}) \vdash_{\mathcal{ER}_c} \Phi \cup \mathcal{CNF}(\mathcal{C}) \cup \mathcal{CNF}(\mathcal{D}) \vdash_{\mathcal{ER}_c} \mathcal{C}'$. □

### 4.3 Lifting Properties

This subsection examines some lifting properties of the calculus $\mathcal{ER}_c$. We shall prove a lifting argument for all clause normalisation rules in $\mathcal{CNF}$ before we then present the lifting lemma for $\mathcal{ER}_c$. The lifting lemma for calculus $\mathcal{ER}_c$, which states that each derivation in $\mathcal{ER}_c$ performed on an instantiated level has a direct counterpart on the uninstantiated level as well, turns out to be quite trivial. The reason is that $\mathcal{ER}_c$ provides rule $\text{Flex}$, which can be employed in addition to rule Prim to introduce the necessary instantiations on the abstract level whenever the replay of the derivation given on the instantiated level is blocked caused by missing clause or unification term structure. Furthermore, clause normalisation rules are not grouped into single rule $\text{CNf}$ but treated as single inference rules of their own, which also ease the particular arguments.

**Lemma 4.11 (Lifting Lemma for Clause Normalisation).** Let $\mathcal{D}_1, \mathcal{D}_2$ be clauses and $\sigma$ be a substitution. For each derivation $\Delta_1 : (\mathcal{D}_1)_\sigma \vdash_{\mathcal{CNF}} \mathcal{D}_2$ exists a clause $\mathcal{D}_3$, a substitution $\delta$, and a derivation $\Delta_2 : \mathcal{D}_1 \vdash_{\mathcal{ER}_c} \mathcal{D}_3$, such that $(\mathcal{D}_3)_\sigma \delta \equiv \mathcal{D}_2$.

**Proof:** The proof is by case distinction on the rules in $\mathcal{CNF}$. As all cases are analogous we only consider the $\text{VI}_f$ rule here. In this case we have that $(\mathcal{D}_1)_\sigma = (L_1)_\sigma \lor \ldots \lor (L_n)_\sigma \lor [A_\sigma \lor B_\sigma]^F$ and $\mathcal{D}_2 := (L_1)_\sigma \lor \ldots \lor (L_n)_\sigma \lor [A_\sigma \lor B_\sigma]^F$ for some literals $(L_i)_\sigma$, $1 \leq i \leq n$, and terms $A_\sigma, B_\sigma$.

We now consider the possible structure of the focused literal in the uninstantiated level $\mathcal{D}_1 : L_1 \lor \ldots \lor L_n \lor [C]^F$.

In case $[C]^F \equiv [A \lor B]^F$ we can obviously apply $\text{VI}_f$ leading to $\mathcal{D}_3 : L_1 \lor \ldots \lor L_n \lor [A]^F$, such that the assertion follows trivially.

Otherwise we have $[C]^F \equiv [H \cup \overline{m}]^F$, such that $(H \cup \overline{m})_\sigma \equiv A_\sigma \lor B_\sigma$, as either (i) $H_\sigma \equiv \lambda X^m \cdot L \lor R$, with $([L \cup \overline{m} / X^m])_\sigma \equiv A_\nu$ and $([R \cup \overline{m} / X^m])_\sigma \equiv B_\nu$, or (ii) $H_\sigma \equiv \lambda X^m \cdot X^n \cdot T$ (for $1 \geq n \geq m$), with $((X^n \cdot T)[U^m / \overline{m}])_\sigma \equiv A_\sigma \lor B_\sigma$.

Case (i): we now apply rule Prim to clause $\mathcal{D}_1$ with general binding $\lambda X^m \cdot (H^1 \overline{m}) \lor (H^2 \overline{m})$ for predicate variable $H$ ($H^1, H^2$ are new predicate variables of appropriate type) and obtain the clause:

$$\mathcal{D}_4 : L_1 \lor \ldots \lor L_n \lor [H \equiv \lambda X^m \cdot (H^1 \overline{m}) \lor (H^2 \overline{m})]^F$$

With rule $\text{Subst}$ we get clause

$$\mathcal{D}_5 : (L_1)_\sigma \lor \ldots \lor (L_n)_\sigma \lor [(H^1 \overline{m}) \lor (H^2 \overline{m})]^F$$
where $\tau := \{\lambda x^m. (H^1 \ X^m) \lor (H^2 \ X^m) / H\}$. Now rule $V^F$ is applicable which leads to clause $D_3 : \langle L_1 \rangle \lor \ldots \lor \langle L_m \rangle \lor (H^1 \ U^m)^F$. It is easy to verify that $\tau$ is more general than $\sigma$ (because of the flexible variables $H^1$ and $H^2$) and that hence there must be a substitution $\gamma$ that appropriately instantiates the new predicate variables $H^1$ and $H^2$ and that coincides with $\sigma$ on all other variables, such that we have $(\langle L_1 \rangle \gamma)^{\tau} \equiv (\langle L_1 \rangle \sigma) (H^1 \ U^m)^{\gamma \sigma} \equiv A_\sigma$. This finally proves the assertion as $(D_3 \gamma)^{\tau} \equiv D_2$.

Case (ii): In this case the assertion follows analogously if we apply rule $Prim$ with the projection binding $\lambda x^m. X^n (H \ X^m)$.

We are now ready to prove the lifting lemma for calculus $\mathcal{ER}_c$. The main result (a special case of statement 4.12(2)) is that for any substitution $\sigma$ and clause set $\Phi$ holds that $\Phi$ is refutable in calculus $\mathcal{ER}_c$ if provided that $\Phi_\sigma$ is. Here we even prove a more general result, stating that lifting is even possible for general derivations and not only for refutations.

**Lemma 4.12 (Lifting Lemma for $\mathcal{ER}_c$).** Let $\Phi$ be a set of clauses, $D_1$ be a clause and $\sigma$ a substitution. We have that:

1. For each derivation $\Delta_1 : \Phi_\sigma \vdash \mathcal{ER}_c \ D_1$ there exists a substitution $\delta$, a clause $D_2$ and a derivation $\Delta_2 : \Phi_{\mathcal{ER}_c} \ D_2$, such that $(D_2)_{\delta \circ \sigma} \equiv D_1$.

2. For each derivation $\Delta_1 : \Phi_\sigma \vdash \mathcal{ER}_c \ D_1$ there exists a substitution $\delta$, a clause $D_2$ and a derivation $\Delta_2 : \Phi_{\mathcal{ER}_c} \ D_2$, such that $(D_2)_{\delta \circ \sigma} \equiv D_1$.

**Proof:**

(1) The proof is by case distinction on all rules in $\mathcal{ER}_c$ and in all cases we construct a derivation $\Delta_2$ as required.

- **Res** Assume the first step in $\Delta_1$ employs resolution rule $Res$ to clauses $[A_a]^{\sigma} \lor C$ and $[B_b]^{\beta} \lor D$ with resolvent $D_1 : C_a \lor D_b \lor [A_{a=b}]^F$. Then an analogous resolution step is possible between $[A]^{\sigma} \lor C$ and $[B]^{\beta} \lor D$ leading to resolvent $D_2 : C \lor D \lor [A=B]^F$. We trivially have that $(D_2)_{\sigma} \equiv D_1$.

- **Prim** Assume the first step in $\Delta_1$ employs rule $Prim$ to a flexible literal in a clause $C \lor [H (A_x)^\gamma]^\sigma$ leading to clause $D_1 : C_a \lor [H (A_x)^\gamma]^\sigma \lor [H = G]^F$, where $G$ is a general binding for variable $H$ imitating a logical connective. Then an analogous proof step with an identical partial binding is possible on the uninstantiated clause $C \lor [H \ A_x]^{\sigma}$ leading to $D_2 : C \lor [H \ A_x]^\sigma \lor [H = G]$ such that $(D_2)_{\sigma} \equiv D_1$.

- **Fac** This case is analogous to $Res$.

- **Triv, Dec, Func, FlexFlex, Leib, Equiv** These cases are all analogous to $Prim$. Note that $\equiv$ is a special symbol not available in the signature and thus the abstract literal must also have head $\equiv$. Therefore, all these rules must be applicable on the abstract level as well.

- **Subst** In this case the ground literal is of form $[X = A_x]^{\sigma}$. If $X_\alpha$ is of base type, the corresponding abstract literal obviously must have form $[y_a = A]^F$ for a variable $Y$. Then the assertion follows immediately. If $X_\alpha \rightarrow \beta$ is of functional type the corresponding abstract literal can principally also be of form (i) $[y_{\gamma \rightarrow \alpha \beta} = A]^F$ or (ii) $[C_\gamma = a \beta] B_\alpha = A]^F$, where $C$ is a $\lambda$-abstraction. Case (ii) can actually be excluded due to our special head-normal form convention of unification constraints. In case (ii) rule $Subst$ can not immediately be replayed at abstract level. But in a straightforward derivation which employs the rules $FlexRigid$ or $FlexFlex$ one can finally subsequently apply a substitution that is identical or more general than $[A_{a=x}]$. E.g., let the ground derivation be $C_\alpha : [X_{_0} = \beta] \lor [X_{_0} = \lambda Z \bullet p \rightarrow Z] \lor \ldots \lor [X_{_m} = \lambda Z \bullet p \rightarrow Z]^F \vdash Subst D_1 : [p]^{\beta F}$, and let $C : [\forall x_0 \ldots x_n \ a] [Y (a) = \lambda Z_\bullet p \rightarrow Z]^F$, such that $\sigma \equiv [\lambda Z \bullet X_{_0} \rightarrow \gamma; Y \ a]^{\neg}$. We consider the following derivation on the abstract level $(s_1, s_2)$, which is a Skolem term, and $H^1$ and $H^2$ are new free variables of appropriate type: $C \vdash FlexFlex [(Y a) b]^T \lor [(Y a) s = P s] F \lor FlexFlex [(Y a) b] T \lor [(Y a) s = P s] F \lor [P = \lambda \forall X \bullet p \rightarrow (H_1 \rightarrow V)]^F \vdash Subst [(Y a) b] T \lor [(Y a) s = p (H_1)]^{F \vdash FlexRigid} [(Y a) b] T \lor [(Y a) s = p (H_1)]^{F \vdash Subst} [y = \lambda \forall X \bullet p \rightarrow (H_1 \rightarrow U)]^{F \vdash Subst} [p (H_2 a b)] T \lor [p (H_2 a s) = p (H_1)]^{F \vdash Dec, Triv}$. 


\[ p \left( H^2 a b \right) \] 
\[ \lor \left( \left( H^2 a s \right) = \left( H^1 s \right) \right) \] 
\[ \vdash^{F_{\text{Flexible}}} \left[ p \left( H^2 a b \right) \right] \] 
\[ \lor \left[ \left( H^2 a s \right) = \left( H^1 s \right) \right] \] 
\[ \vdash^{H^1} \lambda x_{\alpha} X_{\alpha} \left[ \left( H^2 a b \right) \right] \] 
\[ \lor \left[ \left( H^2 a s \right) = \left( H^1 s \right) \right] \] 
\[ \vdash^{H^1} \lambda x_{\alpha} X_{\alpha} \left[ \left( H^2 a b \right) \right] \] 
\[ \lor \left[ \left( H^2 a s \right) = \left( H^1 s \right) \right] \] 
Thus, the subsequent instantiations computed on the abstract level in our example lead to clause \( D_2 \), i.e., the result of the substitution step on the ground level.

**FlexRigid** Whereas the abstract literal must have head \( \equiv \) it is probably a *Flexible*-constraint such that we cannot immediately replay the *FlexRigid* step. In this case we employ the rules *Flexible* and *Subst* in order to introduce the corresponding rigid head of the instantiated literal. Thereby the *FlexRigid* step performed on the instantiated level becomes possible on the abstract level as well. It is easy to verify that the resulting clause on the abstract level is more general than the instantiated counterpart.

\[ r \in \mathcal{NF} \quad \text{by clause normalisation lifting lemma 4.11.} \]
(2) The proof is by induction on the length of derivation \( \Delta \). The base case is trivial and in the induction step we first employ statement (1) and then the induction hypothesis. \( \square \)

### 4.4 Completeness

We now focus on the Henkin completeness proof for the calculus \( \mathcal{ER}_{\text{HC}} \). Towards this end we first prove a lemma stating that reduction to head-normal form is sufficient in calculus \( \mathcal{ER}_{\text{HC}} \) (note our special definition of head-normal form for unification constraints as described in Chapter 2.1). A second lemma then provides some important refutational properties of \( \mathcal{ER} \). Finally in theorem 4.16 we show that the set of propositions that cannot be refuted in calculus \( \mathcal{ER}_{\text{HC}} \) defines an abstract consistency property for Henkin models as defined in 3.6. The latter entails Henkin completeness for \( \mathcal{ER}_{\text{HC}} \) by theorem 3.29.

**Lemma 4.13 (Head-Normal Form).** Let \( \Phi \) be a set of clauses. If \( \Delta : \Phi_{\alpha n} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \) then \( \Delta^\tau : \Phi_{\alpha h} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \)

**Proof:** The proof is by induction on the length \( n \) of \( \Delta \) and the base case \( n \equiv 0 \) is trivial. In the step case \( (n > 0) \) we consider the first derivation step in \( \Phi_{\alpha n} \vdash^{r} \Phi^\tau \vdash \Box \). In case \( r \in \{ Rules, Faces \} \) the assertion follows immediately by induction hypotheses as both rules are applicable independently from the structure of the literals atoms we focus on. In case \( r \in \{ Prim \} \cup \mathcal{N} \) the heads of the atoms obviously play an important role. But note that the head symbols in the \( \beta \eta \)-normal form and the head-normal form coincide, such that rule \( r \) is applicable in the head-normal form case as well. Therefore, we again get the assertion by induction hypotheses. In case \( r \in \mathcal{NL} \) our special definition of head-normal form for unification constraints (which requires both sides of the constraint to be reduced to head-normal form) ensures the assertion with an analogous argument as in the previous case. (Note that the other direction of this lemma, which we do not need in this thesis, can be proven analogously.) \( \square \)

**Remark 4.14 (Head-Normal Form).** It is not essential for our calculi whether we keep our clauses and literals in head-normal form or \( \beta \eta \)-normal form. The reason why we have chosen reduction to head-normal form is that we expect that reduction to head-normal form is less expensive in practice (which has not been investigated yet). One can also choose \( \beta \eta \)-normal form. This variation will hardly influence any of the further discussions and theorems in this thesis.

**Lemma 4.15.** Let \( \Phi \) be a set of clauses and \( A, B \) be formulae. It holds:

1. If \( \Phi \vdash \left[ A \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \) and \( \Phi \vdash \left[ B \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \), then \( \Phi \vdash \left[ A \lor B \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \).
2. If \( \Phi \vdash \left[ A \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \) and \( \Phi \vdash \left[ A \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \), then \( \Phi \vdash \left[ A \equiv B \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \).

**Proof:** (1) Without loss of generality we assume that the pre-clauses \( \left[ A \right]^{T} \) and \( \left[ B \right]^{T} \) are variable-disjoint. We can replay derivation \( \Phi \vdash \left[ A \right]^{T} \vdash^{\mathcal{ER}_{\text{HC}}} \Box \) in context \( \Phi \vdash \left[ A \lor B \right]^{T} \), such that we
get for each clause $C \in \text{CNF}(\{B\}^T)$ $\Phi * [A \lor B]^T \vdash_{CR} [A]^T \lor C \vdash_{CR} C \lor E$ for a set of FlexFlex constraints $E$ containing no variables occurring in $[B]^T$ or $C$. By corollary 4.6 we know that each unifier $\sigma$ of $E$ can be derived in $IL\xi$ and thus we get for each unifier $\sigma$ of clause $C \lor E$ that $C \lor E \vdash_{IL\xi} C_{\sigma}$. Now the assertion follows by lemma 4.10 as $\Phi * [B]^T \vdash_{CR} \Box$ and as the domain of $\sigma$ contains none of the free variables in $C$, such that $C_{\sigma} \equiv C$ for each clause $C \in \text{CNF}(\{B\}^T)$.

(2) This statement (which is also well known from first-order resolution) can be proven by a tedious but straightforward computation which we only roughly sketch here. From the two assumptions we get by lemma 4.10 that (i) $\Phi * \text{CNF}(\{A\}^T) * \text{CNF}(\{B\}^T) \vdash_{CR} \Box$ and (ii) $\Phi * \text{CNF}(\{A\}^T) * \text{CNF}(\{B\}^T) \vdash_{CR} \Box$. The idea now is to apply exhaustive clause normalisation to the clause $[A \equiv B]^T$ and then to show that (iii) $\Phi * \text{CNF}(\{A \equiv B\}^T)$ $\vdash_{CR} \Box$. Note that $\text{CNF}([A \equiv B]^T) = (C \lor D|C \in \text{CNF}([A]^T) \text{ and } D \in \text{CNF}([B]^T) \text{ for } \alpha \in \{T, F\})$. Thus, the task is to show that (iii) is a consequence of (i) and (ii), which is possible, e.g., by simultaneous induction on the structure of $A$ and $B$.

\[ \Box \]

**Theorem 4.16 (Completeness of $\mathcal{CE}$).** The calculus $\mathcal{CE}$ is complete with respect to Henkin models.

**Proof:** We adapt the proofs given in [BK98a] and [Koh94b] which in turn are based on the ideas of [And71].

Let $I_3$ be the set of $\Sigma$-sentences which cannot be refuted by the calculus $\mathcal{CE}$ ($I_3 := \{ \Phi \subseteq c \text{uff}_{\Sigma}(\Sigma) | \Phi_d \vdash_{CR} \Box \}$), then we show that $I_3$ is a saturated abstract consistency class for Henkin models 3.6 which entails Henkin completeness for $\mathcal{CE}$ by theorem 3.29.

In particular we have to verify that $I_3$ ensures the abstract consistency properties $\nabla_c$, $\nabla_{\omega}$, $\nabla_{\beta}$, $\nabla_{\alpha}$, $\nabla_{\beta}$, $\nabla_{\beta}$, $\nabla_{\alpha}$, $\nabla_{\alpha}$. Furthermore we have to show that $I_3$ is saturated.

$\nabla_c$

Suppose that $A, \neg A \in \Phi$, where $A \in c \text{uff}_{\Sigma}(\Sigma)$. Since $A$ is atomic we have $\Phi_d * [A]^T * [-\neg A]^T \vdash_{CN} \Phi_d * [A]^T * [A]^T$ and hence we can derive $\Box$ with $Res$ and $Triv$. This contradicts our assumption.

In all of the remaining cases, we show the contrapositives, e.g., in the next case we prove, that for all $\Phi \in I_3$, if $\Phi \nvdash A \nvdash \not{\Phi} \in I_3$, then $\Phi \nvdash \neg A \nvdash \not{\Phi}$, which entails the assertion.

$\nabla_\omega$

Let us assume that $\Phi_d * [\neg \neg A]^T * [A]^T \vdash_{CR} \Box$. We immediately get the assertion since $[\neg \neg A]^T \vdash_{CN} [A]^T$.

$\nabla_\beta$

If $\Phi_d * [A]^T * [A \land \alpha]^T \vdash_{CR} \Box$, then we get that $\Phi_d * [A]^T \vdash_{CR} \Box$ by lemma 4.13 (note that $A$ is assumed to be in head-normal form).

$\nabla_{\alpha}$

If $\Phi_d * [\{A \lor B\}]^T * [A]^T \vdash_{CR} \Box$ and $\Phi_d * [\{A \lor B\}]^T * [B]^T \vdash_{CR} \Box$, then $\Phi_d * [A \lor B]^T \vdash_{CR} \Box$ by lemma 4.15(1).

$\nabla_{\alpha}$

Analogously to $\nabla_\omega$, as $[\neg (A \lor B)]^T \vdash_{CN} [\neg A]^T$ and $[\neg (A \lor B)]^T \vdash_{CN} [\neg B]^T$.

$\nabla_{\alpha}$

Let $\Phi_d * [\Pi^X F]^T * [F A]^T \vdash_{CR} \Box$ for each closed formula $A$. By lifting lemma 4.12 we get that $\Phi_d * [\Pi^X F]^T * [F X]^T \vdash_{CR} \Box$ for a new variable $X_\alpha$ and thus obviously $\Phi_d * [\Pi^X F]^T \vdash_{CR} \Box$.

$\nabla_{\alpha}$

Let us assume that $\Phi_d * [\neg (\Pi F)]^T * [\neg (A \lor B)]^T \vdash_{CR} \Box$. Note that $\neg (\Pi F)$ is a closed formula and furthermore that $w$ does not occur in $\Phi_d * [\neg (\Pi F)]^T$. We get the assertion as $[\neg (\Pi F)]^T \vdash_{CN} [\neg (A \lor B)]^T$ for a Skolem constant $w'$, which is just a renaming of $w$ above.

$\nabla_{\alpha}$

We show that if $\Phi_d * [\neg (A \lor B)]^T * [\neg A]^T \vdash_{CR} \Box$ and $\Phi_d * [\neg (A \lor B)]^T * [B]^T \vdash_{CR} \Box$, then $\Phi_d * [\neg (A \lor B)]^T \vdash_{CR} \Box$. Note that $\Phi_d * [\neg (A \lor B)]^T \equiv \Phi_d * [\Pi (A \lor \alpha \rightarrow \neg P \lor P A \lor P B)]^T \vdash_{CN} \Phi_d * [r A]^T \equiv [r B]^T$, where $r_{\alpha \rightarrow \omega}$ is a new Skolem constant. Now consider the following derivations.

\[ \begin{array}{|c|c|} \hline \text{Res} & [r A \equiv r B]^T \\ \text{Dec, Triv} & [A \equiv B]^T \\ \text{Equiv} & [A \equiv B]^T \\ \hline \end{array} \]
Hence $\Phi_{cl} \vdash \neg (A = B)^T \vdash_{\varphi_{\alpha}} \Phi_{cl} \vdash \neg (A = B)^T \vdash [A \iff B]^F$ and we get the conclusion as a consequence of lemma 4.15(2).

Thus we show that if $\Phi_{cl} \vdash \neg (F w \equiv G)^T \vdash_{\varphi_{\alpha}} \Phi_{cl} \vdash \neg (F w \equiv G)^T \vdash [F \iff G]^T$ then $\Phi_{cl} \vdash \neg (F w \equiv G)^T \vdash [F \iff G]^T \vdash \varphi_{\alpha}$. Note that $\Phi_{cl} \vdash \neg (F \vdash G)^T \vdash [F \iff G]^T \vdash \varphi_{\alpha}$. To show that $\Phi_{cl} \vdash \neg (F \vdash G)^T \vdash \varphi_{\alpha}$ we need to show that $\Phi_{cl} \vdash \neg (F \vdash G)^T \vdash \varphi_{\alpha} \vdash [F \iff G]^T \vdash \varphi_{\alpha} \vdash [F \iff G]^T \vdash \varphi_{\alpha}$. Now consider the following derivations:

\[
\begin{array}{c}
[r F]^T \vdash [r G]^F \\
\vdash [r F \equiv r G]^F \\
\vdash [F \equiv G]^F \\
\vdash [t (F s)]^F \\
\vdash [t (G s)]^F
\end{array}
\]

Here again $s_\alpha$ and $t_\beta \alpha$ are new Skolem constants. Hence $\Phi_{cl} \vdash [r F]^T \vdash [r G]^F \vdash_{\varphi_{\alpha}} \Phi_{cl} \vdash [r F]^T \vdash [r G]^F \vdash [t (F s)]^T \vdash [t (G s)]^F$. Now the conclusion follows from the assumption that $s, t$ and $r$ are only renamings of the Skolem symbols $w, p$ and $q$ and as all do not occur in $\Phi_{cl}$.

To see that $I_2$ is saturated let $A \in \text{cuff}_{\Sigma}(\varphi_{\alpha})$ and $\Phi \subseteq \text{cuff}_{\Sigma}(\varphi_{\alpha})$ with $\Phi \vdash \varphi_{\alpha}$. We have to show that $\Phi_{cl} \vdash A \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$. For that suppose $\Phi_{cl} \not\vdash A \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$. For that suppose $\Phi_{cl} \not\vdash A \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$. By lemma 4.15(1) we get that $\Phi_{cl} \vdash A \lor \neg A \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$, and hence, since $A \lor \neg A$ is a tautology, it must be the case that $\Phi_{cl} \vdash \neg \varphi_{\alpha}$, which contradicts our assumption.

Remark 4.17 (Eager Unification). In contrast to Huet [Hue72, Hue73a] eager unification is essential within our approach. This is illustrated by the arguments for $\forall_q$ and $\exists_q$ in the completeness proof 4.16 as well as many of the examples presented in Chapter 8.

Conjecture 4.18 (Restricted rule Leib). Even though rule Leib is employed to terms of arbitrary types in the completeness proof (see 4.16(\forall_q))) we conjecture, that this rule can be restricted to unification constraints between terms of primitive type:

\[
\frac{C \lor [M_{\alpha} - N_{\alpha}]^F}{C \lor [\forall P_{\alpha} \alpha \Rightarrow P N]^F} \text{ Leib}
\]

Unfortunately a formal proof for this conjecture, whose validity would be of practical importance, is still missing. In order to prove this conjecture, it would suffice to prove that if $\Phi_{cl} \vdash [A \equiv_{\alpha \rightarrow \beta} B]^T \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$ then $\Phi_{cl} \vdash [A s \equiv_{\alpha \rightarrow \beta} B s]^T \vdash_{\varphi_{\alpha}} \varphi_{\alpha}$ for any new Skolem term $s_{\alpha}$.

Note that $\forall X_{\alpha}, A \equiv_{\alpha \rightarrow \beta} B X \vdash \varphi_{\alpha} \vdash [A s \equiv_{\alpha \rightarrow \beta} B s]^F$. Thus, this lemma expresses one direction of the functional extensionality property formulated with Leibniz equality and with this lemma we could reduce the applications of rule Leib in all three calculi ($\varphi_{\alpha}, \varphi_{\beta}$ and $\varphi_{\beta}$) discussed in this thesis and employ the restricted rule Leib instead. This result would be of practical importance as it allows us to restrict the search space. In the extensional RUE-Resolution approach rule Leib may even become redundant (c.f. Remark 8.1).

4.5 Theorem Equivalence

With respect to the theoretical goal of proving Henkin completeness it does not make a difference whether exhaustive clause normalisation derivations are grouped together into one rule $Cif$ like in calculi $\varphi_{\alpha}$ and $\varphi_{\beta}$, or if the single clause normalisation rules are lifted as single inference rules to calculus level like in $\varphi_{\beta}$. But note that there is a practical motivation for the ungrouping of $\varphi_{\beta}$-derivation as well: Calculus $\varphi_{\beta}$ allows to avoid redundant applications of identical unification derivations to all proper clauses belonging to the same abstract clause. For instance, it may be
4.5. THEOREM EQUIVALENCE

more appropriate first to apply unification rules to a non-proper clause and then to apply clause normalisation rule to the result than the other way around.

Remark 4.19 (Convention).

1. Whereas the \(\CNF\) and the \(\mathcal{UN}\) rules aside from \textit{Subst} do not directly influence each others applicability, they may indirectly influence each others applicability via Skolemisation. More precisely if \(r_2\) is rule \(\Pi^F\) and \(r_1\) is rule \textit{Func} the arity of the Skolem term that is introduced in \textit{Func} increases its arity when switching \(r_1\) and \(r_2\). We have already pointed out in Remark 4.1 that we can ignore this fact for the remainder of this thesis.

2. Another important and simplifying convention in all proof transformations in this thesis is to consider proper proof trees instead of proof graphs, such that each derived clause in a derivation is used exactly once as a premise clause in the subsequent derivation steps.

Lemma 4.20 (Derivability of Proper Clauses). For each proper clause \(C\) and clause set \(\Phi\), such that \(\Delta_1 \vdash \Phi \vdash_{\mathcal{ER}_P} C\), there is a \(\mathcal{ER}_F\)-derivation \(\Delta_2 \vdash \Phi \vdash_{\mathcal{ER}_P} C\).

Proof: The proof idea is to show that the single, distributed clause normalisation steps can be grouped in exhaustive \(\CNF\)-chains, which can then be replaced by rule \textit{Con}. The proof is by induction\(^5\) on the length \(n\) of derivation \(\Delta_1\) and the base case \((n = 0)\) is trivial. In the induction step \((n > 0)\) we consider the first step in derivation \(\Delta_1 \vdash \Phi \vdash_{\mathcal{ER}_P} C\), where \(r_1 \in \mathcal{ER}_F\). We proceed by examining all possibilities for \(r_1\):

- \(r_1 \in \{\text{Res, Fac, Prim}\}\) As these rules operate on proper clauses only, we have on the one hand that \(r_1\) is also a proper \(\mathcal{ER}_F\) derivation step. On the other hand we know by induction hypothesis that there is a proper \(\mathcal{ER}_F\) derivation \(\Phi \vdash D_1 \vdash_{\mathcal{ER}_P} C\). Thus \(\Phi \vdash_{\mathcal{ER}_F} C\).

- \(r_1 \in \mathcal{UN} \cup \text{Leib, Equiv}\) Analogous to above the first step is also a proper \(\mathcal{ER}_F\) derivation step, such that the assertion follows immediately by induction hypotheses.

\(r_1 \in \CNF\) In this case we look for the first step in derivation \(\Delta_1\) that employs a non-\(\CNF\)-rule. Without loss of generality let us assume that this happens in step \(n \geq 1\). Then \(\Delta_1\) has form \(\Delta_1 \vdash \Phi \vdash^{r_1} \Phi \vdash^{r_2} \ldots \vdash^{r_{n-1}} \Phi \vdash D_1 \vdots D_{n-1} \vdash^{r_n} \Phi \vdash D_1 \vdots D_n \vdash_{\mathcal{ER}_P} C\), such that \(r_j \in \CNF\) for \(1 < j < n\). Without loss of generality let us assume that each of the steps \(r_j\) employs the newly generated clause from the previous step as premise clause (we can reorder the derivation steps in \(\Delta_1\) without any influence to the particularly derived clauses such that this assumption is met).

In case \(r_n \in \{\text{Res, Fac, Prim}\}\), \(D_n\) must be a proper clause, such that we can obviously replace the initial \(n - 1\) derivation steps in \(\Delta_1\) by a single application of rule \textit{Con} in calculus \(\mathcal{ER}_F\). Now we again employ the convention that we consider proper proof trees instead of proof graphs in our formal proofs such that each derived clause in a derivation is used exactly once as a premise clause in one of the following derivation steps. Consequently \(\Delta_1 \vdash \Phi \vdash_{\mathcal{CNF}} \Phi \vdash D_n \vdash_{\mathcal{ER}_P} C\) and the assertion follows by induction hypotheses.

As \(\square\) is also a proper clause we immediately get the following corollary.

\(^5\)Note that with convention 4.19, which says that we consider proof trees rather than proof graphs, this induction proof (and many others in this thesis) should actually be carried out by induction on the structure of the proof tree or by induction on the depth of the proof tree. But note that the derivations described by proof trees can be linearised in a unique way, such that the length of the linearised proof trees gives us an well-founded ordering as well.
Corollary 4.21 (Theorem Equivalence of \( \mathcal{ER}_f \) and \( \mathcal{ER}_f \)). The calculi \( \mathcal{ER}_f \) and \( \mathcal{ER}_f \) are theorem equivalent, i.e., \( \Phi \vdash \mathcal{ER}_f \square \), iff \( \Phi \vdash \mathcal{ER}_f \square \).

It is no longer possible to delay all (pre-)unification rules, but our claim is that the additional rule \textit{FlexFlex} can still be delayed until the end of a refutation. This result would give us that rule \textit{FlexFlex} is not needed at all, as \( \square \) is defined modulo \textit{flex-flex} constraints.

Conjecture 4.22 (Theorem Equivalence of \( \mathcal{ER} \) and \( \mathcal{ER}_f \) (or \( \mathcal{ER}_f \)).

The calculi \( \mathcal{ER} \) and \( \mathcal{ER}_f \) (or \( \mathcal{ER}_f \)) are theorem equivalent.

The above conjecture is motivated by the observation that none of the challenging examples discussed in this thesis or in any of the papers [BK98a, BK98b, Ben97, BK97b] requires an application of rule \textit{FlexFlex}. Furthermore, our case study with the extensional higher-order resolution prover LEO [BK98b] showed that none of the examples from the MIZAR-articles \textit{Boolean and Basic Properties of Sets} [TS89, Byl89], which are very interesting with respect to extensionality principles, requires the application of the \textit{FlexFlex} rule.
Chapter 5

Extensional Higher-Order Paramodulation: $\mathcal{EP}$

In this chapter we shall try to adapt traditional first-order paramodulation to higher-order logic. The paramodulation rule is not sufficient to ensure Henkin completeness as we run into problems with the Boolean and functional extensionality principles of primitive equality, hence additional extensionality rules are needed.

5.1 A Naive and Incomplete Adaptation of Paramodulation

Figure 5.1 shows the traditional paramodulation rule and introduces a more elegant higher-order reformulation.\(^1\) Note that we assume the symmetric case for both rules.

The paramodulation rule $Para$ replaces any subterm $T_\beta$ (which contains no variable that is bound outside $T$) by $R_\beta$ provided that $T$ and $L$ are uninifiable. Analogously to higher-order resolution and factorisation, unification has to be delayed and thus a unification constraint $[T = L]^F$ is added to the resulting clause. Unfortunately, we thereby introduce with each application of rule $Para$ twice as many (because of symmetric application of equations) new clauses into the search space as there are subterms $T_\alpha$ of type $\alpha$ in the given term $A$. To illustrate applications of rule $Para$ we consider the following two unit clauses $C_1 : [p \ (f \ (f \ a))]^F$ and $C_2 : [f = h]^F$, where $p_{\rightarrow \alpha}, f_{\rightarrow \alpha}, h_{\rightarrow \alpha}$, and $\alpha$ are constants. By application of paramodulation in left to right direction we obtain the following two clauses: $C_3 : [p \ (h \ (f \ a))]^F \lor [f = f]^F$ and $C_4 : [p \ (f \ (h \ a))]^F \lor [f = f]^F$.

Our trivial example the generated unification constraints can be immediately eliminated in eager unification attempt with rule $Tri$.

<table>
<thead>
<tr>
<th>Paramodulation</th>
<th>(defined for proper clauses only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[A[T_\beta]]^\alpha \lor C \quad [L = ^\beta R]^F \lor D$</td>
<td>$[A[R]]^\alpha \lor C \lor D \lor [T = ^\beta L]^F$ $Para$</td>
</tr>
<tr>
<td>$[A]^\alpha \lor C \quad [L = ^\beta R]^T \lor D$</td>
<td>$[P_{\rightarrow \alpha} R]^\alpha \lor C \lor D \lor [A = \alpha P_{\rightarrow \alpha} L]^T$ $Para'$</td>
</tr>
</tbody>
</table>

In $Para$ subterm $T$ of $A$ must not contain free variables that are bound outside

\(^1\) This rule was suggested by Michael Kohlhase.
Rule \textit{Para}' combines the expressiveness of our higher-order language with the power of higher-order (pre-)unification and avoids the introduction of too many clauses into the search space. Instead only one new clause with a new flexible literal head is created. The idea of this rule is very simple: it describes the respective resolution step between the left premise clause and the clause that corresponds to the right premise clause when replacing the positive primitive equation by a (normalised) positive Leibniz equation (the modified right clause would have form \([P L]^T \vee [P R]^T \vee D \) — or with an additional primitive substitution step \([P L]^T \vee [P R]^F \vee D \). When applying rule \textit{Para}' to the clauses \(C_1\) and \(C_2\) from left to right, we introduce only one new clause: \(C_5 : [P h]^T \vee [p (f (f a)) = P f]^F\). By eager pre-unification we can generate the following four instantiations for \(P\) and propagate this partial solutions with rule \textit{Subst} to the literal \([P h]^T\):

\[
\begin{align*}
1) & \ P \leftarrow \lambda z_{\rightarrow a} \cdot p (f (f a)) & 2) & \ P \leftarrow \lambda z_{\rightarrow a} \cdot p (Z (f a)) \\
3) & \ P \leftarrow \lambda z_{\rightarrow a} \cdot p (f (Z a)) & 4) & \ P \leftarrow \lambda z_{\rightarrow a} \cdot p (Z (Z a))
\end{align*}
\]

The pure imitation solution (1) introduces clause \(C_1\) again and is thus redundant.\(^2\) By instantiating solutions (2) and (3) we obtain exactly the clauses \(C_3\) and \(C_4\) (clearly without the trivial pairs), which are identical to the result of the applications of the traditional paramodulation rule. Solution (4) is the most interesting one as it encodes the simultaneous application of \([f = h]^T\) to both subterms \(f\) in \([p (f (f a))]^F\). As we have illustrated, the only clause resulting from the application of rule \textit{Para}' encodes all the possible traditional paramodulation steps together with all simultaneous applications and even the original clause itself. The only problem for practical applications is, that we introduce a new flexible literal head with each application of \textit{Para}', such that the primitive substitution rule becomes applicable. As this seems to be rather useless, a good heuristic in practice might be to avoid primitive substitution steps on flexible heads generated by rule \textit{Para}'.

\textbf{Remark 5.1 (Reflexivity Resolution).} Note that a reflexivity resolution rule is already naturally integrated into calculus \(\mathcal{CR}\), as it is derivable with the help of the unification rules. The unification rules already operate on negative primitive equation literals, i.e., unification constraints, like \([f X = Q q]^F\), such that each negative equation between two unifiable terms can be refuted by them.

We want to point out, that the encoding of unification constraints as negated literals is essential here, whereas it is not of importance for calculus \(\mathcal{CR}\). If we would not encode unification constraints as negative primitive equation literals and would differentiate between both concepts, then we obviously have to face the reflexivity resolution problem.

\textbf{Remark 5.2 (Paramodulation into Unification Constraints).} It turns out that paramodulation into unification constraints is not necessary, and in fact we can show its derivability:

\[
\frac{C_1 : C \lor [A[T_a]] = B^F \quad C_2 : [L =^a R]^T \lor D}{C_3 : C \lor D \lor [A[R_a]] = B^F \lor [T =^a L]^F} \quad \text{Para}
\]

Each such step can be replaced by the following derivation (\(p\) is a Skolem constant):

\[
\begin{align*}
\text{Leib}(C_1), \text{CNF} : & \quad \text{Para}(D_1, D_2) : \quad \text{Res}(D_3, D_3, F \text{ac}, \text{Triv}) : \quad \text{Dec}(D_5, \text{Triv}) : \\
D_1 : C \lor [p A[T_a]]^T & \quad D_2 : C \lor [p B]^F & \quad D_3 : C \lor D \lor [p A[R_a]]^T \lor [T =^a L]^F & \quad D_5 : C \lor D \lor [A[R_a]] = B^F \lor [T =^a L]^F
\end{align*}
\]

On the one hand paramodulation into unification constraints may shorten proofs and in some examples this seems to be very appropriate, but on the other hand such an approach may be hard to guide in practice.

\(^2\)On the other hand this possibly leads to interesting heuristics in practice: When applying rule \textit{Para}' with a right premise clause which is a unit equation, we can remove the left premise clause from the search space as this clause gets encoded into the result of the paramodulation step itself.
5.1. A NAIVE AND INCOMPLETE ADAPTATION OF PARAMODULATION

Remark 5.3 (Functional Extensionality and rule Para'). It has been claimed by an unknown referee of [Ben98], that rule Para' already captures full extensionality, such that rule Para should be preferred over Para. Example $E_1^{Para'} (\forall X, \exists g \cdot X = g \Rightarrow g \in J)$ discussed in Section 8.6 demonstrates that this is not true: even rule Para' requires additional extensionality rules to ensure full functional extensionality.

Definition 5.4 ($\mathcal{EP}_{naive}$). The calculus $\mathcal{EP}_{naive}$ consists of the rules of calculus $\mathcal{ER}$ (see Figure 4.2) enriched by the paramodulation rule Para (see Figure 5.1). We assume that the result of each rule application is transformed into head-normal form. A set of formulae $\Phi$ of refutable in calculus $\mathcal{EP}_{naive}$, if there is a derivation $\Delta : \Phi \vdash \Box$, where $\Phi := \{[F][T] | F \in \Phi\}$ is the set of clauses obtained from $\Phi$ by simple pre-classification. We want to point out that primitive equations are not expanded by Leibniz definition.

Next, we discuss soundness of the extended calculus $\mathcal{EP}_{naive}$ and show by a counterexample that $\mathcal{EP}_{naive}$ is not complete with respect to Henkin semantics.

Theorem 5.5 (Soundness of naive higher-order paramodulation). The calculus $\mathcal{EP}_{naive}$ is sound with respect to Henkin semantics.

Proof: Soundness of the traditional paramodulation rule Para is obvious (note that we avoid the replacement of subterms $T$ with free variables that are bound outside): given a standard model $M$ for the two premise clauses, then one can easily see that the paramodulant is also valid in $M$, as either the unification constraint evaluates to $\bot$ and we are done or it evaluates to $T$ giving rise to the validity of literal $[A[R]]^a$, in case $[A[T]]^a$ guaranteed the validity of the first premise clause and $[L = R]^T$ guaranteed the validity of the second premise clause (all other cases are trivial).

The proof of soundness for the new rule Para' is analogous, even though this rules looks more complicated: we consider all possible variable assignments $\varphi$ which map variable $P_{a\rightarrow o}$ to a function in the domain $D_{a\rightarrow o}$ and employ an analogous argumentation like above.

Theorem 5.6 (Completeness of $\mathcal{EP}_{naive}$). The calculus $\mathcal{EP}_{naive}$ is incomplete with respect to Henkin semantics.

Proof: The assertion is proven by the following counterexamples to the assumption of Henkin completeness of calculus $\mathcal{EP}_{naive}:

Example 5.7 (Incompleteness of $\mathcal{EP}_{naive}$).

$E_1^{Para'} \neg \exists X_{a\rightarrow o} (X = \neg X)$

This formula expresses, that the negation operator is fix-point free, which is obviously the case in Henkin semantics. Our calculus is not able to find a proof as clause normalisation of the negated assertion leads to the single clause

$C_1: [a = \neg a]^T$

where $a_o$ is a new Skolem constant. The only rule that is applicable is self-paramodulation on positions 1, 2 and 0 leading to the following clauses:

Para$C_1, C_1$ at 1: $C_2: [a = \neg a]^T \lor [-a = a]^T$ $C_3: [-a = \neg a]^T \lor [a = a]^T$

Para$C_1, C_1$ at 2: $C_4: [a = \neg a]^T \lor [-a = a]^T$ $C_5: [a = a]^T \lor [a = a]^T$

Para$C_1, C_1$ at 0: $C_6: [a]^T \lor [-a = (a = \neg a)]^T$ $C_7: [a]^T \lor [a = (a = \neg a)]^T$

Case distinction on the possible denotations $\{T, F\}$ for $a$ shows that all these clauses are tautologies. Thus no refutation is possible in $\mathcal{EP}_{naive}$.

$E_2^{Para'} \neg \exists X_{a\rightarrow o} \forall P_{a\rightarrow o} \exists X_{a\rightarrow o} [G, X \Rightarrow \alpha] P$

This is a simple formulation of Cantor's theorem stating that there exists no surjective function from the set of individuals into the set of sets of individuals. Clause normalisation results in

$C_1: [G, X \Rightarrow \alpha] P]^T$
where $p_{i \rightarrow o}$ is a Skolem constant. Similar to the case above, a refutation in $\mathcal{E}_{\text{naive}}$ is not possible.

$E^\text{Para}_3 \vdash \exists M_{o \rightarrow o}, M \neq \emptyset$, where $\emptyset_{o \rightarrow o}$ is defined as $\lambda X_{o} \cdot F_{o}$. This formula expresses, that there exists a set of truth values that is not empty. As the set of truth values is fixed in Henkin semantics and has exactly two elements, this assertion is obviously valid. Clause normalisation results in

$$C_1 : [M = \lambda X_{o} \cdot \perp]^T$$

where $M_{o \rightarrow o}$ is a free variable. Analogous to $E^\text{Para}_3$ no refutation is possible.

$E^\text{Para}_4 \vdash \exists M_{o \rightarrow o}, M = \overline{M}$, where the set complement operator $\overline{X_{o \rightarrow o}}$ is defined by $\lambda S_{o \rightarrow o} \cdot \lambda X_{o} \cdot \overline{(X \in S)}$ and $\in_{o \rightarrow o}$ as function (predicate) application $\lambda X_{o} \cdot \lambda S_{o \rightarrow o} \cdot S \cdot X$.

This formula expresses, that there is no set $M$ of truth values in Henkin semantics, such that $M$ and its complement set $\overline{M}$ are identical. Clause normalisation leads to

$$C_1 : [m = \lambda X_{o} \cdot \lnot (m \cdot X)]^T$$

where $m_{o \rightarrow o}$ is a fresh Skolem constant for $M$. In contrast to $E^\text{Para}_3$ there is no free variable. Again no refutation is possible.

$E^\text{Para}_5 \vdash \forall P_{i \rightarrow o}, q_{i \rightarrow o}, r_{i \rightarrow o}, X_{i} \cdot (P_{i \rightarrow o} \cdot q_{i \rightarrow o} = P_{i \rightarrow o} \cdot r_{i \rightarrow o})$, then ($\forall X_{i} \cdot q_{i} \cdot X \neq \lambda Z_{i} \cdot \lnot (r \cdot X \cdot Z)$). The first equation expresses, that all relations of type $i \rightarrow i \rightarrow i \rightarrow o$ that can be defined based upon relation $q$ or $r$, are equal (which in fact means that $q$ and $r$ must be equal). The second equation expresses that $q$ is the complement set of $r$, but uses an artificially complicated form (we employed the functional extensionality property once to $q \neq \lambda X_{i} \cdot \lambda Z_{i} \cdot \lnot (r \cdot V \cdot Z)$). This assertion is again valid, as in Henkin semantics the set of truth values contains exactly two elements. Our problem normalises to

$$C_1 : [P \cdot q =_{i \rightarrow o} =_{i \rightarrow o} P \cdot r]^T \quad C_2 : [p \cdot X =_{i \rightarrow o} \lambda Z_{i} \cdot \lnot (r \cdot X \cdot Z)]^T$$

where $P_{i \rightarrow o}, q_{i \rightarrow o}, r_{i \rightarrow o}, X_{i}$ are free variables and $q_{i \rightarrow o}, r_{i \rightarrow o}, X_{i}$ are function constants. Note that apart from self-paramodulation no rule is applicable. Especially there is no paramodulation step possible between the clauses $C_1$ and $C_2$, as there is no subterm of type $i \rightarrow o$ in $C_1$ and no subterm of type $i \rightarrow i \rightarrow i \rightarrow o$ in $C_2$, such that a type conform term rewriting becomes possible.

The overall problem in the first four examples is, that our calculus provides no mechanism to detect positive equations with an implicitly embedded contradiction. For example the clause $[a_{o} =_{o} \lnot a_{o}]^T$ is implicitly contradictory with respect to the Boolean extensionality principle. Examples $E^\text{Para}_2$ and $E^\text{Para}_5\text{'}$ show, that also functional extensionality is involved as the implicit contradiction follows here only with respect to both extensionality principles. The general problem is that in higher-order logic (when considering Henkin semantics as we assume weaker notions like discussed in Chapter 2.1) infinitely many semi-antial domains contain a fix-point free function, such as the negation operator or the set complement operator on a non-empty domain (such as the set of truth values or the domain of all functions from truth values to truth values, etc.).

In example $E^\text{Para}_5\text{'}$ none of the positive equational literals is unsatisfiable taken alone, but it is contradictory in connection with the other ones. Here again a refutation is not possible in calculus $\mathcal{E}_{\text{naive}}$, as we would have to employ the functional extensionality principles to the positive equation in order to obtain a corresponding equation of appropriate type, such that the paramodulation rule becomes applicable.

**Remark 5.8 (Fix-point free functions).** Given a predicate type $\alpha := \alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow o$ ($n \geq 0$). The semantical domain $\text{domain}_{\alpha \rightarrow o}$ contains a fix-point free function provided that it is not empty (which must be the case in Henkin Semantics, as at least the identity function, i.e.,
the evaluation of $\lambda P_\alpha. P_\alpha$, must be an element of this domain). Note, that these fix-point free functions map predicates to predicates. An example in domain $(\alpha \rightarrow \alpha)$ is the negation operator. And in domain $(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$ one can choose the set complement operator defined with the help of the negation operator: $\lambda S_{(\alpha \rightarrow \alpha)} \lambda F_{(\alpha \rightarrow \alpha)} \neg (S F)$. It is easy to see, that analogously to the latter example one can easily construct a fix-point free function in any of the domains domain $(\alpha \rightarrow \alpha)$.

These fix-point free functions cause the problems with positive equations in higher-order logic: Let term $T_{\alpha \rightarrow \alpha}$ be an $\lambda$-expression that denotes such a fix-point free function in domain $D_{\alpha \rightarrow \alpha}$ (it has been illustrated above how to construct such $\lambda$-expressions). Then the positive equation $T X_\alpha = X_\alpha$ is obviously unsatisfiable. Note that such single contradictory equations do not occur in first-order logic.

5.2 Positive Extensionality Rules

Before going further into the investigation of the extensionality problem with positive equational literals, let us first reconsider the analogous problem for negative equations. For example, $[a_\alpha = \neg a]^E$ is a negative equation literal, which is implicitly contradictory with respect to Boolean extensionality. Just as for reflexivity resolution (cf. 5.1) our calculus already provides a solution to this problem as it interprets negative equational literals automatically as unification constraints and offers an appropriate extensionality treatment by the extensionality rules $Leib$, $Func$ and $Equiv$.

Remark 5.9 (Extensionality for Negative Primitive Equations). Our calculi encode unification constraints as negative primitive literals and do not differentiate between unification problems and negative primitive equations. Therefore an additional extensionality treatment for negative primitive equations is not needed as the unification algorithm has already been extended by the special extensionality rules $Leib$, $Func$ and $Equiv$. When differentiating between negative primitive equations and unification constraints, the unit clause $[a_\alpha = \neg a]^E$ is not refutable in the resulting approach, and we would need an additional extensionality treatment for negative primitive equations.

In contrast to this pleasant result, we do have to face the lack of extensionality principles in the case of positive equational literals. And, as we have seen in the examples above, our calculus so far does not provide a solution to it. What we need is a way to test the inequality of the two sides of a positive equation with respect to the extensionality principles and also with respect to the knowledge provided by the other clauses in the search space (see for example $E^E_{a=\alpha}$). This suggests to introduce analogous extensionality rules to $Func$, $Equiv$, and $Leib$, that operate on positive equations. To anticipate the results of the analysis of Henkin completeness, it turns out that a positive counterpart to rule $Leib$ is not needed, and thus our new extensionality for primitive positive equations are $Func^+$ and $Equiv^+$ as given in Figure 5.2.

Remark 5.10 (Rule $Leib^+$). A positive counterpart for rule $Leib$ would have the following form:

$$\frac{C \lor [M_\alpha = N_\alpha]^P \quad \alpha \in \{o, e\}}{C \lor [\forall \alpha \rightarrow \alpha \quad P \Rightarrow P N]^P \quad Leib^+}$$

As this rule is not needed in the completeness proof below, we can conclude that $Leib^+$ is admissible.

5.3 Basic Definitions

We further extend our extensional higher-order paramodulation approach by adding rule $FlexFlex$ as in Chapter 4, and lift the single clause normalisation rules to calculus level, instead of grouping exhaustive clause normalisation derivations together in rule $Cnf$.

The definitions for clause normalisation calculus ($CNF$ (cf. 4.2), $UNI$ and $UNI'$ (cf. 4.4) need not to be modified and the lemmata 4.3 (Soundness of $CNF$), 4.5, and 4.6 (properties of higher-order unification) will be employed in this section again.
Definition 5.11 (Extensional Higher-Order Paramodulation).

$\mathfrak{EP}$ The calculus $\mathfrak{EP}$ consists of the rules of calculus $\mathfrak{ER}$ (see Figure 4.2) enriched by the paramodulation rule $Para$ in Figure 5.1 and the positive extensionality rules displayed in Figure 5.2, i.e., $\mathfrak{EP} := \mathfrak{ER} \cup \{Para, \text{Equiv'}, \text{Func'}\}$.

$\mathfrak{EP}_f$ The extension $\mathfrak{EP}_f$ of calculus $\mathfrak{EP}$, that employs full higher-order unification instead of higher-order pre-unification, is defined as $\mathfrak{EP}_f := \mathfrak{EP} \cup \{\text{Flex} \}.$

$\mathfrak{EP}_c$. The calculus $\mathfrak{EP}_c$, that employs stepwise instead of exhaustive clause normalisation, is defined by $\mathfrak{EP}_c := (\mathfrak{EP} \setminus \{\text{CNf}\}) \cup \text{CNF}.$

For all calculi we assume that the result of each rule application is transformed into head-normal form. A set of formulae $\Phi$ is refutable in calculus $\mathfrak{EP}$, iff there is a derivation $\Delta : \Phi \vdash \mathfrak{EP} \Box$, where $\Phi_\Delta := \{[F_{\Phi}]^T | F \in \Phi\}$ is the set obtained from $\Phi$ by simple pre-clausification. Unification literals are still only accessible to the unification rules.

Theorem 5.12 (Soundness of Extensional Higher-Order Paramodulation).
The calculi $\mathfrak{EP}, \mathfrak{EP}_f$ and $\mathfrak{EP}_c$ are sound with respect to Henkin semantics.

Proof: We already know by 5.5, that calculus $\mathfrak{EP}_{naive}$ is sound. The soundness of the new positive extensionality rules is obvious in Henkin semantics (see Lemmata 2.37 and 2.43), as they simply apply the extensionality properties, which are valid in standard semantics.$\square$

Lemma 5.13 (Proper Derivations). For each non-proper clause $\mathcal{D}$, proper clause $\mathcal{C}$, and clause set $\Phi$, such that $\Phi \ast \mathcal{D} \vdash_{\mathfrak{EP}_c} \mathcal{C}$, we have $\Phi \cup \text{CNF}(\mathcal{D}) \vdash_{\mathfrak{EP}_c} \mathcal{C}$.

Proof: Analogous to lemma 4.10. In the induction step the case $r \equiv Para$ is analogous to the cases $r \in \{Res, Fac, Prim\}$, and the cases $r \in \{Func', \text{Equiv'}\}$ are analogous to the case $r \in \text{UNF}$.$\square$

5.4 Lifting Properties

The clause normalisation lifting property stated in lemma 4.11 is not affected by the modification of the calculus and will be employed in the following lifting lemma for calculi $\mathfrak{EP}_c$. The new problem within this lemma is, that we have to take care of the additional logical connectives $\Rightarrow$ in the extended signature. But fortunately rule $Prim$ became automatically extended as well and allows now to introduce the logical connectives $\Rightarrow$ at head position analogous to the connectives $\neg, \lor, \Pi$ so far.

Lemma 5.14 (Lifting Lemma for $\mathfrak{EP}_c$). Let $\Phi$ be a set of clauses, $\mathcal{D}_1$ be a clause, and $\sigma$ a substitution. We have that:

$$
C \lor [M_\alpha = N_\beta]^T \\
C \lor [M_\alpha \Leftrightarrow N_\beta]^T \text{ Equiv'} \\
C \lor [M_{\alpha \to \beta} = N_{\alpha \to \beta}]^T \text{ X new variable} \\
C \lor [M X = N X]^T \text{ Func'}
$$
5.5. Completeness

1. For each derivation \( \Delta_1 : \Phi \vdash_{EP} D_1 \) exists a substitution \( \delta \), a clause \( D_2 \), and a derivation \( \Delta_2 : \Phi \vdash_{EP} D_2 \), such that \( (D_2)_{\delta} \equiv D_1. \)

2. For each derivation \( \Delta_1 : \Phi \vdash_{EP} D_1 \) exists a substitution \( \delta \), a clause \( D_2 \), and a derivation \( \Delta_2 : \Phi \vdash_{EP} D_2 \), such that \( (D_2)_{\delta} \equiv D_1. \)

Proof:
(1) The proof is by case distinction on all rules in \( EP \), and in all cases we will motivate that there is a derivation \( \Delta_2 \) as required.

Res, Prim, Fac, Cnf, Subst: The corresponding argumentations in 4.12 are not affected by the additional rules or the availability of primitive equality symbols \( =^{\alpha} \) in the signature.

Leib, Equiv, Para: The argumentations for these new rules are analogous to the previous cases.

Triv, Func, Dec, FlexRigid, FlexFlex, Equiv, Leib: These cases are indeed affected by the new primitive equality symbols \( =^{\alpha} \) in the signature, as we may have a unification constraint \([T_1 = T_2]^{F} \) on the instantiated level, but an ordinary negative literal with a flexible head \([H \Upsilon^{\alpha}]^{F} \) on the abstract level. In order to enable the application of the particular unification rule on the abstract layer as well we can use primitive substitution rule Prim in connection with rule Subst in order to introduce the logical connective \( =^{\alpha} \) at head position, i.e., we get the abstract unification constraint \([H_1 \Upsilon^{\alpha} = H_2 \Upsilon^{\alpha}]^{F} \). The remaining argumentations are now analogous to the corresponding ones discussed in 4.12.

(2) The proof is by induction on the length of derivation \( \Delta_1 \). The base case is trivial and in the induction step we first employ statement (1) and then the induction hypothesis.

5.5 Completeness

We first analyse Henkin completeness of the extended calculus \( EP_{E} \). The calculi \( EP \) and \( EP \) are then examined in Subsection 5.6.

Before we present the Henkin completeness theorem 5.27 for \( EP_{E} \), we first adapt the two lemmata 4.15 and 4.15. The first one compares refutability of clause sets in head-normal form with the refutability of clauses in \( \beta \eta \)-normal form, and the second one discusses some important refutational properties.

Lemma 5.15 (Head-Normal Form). Let \( \Phi \) be a set of clauses. If \( \Delta : \Phi \vdash_{EP} \square \), then \( \Delta' : \Phi_{\downarrow \gamma} \vdash_{EP} \square \)

Proof: The proof is analogous to lemma 4.13. In the induction step we additionally have to consider the cases where \( r \in \{ Para, Func', Equiv' \} \), which are analogous to the cases where \( r \in \{ Prim \} \cup \lambdaNF. \)

(One can also prove the other direction, which is not needed in this thesis.)

Lemma 5.16. Let \( \Phi \) be a set of clauses and \( A, B \) be formulae. We have that:

1. If \( \Phi \ast [A]^{T} \vdash_{EP} \square \) and \( \Phi \ast [B]^{T} \vdash_{EP} \square \), then \( \Phi \ast [A \lor B]^{T} \vdash_{EP} \square \).

2. If \( \Phi \ast [A]^{T} \vdash_{EP} \square \) and \( \Phi \ast [A]^{F} \vdash_{EP} \square \), then \( \Phi \ast [A \bowtie B]^{F} \)

Proof: Analogous to lemma 4.15. The new rules do not affect the argumentations.

The main completeness proof of calculus \( EP_{E} \) (cf. 5.27) will employ a generalised paramodulation rule \( GPara \) that is allowed to operate also on non-proper clauses, but which only rewrites identical terms and employs only unit equations instead of conditional equations.

Definition 5.17 (Generalised Paramodulation Rule).

The generalised paramodulation rule \( GPara \) is defined as follows (subterm \( A \) od \( T \) must not contain free variables that are bound outside):

\[
\frac{[T[A]^{\alpha}]^{\beta} \lor C \quad [A =_{\beta} B]^{T}}{[T[B]]^{\alpha} \lor C} \quad GPara
\]
This rule is not restricted to proper clauses and is furthermore allowed to operate on unification constraints. Thus, \(GPam\) generalises rule \(Para\). On the other hand \(GPam\) also restricts \(Pam\) as unification is not employed and thus only identical terms can be replaced by applying this rule. Furthermore, the second premise clause is assumed to consist only of a single positive equation literal.

We want to point out, that rule \(GPam\) is especially designed for the completeness proof 5.27 to ensure the abstract consistency property \(\nabla_i^\alpha\), and the only reason why we introduce the latter restriction for rule \(GPara\) is, that we want to ease the proofs of the lemmata below as much as possible.

Rule \(GPara\) will be proven to be admissible\(^3\) in calculus \(\mathcal{EP}_c\), which completes the completeness proof. For the proof of the admissibility of rule \(GPara\) (in fact we even prove a weak derivability property) we employ additional generalised calculus rules: generalised resolution, factorisation and primitive substitution, which are also admissible or even weakly derivable. Instead of formulating these rules as general as possible, we again (analogously to rule \(GPara\)) restrict ourself to the structures that are needed for our completeness proof.

We want to point out that the investigation of the generalised calculus rules also prepares the investigation of a respective non-normal form calculus.

**Definition 5.18 (Generalised Resolution Rules).** The generalised resolution rules \(GRcs_1\), \(GRcs_2\), and \(GRcs_3\) are defined as follows (for all rules we assume \(\alpha, \beta \in \{ T, F \} \) with \(\alpha \neq \beta\), and for \(GRcs_2\) we assume that \(\overline{\gamma} \notin \text{free}(A)\):

\[
\frac{[A_\gamma \leftarrow \overline{T}_\gamma]^{\alpha} \lor C \quad [A_\gamma \leftarrow \overline{T}_\gamma]^{\beta} \lor D \quad \alpha \neq \beta}{(C \lor D)_{[A_\gamma, X_\gamma; A_\gamma, \overline{T}_\gamma; \overline{T}_\gamma]}} GRcs_1
\]

\[
\frac{[A_\gamma \leftarrow \overline{T}_\gamma]^{\alpha} \lor C \quad [X_\gamma \leftarrow \overline{T}_\gamma]^{\beta} \lor D \quad \alpha \neq \beta}{(C \lor D)_{[A_\gamma/X, T_\gamma; A_\gamma, \overline{T}_\gamma; \overline{T}_\gamma]}} GRcs_2
\]

\[
\frac{[A_\gamma \leftarrow \overline{T}_\gamma]^{\alpha} \lor C \quad [X_\gamma \leftarrow \overline{T}_\gamma]^{\beta} \lor D \quad \alpha \neq \beta}{(C \lor D)_{[A_\gamma/X, T_\gamma; A_\gamma, \overline{T}_\gamma; \overline{T}_\gamma]}} GRcs_3
\]

These rules are not restricted to proper clauses and in this sense they extend rule \(Res\). But the new rules are defined within special contexts only, and in this sense they also restrict the rule \(Res\).

**Definition 5.19 (Generalised Primitive Substitution Rules).** The generalised primitive substitution rule is defined by \((H')\) is a new free variable)

\[
\frac{[H \leftarrow T_\gamma]^{\alpha} \lor C \quad \text{G} \equiv \begin{cases} \neg \text{or} (\lor H'_\gamma) \text{ or } \lambda X_\bullet X & \text{if } \gamma \equiv o \\ \lor & \text{if } \gamma \equiv o \rightarrow o \end{cases}}{(\text{[H \leftarrow T_\gamma]^{\alpha} \lor C})_{\text{G/H}}} \quad GP_\text{rim}
\]

This rule is not restricted to proper clauses, but note that it applies very special instantiations only and that these instantiations are less general as usual partial bindings. E.g., the usual imitation binding \(\lambda X_\bullet \neg (Q^1 X)\) is combined with the subsequent projection binding \(\lambda Y_\bullet Y\) for \(Q^1\), such that we get \(\lambda X_\bullet \neg X\), or simply \(\neg \) as combined binding. An imitation binding for \(\Pi\) is avoided as it is not needed in the completeness proof, i.e., the proof that the generalised resolution rules — and thus the generalised paramodulation rule — are weakly derivable in \(\mathcal{EP}_c\).

\(^3\)Remember the definition of admissible and derivable in our refutation approach: A rule \(r\) is called admissible in one of our resolution calculi \(R\), if adding rule \(r\) to \(R\) does not increase the set of refutable formulae. Furthermore, a rule \(r\) is called derivable in \(R\), if each application of rule \(r\) can be replaced by an alternative derivation in \(R\).
5.5. COMPLETENESS

Definition 5.20 (Generalised Factorisation Rule). The generalised factorisation rule $G\text{Prim}$ is defined by
\[
\frac{|A|^\alpha \lor |B|^\alpha \lor C \quad A_{\sigma} \equiv_{\alpha} B_{\sigma}}{(|A|^\alpha \lor C)_{\sigma}} \quad G\text{Fac}
\]
This rule is not restricted to proper clause but only factorises syntactically unfiable literals.

We will now show, that all our generalised calculus are in a weak sense — modulo subsequent clause normalisation and a kind of lifting — derivable in calculus $\mathcal{EF}_c$. The proof for the weak derivability of $G\text{Para}$ thereby employs the generalised resolution rule $G\text{Res}_1$. $G\text{Res}_2$ and $G\text{Res}_3$ are only needed to ensure the weak derivability property for the generalised resolution rules itself. This also holds for the generalised factorisation and primitive substitution rule.

Remark 5.21. One could omit all the generalised rules and prove the crucial substitutivity property \(\forall\epsilon\) in the main completeness proof of $\mathcal{EF}_c$. Most likely nested induction proofs will then be needed.

Thereby we would lose clarity and also the interesting information on the derivability of the special form generalised calculus rules defined above.

Lemma 5.22 (Weak Derivability of $G\text{Fac}$). Let $\mathcal{C}_1, \mathcal{C}_2$ be clauses and $\mathcal{D}$ be a proper clause. If $\Delta_1: \mathcal{C}_1 \vdash_{G\text{Fac}} \mathcal{C}_2 \vdash_{\mathcal{CNF}} \mathcal{D}$, then there is a derivation $\Delta_2: \mathcal{C}_1 \vdash_{\mathcal{EF}_c} \mathcal{D}$.

Proof: The proof is by induction on the length of the $\mathcal{CNF}$-derivation. In the base case the clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ must be proper, such that we can employ the non-generalised rules $\text{Fac}$ and then full higher-order unification $\mathcal{UNIZ}$ to replace $G\text{Fac}$. In the induction step at least one of the literals of clause rest $C$ must be non-proper. Without loss of generality we can therefore assume, that the first step in the $\mathcal{CNF}$ derivation operates on one of the literals in $C$. We can now switch this first $\mathcal{CNF}$-step with the application of $G\text{Para}$ and apply the induction hypotheses, which leads to the assertion. We want to point out, that we need to employ the conventions stated in Remark 4.19 in this proof.

Lemma 5.23 (Weak Derivability of $G\text{Prim}$). Let $\mathcal{C}_1, \mathcal{C}_2$ be clauses and $\mathcal{D}$ be a proper clause. If $\Delta_1: \mathcal{C}_1 \vdash_{G\text{Prim}} \mathcal{C}_2 \vdash_{\mathcal{CNF}} \mathcal{D}$, then there exists a substitution $\sigma$ and a derivation $\Delta_2: \mathcal{C}_1 \vdash_{\mathcal{EF}_c} \mathcal{E}$, such that $\mathcal{E}_{\sigma} \equiv_{\alpha} \mathcal{D}$.

Proof: Analogously to $G\text{Fac}$ the proof is by induction on the length of the $\mathcal{CNF}$-derivation. In the base case the clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ must be proper, and thus for all instantiations we can employ the normal primitive substitution rule Prim instead. Note that the partial bindings introduced by Prim are more general than the quite special instantiations of rule $G\text{Prim}$, and consequently in all this cases we have that the result of Prim is indeed more general than the application of $G\text{Prim}$. The induction step is analogous to the respective argumentation for $G\text{Fac}$ in lemma 5.22 above.

Lemma 5.24 (Weak Derivability of $G\text{Res}_1, G\text{Res}_2,$ and $G\text{Res}_3$).

1. Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ be clauses and $\mathcal{C}_4$ be a proper clause, such that $\Delta_1: \{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{r} \mathcal{C}_3 \vdash_{G\text{Fac}} \mathcal{C}_4 \vdash_{\mathcal{CNF}} \mathcal{D}$ for $r \in \{G\text{Res}_1, G\text{Res}_2, G\text{Res}_3\}$. If both resolution literals of the generalised resolution step (i.e., the resolution literals in $\mathcal{C}_1$ and $\mathcal{C}_2$) are proper, then there exists $\Delta_2: \{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EF}_c} \mathcal{D}$.

2. Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4$ be clauses and $\mathcal{D}$ be a proper clause, such that $\Delta_1: \{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{r} \mathcal{C}_3 \vdash_{G\text{Fac}} \mathcal{C}_4 \vdash_{\mathcal{CNF}} \mathcal{D}$ for $r \in \{G\text{Res}_1, G\text{Res}_2, G\text{Res}_3\}$. Then there exists a substitution $\sigma$ and a derivation $\Delta_2: \{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EF}_c} \mathcal{E}$, such that $\mathcal{E}_{\sigma} \equiv_{\alpha} \mathcal{D}$.

Proof:

1. The proof is by induction on the length of the $\mathcal{CNF}$-derivation. In the base case the clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ must be proper (i.e., all literals and not just the resolution literals are proper) and thus the application of the generalised resolution rule can be replaced by an application of the ordinary
resolution rule Res and subsequent eager unification. Furthermore the GFac steps can be replaced by an $\mathcal{EP}_c$ derivation as guaranteed by lemma 5.22.

In the step case we simply switch the generalised resolution step with the first clause normalisation step (which is obviously applicable either to $C_1$ or $C_2$ as well and does not operate on $[A]^T$ or $[B]^T$ as these literals are assumed to be proper). The assertion then follows by induction hypotheses.

(2) The proof is by induction on the number of logical connectives in the resolution literals. In the base case, where both resolution literals must be proper, the conclusion follows immediately by (1). In the induction step we examine the three rules separately:

$GR_{es_1}$: Note that in this case the clauses $\{C_1, C_2, C_3\}$ are of form $C_1 : [A \rightarrow \alpha \mid T^\alpha] \vee C$, $C_2 : [A \rightarrow \alpha \mid X^\beta] \vee D$, and $C_3 : (C \vee D)[\alpha X^\gamma]$, where $X^\alpha \notin \text{free}(A)$. We have to consider the following cases: $[A \rightarrow \alpha \mid T^\alpha]$ can be (i) proper (i.e., the logical connectives occur not at head position), (ii) of form $(a^\alpha)$: $[\vee T_1 T_2]^\alpha$ for $n \equiv 2$, (iii) of form $(a^\alpha)$: $[\neg T]^\alpha$, $(b^\alpha)$: $[\forall T]^\alpha$, $(c^\alpha)$: $[\forall M]^\alpha$ for $n \equiv 1$, or (iv) of form $(a^\alpha)$: $[\exists M]^\alpha$, $(b^\alpha)$: $[\forall M N]^\alpha$ for $n \equiv 0$.

As representatives we discuss here the cases where $[A \rightarrow \alpha \mid T^\alpha]$ is of form (iv)$(c^\alpha)$: $[\forall M N]^T$ for $n \equiv 0$, (iii)$(b^F)$: $[\forall T]^F$, and (iii)$(b^F)$: $[\forall T]^F$ for $n \equiv 1$. Note that case (i) is trivial and follows by (1).

(iv)$(c^\alpha)$ In this case clause $C_2$ must be of form $[\forall M N]^F \vee D$. We first perform the following clause normalisation steps:

$$
\begin{align*}
C_1 & : [\forall M N]^T \vee C & C_2 & : [\forall M N]^F \vee D \\
C_3 & : [M]^T \vee [N]^F \vee C & C_6 & : [M]^F \vee D \\
C_5 & : [N]^T \vee [C \vee D] & C_7 & : [N]^F \vee D \\
\end{align*}
$$

Thus, we can derive $C_3$ (and consequently $C_4$) also from clauses $\{C_5, C_6, C_7\}$ with $GR_{es_1}$:

$$
\begin{align*}
C_1 & : [\forall M N]^T \vee C & C_2 & : [\forall M N]^F \vee D \\
C_3 & : [M]^T \vee [N]^F \vee C & C_6 & : [M]^F \vee D \\
C_5 & : [N]^T \vee [C \vee D] & G_{es_1} & \quad C_7 & : [N]^F \vee D \\
\end{align*}
$$

By induction hypotheses (applied to the latter application of $GR_{es_1}$) we know that there is a derivation of the form

$$
\frac{C_1 \quad C_2 \quad C_3 \quad C_6 \quad C_7}{\mathcal{EP}_c} \quad C_4
$$

such that $C_4'$ is more general than $C_4$. Our aim is to apply the induction hypotheses now also to the first application of $GR_{es_1}$, but unfortunately the preconditions are not met, as $C_8$ is not necessarily a proper clause. Therefore we first apply lemma 5.13, which gives us that

$$
\begin{align*}
C_1 & : [\forall M N]^T \vee C & C_2 & : [\forall M N]^F \vee D \\
C_3 & : [M]^T \vee [N]^F \vee C & C_6 & : [M]^F \vee D \\
C_5 & : [N]^T \vee [C \vee D] & G_{es_1} & \quad C_7 & : [N]^F \vee D \\
\end{align*}
$$

\[\mathcal{EP}_c\]

$$
\begin{align*}
\frac{C_1 \quad C_2 \quad C_3 \quad C_6 \quad C_7}{\mathcal{EP}_c} \quad C_4
\end{align*}
$$

\[\text{Note that we here need to employ the conventions of Remark 4.19.}\]
where $\mathcal{CN}(\mathcal{C}_4) \equiv \{D_1, \ldots, D_n\}$. The preconditions for the induction hypotheses are now met for $D_1, \ldots, D_n$, and by applying the induction hypotheses for $n$ times we get

\[
\begin{array}{ccc}
\frac{C_1 \lor C_5 \lor C_6 \lor F \vdash C_2 \lor C_6 \lor F}{E_{\mathcal{P}_k}} & \frac{C_1 \lor C_5 \lor C_6 \lor F \vdash C_2 \lor C_6 \lor F}{E_{\mathcal{P}_k}} \\
D'_1 & \ldots & D'_n
\end{array}
\]

such that the $D'_i$ $(i \equiv 1, \ldots, n)$ are more general than the $D_i$. And as we already know that $\{D_1, \ldots, D_n, \mathcal{C}_4\} \vdash \mathcal{S}_k$, we get with the lifting lemma for $\mathcal{E}_{\mathcal{P}_k}$ (cf. 5.14), that there must be a derivation $\{D'_1, \ldots, D'_n, \mathcal{C}_4\} \vdash \mathcal{S}_k$, where $\mathcal{C}_4'$ is more general than $\mathcal{C}_4$ and $\mathcal{C}_4$, which completes the proof.

(iii)$(b^F)$ Obviously we have that (note that $C_2$ must be of form $[II \cdot X]^F \lor D$)

\[
\begin{array}{c}
\frac{C_1 : [I\!L\! \cdot \! T_\alpha_2]^F \lor C}{C_5 : [T \cdot Y_\alpha]^F \lor C} \quad \frac{C_2 : [I\!L\! \cdot \! X_\alpha_2]^F \lor D}{C_6 : [X \cdot S_\alpha]^F \lor D} \quad \Pi^F \\
\end{array}
\]

\[
C_3 : (C \lor D)[T \cdot X, S \cdot Y] \\
\]

\[
\therefore \mathcal{CN}(\mathcal{C}_3) \\
\mathcal{C}_4
\]

In this derivation $Y$ is a new variable and $S$ a new Skolem term of appropriate type. Induction hypotheses is applicable and we get that

\[
\begin{array}{c}
\frac{C_1 : [I\!L\! \cdot \! T]^F \lor C}{C_5 : [T \cdot Y]^F \lor C} \quad \frac{C_2 : [I\!L\! \cdot \! X]^F \lor D}{C_6 : [X \cdot S]^F \lor D} \quad \Pi^F \\
\end{array}
\]

\[
\therefore \mathcal{CN}(\mathcal{C}_4) \\
\mathcal{C}_4
\]

where $\mathcal{C}_4$ is more general than $\mathcal{C}_4$. This proves the assertion.

(iii)$(b^F)$ Obviously we have that (note that $C_2$ must be of form $[II \cdot X]^F \lor D$)

\[
\begin{array}{c}
\frac{C_1 : [I\!L\! \cdot \! T_\alpha_2]^F \lor C}{C_5 : [T \cdot S_\alpha]^F \lor C} \quad \Pi^F \quad \frac{C_2 : [I\!L\! \cdot \! X_\alpha_2]^F \lor D}{C_6 : [X \cdot Y_\alpha]^F \lor D} \quad \Pi^F \\
\end{array}
\]

\[
C_3 : (C \lor D)[T \cdot X, S \cdot Y] \\
\]

\[
\therefore \mathcal{CN}(\mathcal{C}_3) \\
\mathcal{C}_4
\]

where $Y$ is a new variable and $S$ a new Skolem term of appropriate type. Induction hypotheses is applicable and we get that

\[
\begin{array}{c}
\frac{C_1 : [I\!L\! \cdot \! T]^F \lor C}{C_5 : [T \cdot S]^F \lor C} \quad \Pi^F \quad \frac{C_2 : [I\!L\! \cdot \! X]^F \lor D}{C_6 : [X \cdot Y]^F \lor D} \quad \Pi^F \\
\end{array}
\]

\[
\therefore \mathcal{CN}(\mathcal{C}_4) \\
\mathcal{C}_4
\]

where $\mathcal{C}_4$ is more general than $\mathcal{C}_4$. This proves the assertion.

$G\mathcal{RS}_2$: In this case, the clauses $\{C_1, C_2, C_3\}$ are of form $C_1 : [A \cdot \overline{Y}^n] \lor C, C_2 : [X \cdot \overline{T}^n] \lor D, C_3 : (C \lor D)[A \cdot X, \overline{Y}^n, \overline{T}^n]$, where $n \in \{0, 1\}$. The argumentation is similar to the above case for $G\mathcal{RS}_1$, and we have to consider the following cases: $[A \cdot \overline{Y}^n]$ can be (i) proper, (ii) of form $(a^n) : [V \cdot Y_1]^n$ for $n \equiv 2$, (iii) of form $(a^n) : [\neg Y]^n, (b^n) : [\Pi Y]^n, (c^n) : [(\forall M) Y]^n$ for $n \equiv 1$, or (iv) of form $(a^n) : [\neg M]^n, (b^n) : [\Pi M]^n, (c^n) : [\forall M] N^n$ for $n \equiv 0$. Case (i) again follows immediately by (1), as both resolution literals must be proper. And as a representative for the other cases we
exemplarily discuss the induction step for (iii) \((a^F)\), i.e., in this case the two resolution literals \([A \overline{Y}]^a\) and \([X \overline{T}]^\beta\) are of form \([\neg Y]^F\) and \([X \overline{T}]^T\).

We have that

\[
\begin{align*}
\mathcal{C}_1 : [\neg Y]^F \lor C & \quad \mathcal{C}_5 : [Y]^T \lor C & \quad \mathcal{C}_6 : ([X \overline{T}]^T \lor D)_{[X \overline{T} \cdots \overline{X} \overline{Y} / X]} & \quad \mathcal{C}_7 : ([T]^F \lor D)_{[\neg X]} & \quad \mathcal{G} \text{Prim} \\
\mathcal{C}_2 : [X \overline{T}]^T \lor D & & \mathcal{C}_8 : \mathcal{C}_3 : (C \lor D)_{[\neg X, \overline{T} / Y]} & & \mathcal{G} \text{Res}_2 \\
\mathcal{C}_4 : CNF \\
\mathcal{C}_9 \\
\mathcal{C}_10 \\
\mathcal{C}_11 & & \mathcal{G} \text{Para}
\end{align*}
\]

By Induction hypotheses we get that

\[
\begin{align*}
\mathcal{C}_1 : [\neg Y]^F \lor C & \quad \mathcal{C}_5 : [Y]^T \lor C & \quad \mathcal{C}_6 : ([\neg T]^T \lor D)_{\{ \overline{X} \overline{Y} \}} & \quad \mathcal{C}_7 : ([T]^F \lor D)_{[\neg X]} & \quad \mathcal{G} \text{Prim} \\
\mathcal{C}_2 : [X \overline{T}]^T \lor D & & \mathcal{C}_8 : \mathcal{C}_3 : (C \lor D)_{[\neg X, \overline{T} / Y]} & & \mathcal{G} \text{Res}_2 \\
\mathcal{C}_4 \\
\mathcal{C}_9 \\
\mathcal{C}_10 & & \mathcal{G} \text{Para}
\end{align*}
\]

where \(\mathcal{C}_9\) is more general than \(\mathcal{C}_4\). By applying weak derivability of \(\mathcal{G} \text{Prim}\) (cf. 5.23) two times in connection with the lifting lemma for \(\mathcal{G} \text{Para}\) (cf. 5.14), we now get that \(\{\mathcal{C}_1, \mathcal{C}_2\} \vdash \mathcal{G} \text{Para}\), where \(\mathcal{G} \text{Para}\) is obviously more general than \(\mathcal{C}_4\), and which proves the assertion.

\(\mathcal{G} \text{Res}_3\): In this case the clauses \(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\) are of form \(\mathcal{C}_1 : [A \overline{T}]^a \lor C, \mathcal{C}_2 : [X \overline{Y}]^\beta \lor D, \mathcal{C}_3 : (C \lor D)_{[\overline{A} / X, \overline{T} / \overline{T}]}\), where \(n \in \{0, 1\}\). We have to consider the following cases (analogous to \(\mathcal{G} \text{Res}_1\)): \([A \overline{T}]^a\) can be (i) proper (i.e., the logical connectives occur not at head position), (ii) of form (a\(^b\)): \(\forall \overline{T}_1 \overline{T}_2^a\) for \(n \equiv 2\), (iii) of form (a\(^a\)): \([\neg T]^a\), \([\neg M]^a\), (c\(^a\)): \([\forall M \overline{T}]^a\) for \(n \equiv 1\), or (iv) of form (a\(^0\)): \([\forall M \overline{N}]^a\), \([\forall \overline{M} \overline{N}]^a\) for \(n \equiv 0\). Case (i) again follows by (1). As a representative for the other cases we briefly sketch induction step for (iii) \((c^F)\), i.e., the resolution literals \([A \overline{Y}]^a\) and \([X \overline{T}]^\beta\) are of form \([\forall M \overline{T}]^F\) and \([X \overline{T}]^T\). We first employ rule \(\mathcal{G} \text{Prim}\) and obtain

\[
\begin{align*}
\mathcal{C}_2 : [X \overline{T}]^T \lor D & \quad \mathcal{C}_3 : ([\forall M \overline{N}]^F \lor [D])_{[\forall M \overline{N} / X]} & \quad \mathcal{G} \text{Prim}
\end{align*}
\]

The rest of the proof is analogous to the case (iv) \((c^F)\) for \(\mathcal{G} \text{Res}_1\) as discussed before. The main difference is, that we additionally have to get rid of the initial \(\mathcal{G} \text{Prim}\) step here, which can be done analogous to the case discussed for \(\mathcal{G} \text{Res}_2\) above.

The following paramodulation lemma now shows, that the generalised paramodulation rule \(\mathcal{G} \text{Para}\) is weakly derivable in \(\mathcal{G} \text{Para}\), too. The proof employs the (weakly derivable) generalised resolution rules \(\mathcal{G} \text{Res}_1\) and \(\mathcal{G} \text{Res}_3\).

**Lemma 5.25 (Weak Derivability of \(\mathcal{G} \text{Para}\)).**

1. Let \(\mathcal{C}_1\) be a clause and \(\mathcal{C}_2\) be a proper clause.

   (a) If \(\mathcal{C}_1 : [A]^a \lor D\) and \(\mathcal{C}_2 : [B]^T\), then there exists \(\Delta_1 : \{\mathcal{C}_1, \mathcal{C}_2\} \vdash \mathcal{G} \text{Para}\ D : [C]^a \lor E\), and a substitution \(\sigma\), such that \(\mathcal{D}_\sigma \equiv_\Delta [B]^a \lor D\) (i.e., \(\mathcal{D}\) is more general than the clause we would obtain by a respective generalised paramodulation step).

   (b) If \(\mathcal{C}_1 : [A \overline{T}]^a \lor \mathcal{C}_2 : [B \overline{N}]^T\), then there exists \(\Delta_2 : \{\mathcal{C}_1, \mathcal{C}_2\} \vdash \mathcal{G} \text{Para}\ D : [C \overline{T}]^a \lor E\), and a substitution \(\sigma\), such that \(\mathcal{D}_\sigma \equiv_\Delta [B \overline{N}]^T\).
5.5. COMPLETENESS

2. Let \( \mathcal{C}_1 : [T[A]]^\alpha \lor D \), \( \mathcal{C}_2 : [A = B]^T \) and \( \mathcal{C}_3 : [T[B]]^\alpha \lor D \), such that there is a derivation

\[ \Delta_1 : \{ \mathcal{C}_1, \mathcal{C}_2 \} \vdash_{GPara} \mathcal{C}_3 \vdash_{\mathcal{C}NF} \mathcal{C}_4 \],

for a proper clause \( \mathcal{D} \) derived by clause normalisation from \( \mathcal{C}_3 \). Then there exists a derivation \( \Delta_2 : \{ \mathcal{C}_1, \mathcal{C}_2 \} \vdash_{\mathcal{C}NF} \mathcal{D}' \) and a substitution \( \sigma \), such that \( \mathcal{D}' \equiv \mathcal{D} \).

Proof:

(1) We first concentrate on case (1a) and consider the following \( \mathcal{C}NF \)-derivation:

\[
\frac{\mathcal{C}_2 : [A = B]^T}{\mathcal{C}_4 : [A] \vDash [B]^T} \quad \text{Equiv'} \\
\mathcal{C}_5 : [A]^F \lor [B]^T \\
\mathcal{C}_6 : [A]^T \lor [B]^F
\]

Depending on the polarity \( \alpha \) of literal \( [A]^\alpha \) in clause \( \mathcal{C}_1 \), we now apply the generalised resolution rule \( GRes_1 \) either to \( \mathcal{C}_1 \) and \( \mathcal{C}_5 \), or to \( \mathcal{C}_1 \) and \( \mathcal{C}_6 \), thereby deriving clause \( \mathcal{C}_3 : [B]^\alpha \lor D \). By weak derivability of \( GRes_1 \) (cf. 5.24(2)) we get a derivation \( \Delta_1 : \{ \mathcal{C}_1, \mathcal{C}_2 \} \vdash_{\mathcal{C}NF} \mathcal{D} \) as required, i.e., \( \mathcal{D} \) is more general than \( [B]^\alpha \lor D \). The case (1b) is analogous as

\[
\frac{\mathcal{C}_2 : [(A \rightarrow \alpha \exists x_2)] = [(B \rightarrow \alpha \exists x_2)]^T}{\mathcal{C}_4 : [(A \rightarrow \alpha \exists x_2)] \vDash [(B \rightarrow \alpha \exists x_2)]^T} \quad \text{Equiv'} \\
\mathcal{C}_5 : [A \rightarrow \alpha \exists x_2]^F \lor [B \rightarrow \alpha \exists x_2]^T \\
\mathcal{C}_6 : [A \rightarrow \alpha \exists x_2]^T \lor [B \rightarrow \alpha \exists x_2]^F
\]

and thus the application of \( GRes_1 \) to \( \mathcal{C}_1 \) and \( \mathcal{C}_5 \), or to \( \mathcal{C}_1 \) and \( \mathcal{C}_6 \) derives clause \( \mathcal{C}_3 : [B \rightarrow \alpha \exists x_2]^\alpha \lor D \).

(2) The tedious proof is by induction on the length \( n \) of the embedded \( \mathcal{C}NF \) derivation \( \mathcal{C}_3 \vdash_{\mathcal{C}NF} \mathcal{C}_4 \).

\( n = 0 \) : In the base case \( \mathcal{C}_3 \) must be a proper clause, i.e., the clause rests \( D \) consist only of proper literals and the literal \( [T[B]]^\alpha \) is proper. We now consider the possible cases for literal \( [T[A]]^\alpha \).

1. \( [T[A]]^T \) is a proper literal and \( \mathcal{C}_1 \) is a proper clause. Instead of generalised paramodulation we can apply in this case the standard paramodulation rule \( Para \) to \( \mathcal{C}_1 \) in order to derive \( \mathcal{C}_4 \) by \( \Delta_2 : \{ \mathcal{C}_1, \mathcal{C}_2 \} \vdash_{Para} [T[B]]^T \lor D_1 \lor [A] \lor \vdash_{Triv} \mathcal{C}_4 \). In this case \( \sigma \) is the empty substitution.

2. \( [T[A]]^F \) is a proper literal and \( \mathcal{C}_1 \) is a proper clause. If \( [T[A]]^F \) is not a unification constraint, then the argumentation is analogous to the above case. In case \( [T[A]]^F \) is a unification constraint, it is (without loss of generality) of form \( [T_1[A] = T_2]^F \). In this case we employ a derivation that is analogous to the one already discussed in remark 5.2:

\[
\text{Leib} (\mathcal{C}_1), \mathcal{C}NF : \\
\text{Para} (\mathcal{D}_1, \mathcal{C}_2), \text{Triv} : \\
\text{Res} (\mathcal{D}_3, \mathcal{D}_2), \text{Fac}, \text{Triv} : \\
\text{Dec} (\mathcal{D}_4), \text{Triv} : \\
\]

Again we have that \( \sigma \) is the empty substitution.

3. \( [T[A]]^\alpha \) is not a proper literal, i.e., has a logical connective different from \( \land \) at head position. We consider the possible subterm positions \( \nu \) of term \( A \) in \( T \) (noted \( T[A]_\nu \)).

As this would contradict our assumption \( ([T[B]]^\alpha \) is a proper literal), we know that for literal \( [T[A]]^\alpha \) the following cases are impossible: \( \neg (M[A]^\nu) \), \( \lor (M[A]^\nu) \), \( [\lor M \subseteq (N[A]^\nu)] \) and \( \square (M[A]^\nu) \). The remaining cases are:
(a) Literal [T[A]]^n has form \([-M]^n\), \([\lor M N]^n\) or \([\land M]^n\). Then \(\mathcal{C}_2\) is of form \([\neg M] = B]^T\), \([\lor M N] = B]^T\) and \([\land M] = B]^T\) respectively. For all these cases the assertion now follows by (1a).

(b) Literal [T[A]]^n has form \([-M]^n\), \([\lor M N]^n\), \([\land M]^n\) or \([\lor M N]^n\). Then analogous to above \(\mathcal{C}_2\) is of form \([\neg = B]^T\), \([\lor = B]^T\), \([\land = B]^T\) and \([\lor M = B]^T\) respectively. Applying positive extensionality rule \(\text{Func}^+\) to these clauses leads to \([\neg X = (B X)]^T\), \([\lor X Y = (B X Y)]^T\), \([\land X = (B X)]^T\) and \([\lor M X = (B X)]^T\). For all these cases the assertion now follows by (1b).

\(n > 0\): In the induction step we know that literal \([T[B]]^n\) must have a logical connective at head position, as otherwise the length of the \(\mathcal{G}^\mathcal{F}\)-derivation \(\mathcal{C}_3 \vdash_{\mathcal{G}^\mathcal{F}} \mathcal{C}_4\) is zero. We now distinguish the following two cases:

1. At least one of the literals in \(D\) is not proper. In this case, it is obvious that we can reorder the \(\mathcal{G}^\mathcal{F}\)-derivation, such that one of the clause normalisation steps that modifies \(D\) comes first.\(^5\) Without loss of generality, let us assume that \(\mathcal{G}^\mathcal{F}\)-rule \(v\) transforms in step \(k (1 < k \leq n)\) of derivation \(\Delta_1\) the clause rest \(D\) into \(D'\). Then we can reorder derivation \(\Delta_1\) and obtain a derivation \(\Delta_3:\{C_1, C_2\} \vdash_{\text{GPara}} C_3 \vdash_{\text{GPara}} \mathcal{C}_3: [T[B]]^n \lor D' \vdash_{\mathcal{G}^\mathcal{F}} \mathcal{C}_4\), such that rule \(v\) is applied first. Obviously, we can also switch the first two steps in \(\Delta_3\), such that we get \(\Delta_4:\{C_1, C_2\} \vdash_{\text{GPara}} [C_1: [T[A]]^n \lor D', C_2] \vdash_{\text{GPara}} \mathcal{C}_3 \vdash_{\mathcal{G}^\mathcal{F}} \mathcal{C}_4\). Now induction hypotheses is applicable and we get that there is \(\mathcal{E}^\mathcal{F}_v\)-derivation \(\Delta_5:\{C_1, C_2\} \vdash_{\mathcal{E}^\mathcal{F}_v} \mathcal{C}_4\).

2. \(D\) consist only of proper literals. We again examine the structure of literal \([T[A]]^n\) and distinguish between all possible subterm positions \(p\) of term \(A\) in \(T\).

(a) Literal \([T[A]]^n\) is of form \([\neg (M[A])^p]^n\), \([\lor M (N[A])^p]^n\) or \([\land M (N[A])^p]^n\). As the proofs are analogous we only present the case where \(C_1\) is of form \([\lor M (N[A])]^T\lor D\). Consider now derivation \(\Delta_1\), which obviously applies the \(\mathcal{G}^\mathcal{F}\)-rule \(\lor^T\) first within the normalisation process (note that \(D\) contains only proper literals). Thus we have \(\Delta_1: \{C_1, C_2\} \vdash_{\text{GPara}} C_3 \vdash_{\text{GPara}} \mathcal{C}_3: [M]^T \lor N[B]^n \lor D \vdash_{\mathcal{G}^\mathcal{F}} \mathcal{C}_4\). We can obviously switch the first two steps in \(\Delta_1\), such that we get an alternative derivation \(\Delta_1: \{C_1, C_2\} \vdash_{\text{GPara}} C_3 \vdash_{\text{GPara}} \mathcal{C}_3: [M]^T \lor N[A]^n \lor D, C_2\} \vdash_{\text{GPara}} \mathcal{C}_3 \vdash_{\mathcal{G}^\mathcal{F}} \mathcal{C}_4\).

By induction hypotheses we now get that there is a derivation \(\Delta_2: \{C_1, C_2\} \vdash_{\text{GPara}} \mathcal{C}_3 \vdash_{\text{GPara}} \mathcal{C}_4\).

(b) Literal \([T[A]]^n\) is of form \([-M]^n\), \([\lor M N]^n\) or \([\land M]^n\). Then \(\mathcal{C}_2\) is of form \([\neg M] = B]^n\), \([\lor M N] = B]^n\) or \([\land M] = B]^n\). In all cases the assertion follows immediately by (1a).

(c) Literal \([T[A]]^n\) is of form \([-M]^n\), \([\lor M N]^n\), \([\land M]^n\) or \([\lor M N]^n\). Then \(\mathcal{C}_2\) is of form \([\neg = B]^T\), \([\lor = B]^T\), \([\land = B]^T\) and \([\lor M = B]^T\) respectively. Applying positive extensionality rule \(\text{Func}^+\) to these clauses leads to \([\neg X = (B X)]^n\), \([\lor X Y = (B X Y)]^n\), \([\land X = (B X)]^n\) and \([\lor M X = (B X)]^n\). For all these cases the assertion now follows by (1b).\(\square\)

As \(\square\) is just a special proper clause we immediately get the following corollary.

**Corollary 5.26 (Admissibility of Generalised Paramodulation Rule).**

The generalised paramodulation rule \(\text{GPara}\) is admissible in calculus \(\mathcal{E}^\mathcal{F}_v\).

---

\(^5\) Note that we here need to employ the conventions of Remark 4.19.

\(^6\) Note that we here need to employ the conventions of Remark 4.19.
5.6. THEOREM EQUIVALENCE

Theorem 5.27 (Completeness of \( \mathcal{EP}_c \)). The calculus \( \mathcal{EP}_c \) is complete with respect to Henkin models.

Proof: Let \( \Gamma_2 \) be the set of \( \Sigma \)-sentences which cannot be refuted by the calculus \( \mathcal{EP}_c \) (\( \Gamma_2 := \{ \Phi \subseteq \mathrm{cuff}_\alpha(\Sigma) | \Phi, \Delta \nvdash \square \} \)), then we show that \( \Gamma_2 \) is a saturated abstract consistency class for Henkin models with primitive equality (cf. Definition 3.18). This entails completeness with the model existence theorem for Henkin models with primitive equality (cf. Theorem 3.29).

First we have to verify that \( \Gamma_2 \) validates the abstract consistency properties \( \nabla_a, \nabla_n, \nabla_b, \nabla_v, \nabla_A, \nabla_c, \nabla_e, \nabla_q \), and that \( \Gamma_2 \) is saturated. For all of these cases, the proofs are identical to the respective arguments in theorem 4.16. The only difference is that we employ lemmata 5.14 and 5.16 instead of 4.12 and 4.15. Thus, all we need to ensure is the validity of the additional abstract consistency properties \( \nabla'_a \) and \( \nabla'_v \) for primitive equality.

\[
\nabla'_a \quad \neg (A =^\alpha A) \notin \Phi. \\
\nabla'_v \quad \text{If } F[A]_p \in \Phi \text{ and } A = B \in \Phi, \text{ then } \Phi \ast F[B]_p \in \Gamma_2.
\]

(\( \nabla'_a \)) We have that \( [A =^\alpha A]^T \vdash^{\text{equiv}} \square \), and thus \( \neg (A =^\alpha A) \) cannot be in \( \Phi \).

(\( \nabla'_v \)) Analogous to the cases in 4.16 we show the contrapositive of the assertion, i.e., we assume that there is derivation \( \Delta_1 : \Phi, A \vdash^{\text{equiv}} \square \), and then ensure that \( \Delta_2 : \Phi \ast [F[A]_p]^T \ast [A = B]^T \vdash^{\mathcal{EP}_c} \square \). Now consider the following \( \mathcal{EP}_c \)-derivation:

\[
\begin{align*}
F[A]_p & \quad [A = B]^T \\
\end{align*}
\]

By Corollary 5.26 the generalised paramodulation rule \( \text{GPara} \) is admissible for calculus \( \mathcal{EP}_c \), which guarantees, that there is a derivation \( \Delta_2 : \Phi, A \vdash^{\text{equiv}} \square \), that does not apply \( \text{GPam} \). This proves the assertion. \( \square \)

5.6. Theorem Equivalence

We shall now prove that \( \mathcal{EP}_c \) and \( \mathcal{EP}_f \) are theorem equivalent. Theorem equivalence of \( \mathcal{EP}_c \) and \( \mathcal{EP}_f \) is only presented as a conjecture.

Lemma 5.28 (Proper Clauses in \( \mathcal{EP} \) and \( \mathcal{EP}_f \)). For each non-proper clause \( \mathcal{C} \) and clause set \( \Phi \), such that \( \Phi \vdash^{\mathcal{EP}_c} \mathcal{C} \), we have \( \Phi \vdash^{\mathcal{EP}_f} \mathcal{C} \).

Proof: Analogous to lemma 4.20. In the induction step, the case for rule \( \text{Para} \) is treated analogously to the cases \( r \in \{ \text{Res, Fac, Prim} \} \), and the rules \( \text{Func}^\ast \) and \( \text{Equiv}^\ast \) are treated analogously to the unification rules. \( \square \)

As \( \square \) is also a proper clause we immediately get the following corollary:

Corollary 5.29 (Theorem Equivalence of \( \mathcal{EP}_c \) and \( \mathcal{EP}_f \)). The calculi \( \mathcal{EP}_c \) and \( \mathcal{EP}_f \) are theorem equivalent.

Conjecture 5.30 (Theorem Equivalence of \( \mathcal{EP} \) and \( \mathcal{EP}_c \) (or \( \mathcal{EP}_f \))). The calculi \( \mathcal{EP} \) and \( \mathcal{EP}_c \) (or \( \mathcal{EP}_f \)) are theorem equivalent.

The proof for the latter conjecture will most likely be analogous to the proof of the theorem equivalence for calculi \( \mathcal{ER}_c \) and \( \mathcal{R} \).
Chapter 6

Extensional Higher-Order
RUE-Resolution:  $\mathcal{ERUE}$

6.1 Resolution on Unification Constraints

In this section we will extend the Resolution by Unification and Equality approach of Digrigoli [Digi9] to higher-order logic. The key idea of the resulting calculus $\mathcal{ERUE}$ is to allow resolution and factorisation rules also to operate on unification constraints.\(^1\) More precisely, our approach allows to compute partial $E$-unifiers with respect to a specified theory $E$ (where $E$ is given in form of a set of unitary or conditional equations in clause form in the search space) by employing resolution on unification within constraint the calculus itself. This is due to the fact that the extensional higher-order resolution approach already realises a test calculus for general higher-order $E$-pre-unification (or general higher-order $E$-unification in case we also add rule $\text{FlexFlex}$; cf. Remark 4.8). Furthermore, each partial instantiation can be applied to a clause with rule $\text{Subst}$. Finally, like in the traditional first-order RUE-resolution approach, the non-solved unification constraints are encoded as still open unification constraints within the particular clauses itself.

Remark 6.1 (Incompleteness of a naive RUE-Resolution approach). Similar to our naive adaptation of the first-order paramodulation approach, we obtain a Henkin incomplete RUE-resolution approach, if we do not additionally add the positive extensionality axioms. This is illustrated by example $\mathcal{E}^\text{para}_r$ already used in the incompleteness proof 5.6 for $\mathcal{EP}$. In a naive \textbf{ERUE} approach, i.e., without positive extensionality rules, no single rule would be applicable to the contradictory unit clause $C_1: \{a = \neg a\}^7$.

6.2 Basic Definitions

Similar to the approaches $\mathcal{ER}$ and $\mathcal{EP}$ discussed in the previous chapters, we further extend the extensional higher-order RUE-resolution approach by adding rule $\text{FlexFlex}$ and lifting the single clause normalisation rules to calculus level.

Definition 6.2 (Extensional Higher-Order RUE-Resolution).

$\mathcal{ERUE}$ The calculus $\mathcal{ERUE}$ consists of the rules of calculus $\mathcal{ER}$ (see Figure 4.2) enriched by the positive extensionality rules displayed in Figure 5.2. The most important aspect is that we allow to resolve and factorise on unification constraints. Furthermore, we want to point out that unification constraints are assumed to be symmetric which could also be formulated by

\(^1\)The idea to resolve on unification constraints arose while discussing the necessity of Rule $\text{Leib}$ within extensional higher-order resolution with Frank Pfenning during a stay at Carnegie Mellon University.
6.3. LIFTING PROPERTIES

the following rule:

\[ C_1 : C \lor [A = B]^F \]
\[ C_2 : C \lor [B = A]^F \]
\[ \text{Sym} \]

We employ this convention despite the fact that the symmetry rule \( \text{Sym} \) is derivable (c.f. remark 6.3) in \( \mathcal{RUE} \) — as well as in \( \mathcal{ER} \) and \( \mathcal{EP} \) — as it shortens and eases derivations and is acceptable with respect to its complexity in practice.

\( \mathcal{RUE}_f \) The extension \( \mathcal{RUE}_f \) of calculus \( \mathcal{RUE} \) that employs full higher-order unification instead of higher-order pre-unification is defined as \( \mathcal{RUE}_f := \mathcal{RUE} \cup \{ \text{FlexFlex} \} \).

\( \mathcal{RUE}_{fp} \) The calculus \( \mathcal{RUE}_{fp} \) that employs stepwise instead of exhaustive clause normalisation is defined by \( \mathcal{RUE}_{fp} := (\mathcal{RUE} \setminus \{ \text{CNF} \}) \cup \text{CNF} \).

We assume furthermore, that the result of each rule application is transformed into head-normal form. A set of formulae \( \Phi \) is refutable in calculus \( \mathcal{RUE} \), if there is a derivation \( \Delta : \Phi \vdash_{\text{RUE}} \square \), where \( \Phi_{\Delta} := \{ [F]^T | F \in \Phi \} \) is the set obtained from \( \Phi \) by simple pre-classification.

**Remark 6.3 (Symmetric Unification Constraints).** The following derivation shows that the symmetry rule \( \text{Sym} \) is derivable in \( \mathcal{RUE} \) (\( \mathcal{ER} \) and \( \mathcal{EP} \)):

- \( \text{Leib}(C_1) : \]
  - \( D_1 : C \lor [p A]^F \)
  - \( D_2 : C \lor [p B]^F \)
- \( \text{Res}(D_2, D_1), \text{Fac}, \text{Triv} : \]
  - \( D_3 : C \lor [p B = p A]^F \)
- \( \text{Dec}(D_3), \text{Triv} : \]
  - \( C_2 : C \lor [B = A]^F \)

**Theorem 6.4 (Soundness of Extensional Higher-Order RUE-Resolution).**
The calculi \( \mathcal{RUE}_{fp} \), \( \mathcal{RUE}_f \), and \( \mathcal{RUE} \) are sound with respect to Henkin semantics.

**Proof:** Soundness of most of our calculus rules have already been discussed in lemmata 4.9 and 5.12. We additionally have to verify that resolution and factorisation on unification constraints is sound wrt. Henkin semantics. Note that unification constraints are treated as ordinary negative literals with a primitive equality symbol at head position. Thus, there is nothing new to show here and we can employ the standard argumentation for soundness of the resolution and factorisation rule.

**Lemma 6.5 (Proper Derivations).** For each non-proper clause \( \mathcal{C} \), proper clause \( \mathcal{C}' \), and clause set \( \Phi \), such that \( \Phi \vdash_{\text{RUE}_{fp}} \mathcal{C}' \), we have that \( \Phi \cup \text{CNF}(\mathcal{C}) \vdash_{\text{RUE}_{fp}} \mathcal{C}' \).

**Proof:** Analogous to lemmata 4.10 and 5.13. It does not cause any problems that we allow to resolve on unification constraints.

6.3 Lifting Properties

The clause normalisation lifting lemma 4.11 is not affected by the modifications to the calculus and can be safely employed in the following lifting lemma for calculus \( \mathcal{RUE}_{fp} \).

The proof of the adapted main lifting lemma is analogous to the one presented for \( \mathcal{ER}_{fp} \).

**Lemma 6.6 (Lifting Lemma for \( \mathcal{RUE}_{fp} \)).**
Let \( \Phi \) be a set of clauses, \( D_1 \) be a clause and \( \sigma \) a substitution. We have that:

1. For each derivation \( \Delta_1 : \Phi_\sigma \vdash_{\text{RUE}_{fp}} D_1 \), there exists a substitution \( \delta \), a clause \( D_2 \), and a derivation \( \Delta_2 : \Phi \vdash_{\text{RUE}_{fp}} D_2 \), such that \( (D_2)_\delta \equiv D_1 \).

2. For each derivation \( \Delta_1 : \Phi_\sigma \vdash_{\text{RUE}_{fp}} D_1 \) there exists a substitution \( \delta \), a clause \( D_2 \), and a derivation \( \Delta_2 : \Phi \vdash_{\text{RUE}_{fp}} D_2 \), such that \( (D_2)_\delta \equiv D_1 \). (Note that this claim is stronger than \( \Phi \) is refutable by \( \mathcal{RUE}_{fp} \), provided that \( \Phi_\sigma \) is.)
6.4 Completeness

Analogous to the Subsections 4.4 and 5.5, we analyse in this subsection Henkin completeness of calculus \( \mathcal{E}RUE \). The calculi \( \mathcal{ER} \) and \( \mathcal{E}RUE \) are then examined in Subsection 6.5. Like in the case of extensional higher-order resolution, theorem equivalence between \( \mathcal{E}RUE \) and \( \mathcal{E}RUE_c \) has not been proven formally yet and will only be presented as a conjecture.

We first adapt the two lemmata 5.16 and 5.24. The former compares refutability of clause sets in head-normal form with refutability of clauses in \( \beta\eta \)-normal form and the latter one discusses some important refutational properties.

Lemma 6.7 (Head-Normal Form).
Let \( \Phi \) be a set of clauses. If \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} \square \), then \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} \square \).

Proof: The proof is analogous to lemma 4.13. In the induction step we additionally have to consider the cases where \( r \in \{Para, Func', Equiv'\} \), which are analogous to the cases where \( r \in \{Prim\} \cup \mathcal{CNF} \).

Lemma 6.8. Let \( \Phi \) be a set of \( \Sigma \)-sentences and \( \Phi_{\downarrow} \) be the corresponding set of pre-clauses. Furthermore let \( A, B \) be formulae and \( C, D \) be clauses. We have that:
1. If \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [A]^T \neg \) and \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [B]^T \neg \), then \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [A \lor B]^T \neg \).
2. If \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [A]^T \neg \) and \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [A]^F \neg \), then \( \Phi_{\downarrow} \vdash_{\mathcal{ER}UE_c} [A \Rightarrow B]^F \).

Proof: Analogous to lemma 4.15 and 5.16. The new or slightly modified rules do not affect the arguments.

As in Chapter 5, we will now show that the generalised rules \( GFaC, GPrim, GReS1, GReS2 \) and \( GReS3 \) are in a weak sense (modulo subsequent clause normalisation and a kind of lifting) derivable in calculus \( \mathcal{E}RUE_c \). Derivability of \( GReS1 \) will then be employed to establish the admissibility of generalised resolution rule \( GPara \) for calculus \( \mathcal{E}RUE_c \). We will finally use the latter result in the main completeness proof for calculus \( \mathcal{E}RUE_c \) to prove the crucial substitutivity property \( \downarrow^* \) of primitive equality.

Lemma 6.9 (Weak Derivability of \( GFaC \)). Let \( C_1, C_2 \) be clauses and \( D \) be a proper clause. If \( \Delta_1 : C_1 \vdash_{GFaC} C_2 \vdash_{\mathcal{CNF}} D \), then there is a derivation \( \Delta_2 : C_1 \vdash_{\mathcal{ER}UE_c} D \).

Proof: Analogous to the corresponding Lemma 5.22 for \( \mathcal{E}P_k \). Resolution and factorisation on unification constraints do not cause serious new problems.

Lemma 6.10 (Weak Derivability of \( GPrim \)). Let \( C_1, C_2 \) be clauses and \( D \) be a proper clause. If \( \Delta_1 : C_1 \vdash_{GPrim} C_2 \vdash_{\mathcal{CNF}} D \), then there exists a substitution \( \sigma \) and a derivation \( \Delta_2 : C_1 \vdash_{\mathcal{ER}UE_c} \), such that \( \mathcal{E}_{\sigma} \equiv_{\Delta} D \).

Proof: Analogous to the corresponding Lemma 5.23 for \( \mathcal{E}P_k \). Resolution and factorisation on unification constraints causes no extra problem.

Lemma 6.11 (Weak Derivability of \( GReS1, GReS2 \), and \( GReS3 \)).

1. Let \( C_1, C_2, C_3 \) be clauses and \( C_4 \) be a proper clause, such that \( \Delta_1 : \{C_1, C_2\} \vdash_{\mathcal{ER}UE_c} C_3 \vdash_{\mathcal{CNF}} D \) for \( r \in \{GReS1, GReS2, GReS3\} \). If both resolution literals of the generalised resolution step are proper, then there exists \( \Delta_2 : \{C_1, C_2\} \vdash_{\mathcal{ER}UE_c} D \).
2. Let \( C_1, C_2, C_3, C_4 \) be clauses and \( D \) be a proper clause, such that \( \Delta_1 : \{ C_1, C_2 \} \vdash^* C_3 \vdash^* \text{GPara} \) \( C_4 \vdash^* D \) for \( r \in \{ GRes_1, GRes_2, GRes_3 \} \). Then there exists a substitution \( \sigma \) and a derivation \( \Delta_2 : \{ C_1, C_2 \} \vdash^*_\text{EF} \varnothing, \) such that \( \varnothing^\sigma \equiv D \).

Proof: The argument is analogous to 5.24 and resolution on unification constraints does cause new problems. \( \square \)

The following paramodulation Lemma 6.12 now shows that the generalised paramodulation rule \text{GPara} is admissible in \( \text{EF} \).

Whereas Lemma 6.12(1) is again analogous to Lemma 5.25(1), we can prove in 6.12(2) only admissibility instead of the weak derivability property in 5.25(2). This is because in 5.25 we were able to reduce the applications of the generalised paramodulation rule either to the generalised resolution rules and thus finally to the proper resolution rule \text{Res} or to the proper paramodulation rule \text{Para}. Whereas the reduction to rule \text{Res} is analogous here as well, we cannot employ the reduction to the proper paramodulation \text{Para}. Instead we have to show that alternative reductions are possible which employ the \( \text{RUE} \)-resolution idea to resolve against unification constraints. The latter causes the loss of the weak derivability property. But fortunately admissibility is sufficient for our purposes.

**Lemma 6.12 (Admissibility of Generalised Paramodulation Rule).**

1. Let \( C_1 \) be a clause and \( C_2 \) be a proper clause.

   (a) If \( C_1 : [A]^a \vdash D \) and \( C_2 : [A = B]^T \), then there exists \( \Delta_1 : \{ C_1, C_2 \} \vdash^*_\text{EF} \varnothing : [C]^a \vdash E \), and a substitution \( \sigma \), such that \( \varnothing^\sigma \equiv [B]^a \vdash D \) (i.e., \( D \) is more general than the clause we would obtain by a respective generalised paramodulation step)

   (b) If \( C_1 : [A_{\tau \to} T_{\tau}]^a \vdash D \) and \( C_2 : ([A_{\tau \to} X_{\tau}]^a) = (B_{\tau \to} X_{\tau}]^a) \vdash E \) where \( X_{\tau} \notin \text{free}(T_{\tau}) \cup \text{free}(A) \), then there exists \( \Delta_2 : \{ C_1, C_2 \} \vdash^*_\text{EF} \varnothing : [C_{\tau \to} D_{\tau}]^a \vdash E' \) and a substitution \( \sigma \), such that \( \varnothing^\sigma \equiv [B_{\tau \to} T_{\tau}]^a \vdash D \).

2. Let \( C_1[A/B]^p \) be a clause that is obtained from clause \( C \) by replacing the occurrences of term \( B \) at positions \( p \in P \) by term \( A \), and let \( \Phi \) be a set of clauses. If \( \Delta_1 : \Phi \ast C_1[A/B]^p \ast [A = B]^T \vdash^* \text{GPara} \) \( \Phi \ast C_1[A/B]^p \ast [A = B]^T \ast C_1[A/B]_{P \setminus \{p\}} \vdash^*_\text{EF} \varnothing \), then there is a derivation \( \Delta_2 : \Phi \ast [A = B]^T \vdash^*_\text{EF} \varnothing \).

Proof:

(1) For the cases (1a) and (1b) the proof is analogous to the corresponding argumentation in lemma 5.25(1). The only difference is that we here employ Lemma 6.11 instead of 5.24.

(2) The proof is by induction on the length of \( \Delta_1 \) and the base step is the most complicate one as we cannot reduce \text{GPara} paramodulation steps to \text{Para} steps, but instead have to replace them by pure \( \text{EF} \) derivations.

If \( \Box \in \Phi \ast C_1[A/B]_{P \setminus \{p\}} \ast [A = B]^T \) the assertion follows trivially. Therefore let us assume that \( \Box \equiv C_1[A/B]_{P \setminus \{p\}} \). Hence \( C_1[A/B]_{P \setminus \{p\}} \) has form \( C_1[A/B]_{P \setminus \{p\}} \cup [T[A/B]_{P \setminus \{p\}}]^a \), where all but one literal must be flex-flex unification constraints. Without loss of generality we assume that this is the literal \( [T[A/B]_{P \setminus \{p\}}]^a \). Then the new clause \( C_1[A/B]_{P \setminus \{p\}} \ast [A = B]^T \ast C_1[A/B]_{P \setminus \{p\}} \vdash^*_\text{EF} \varnothing \), where position \( p' \) in literal \( [T[A/B]_{P \setminus \{p'\}}]^a \) specifies the position where the replacement has been taken place. Obviously this clause must consist only of flex-flex unification constraints, and hence, polarity \( a \) must be \( F \), as otherwise it would be different from \( \Box \). Thus, the literal \( [T[A/B]_{P \setminus \{p'\}}]^a \) is a flex-flex constraint. We now consider the position \( p' \) in literal \( [T[A/B]_{P \setminus \{p'\}}]^F \), where the generalised paramodulation step has been applied to. The first two possibilities concentrate on replacements that include the head position in this literal.

- If \( p' \) specifies the replacement of the whole atom of the literal in focus, i.e., \( T[A/B]_{P \setminus \{p'\}} \equiv A \), then we get the assertion by (1a).
If $p'$ specifies the replacement of a proper prefix term of the atom $T[A/B]_{r_2}$ (i.e., $T[A/B]_{r_2}$ looks like $(A \overline{U}^n)$), then we first apply the positive functional extensionality rule $Func'$ for $n$-times to $[A = B]^F$, thereby generating a proper clause $[A \overline{X}^n = B \overline{X}^n]^F$. Now $(1b)$ is applicable, which gives us the assertion.

Next we examine the cases where a proper subterm of the atom of $[T[A/B]_{r_2}]^F$ is replaced. As the replacement must lead to a flex-flex unification constraint we already know that $[T[A/B]_{r_2}]^F$ must be a unification constraint and thus must have form $[T[A/B]_{r_2} = T_2[A/B]_{r_4}]^F$.

- If $p'$ refers to a proper subterm of either $T_1[A/B]_{r_2}$ or $T_2[A/B]_{r_4}$, then it must be the case that the generalised paramodulation step was not necessary as the literal $[T[A/B]_{r_2}]^F$ already is a flex-flex unification constraint and thus $C_1[A/B]_P \equiv \Box$, which trivially gives us the assertion.

- Without loss of generality let us assume that $p'$ refers to a prefix of term $T_1[A/B]_{r_2}$. We then know that $[T_1[A/B]_{r_2} = T_2[A/B]_{r_4}]^F$ must have form $[A \overline{U[A/B]_{r_2}}^n = H \overline{V[A/B]_{r_4}}^n]^F$, where $n, m \geq 0$. As the replacement of $A$ by $B$ introduces a flex-flex constraint, we know that term $B$ must have a flexible head, i.e., $B$ has form $(F \overline{R}^l)$, such that $l \geq 0$. Now consider term $A$: If the head of $A$ is a variable then again the replacement of $A$ is not necessary as we already have a flex-flex constraint without employing generalised paramodulation rule. If on the other hand $A$ has a rigid head, i.e., $A$ has form $(a \overline{W}^k)$ for $k \geq 0$, then the proof is a bit more complicate as we have to construct a refutation using $C_1[A/B]_P$ and $[A = B]^F$ without employing generalised paramodulation. Before discussing this derivation let us sum up the structural restrictions that are given. The clause $C_1[A/B]_P$ looks like $C_1'[A/B]_{r_1} \lor ((a \overline{W}^k)(U[A/B]_{r_2})) \equiv (H \overline{V[A/B]_{r_4}})^F$ and clause $[A = B]^F$ has form $[(a \overline{W}^k) = \overline{V}^{\sigma} \equiv \Box (F \overline{R}^l)]^F$, where $a$ is a predicate constant and $F$ a predicate variable of appropriate type. We apply rule FlexRigid to the former clause and positive extensionality rule $Func'$ for $n$-times to the latter clause, such that we get:

$$C_2 : C_1'[A/B]_{r_1} \lor ((a \overline{W}^k)(U[A/B]_{r_2})) \equiv (H \overline{V[A/B]_{r_4}})^F$$

$$\lor [H = \lambda \overline{Z}^m, a (K \overline{Z}^m)^{k+n}]^F$$

$$C_3 : [(a \overline{W}^k) \overline{Y}^\sigma = \Box (F \overline{R}^l)]^F$$

The variable $K$ in $C_2$ and the variables $\overline{V}^\sigma$ in $C_3$ are new free variables. Next, we instantiate the imitation binding in clause $C_2$ with rule Subst. (Thereby we assume that variable $H$ does not occur in any of the literals in $C_1'[A/B]_{r_1}$. If on the other hand $H$ occurs in $C_1'[A/B]_{r_1}$, then we only get a problem if $H$ occurs at literal head positions. In these cases we can use an analogous argumentation to the following ones for these literals as well.)

$$C_4 : C_1'[A/B]_{r_1} \lor [(a \overline{W}^k)(U[A/B]_{r_2})] \equiv (a(K \overline{V[A/B]_{r_4}})^{k+n}]^F$$

By resolving between $C_4$ and $C_3$ we get (note that the unification terms of both unification constraints must have the same type)

$$C_5 : C_1'[A/B]_{r_1} \lor [(a \overline{W}^k)(U[A/B]_{r_2})] \equiv (a(K \overline{V[A/B]_{r_4}})^{k+n}]^F$$

$$\lor [(a(K \overline{V[A/B]_{r_4}})^{k+n}] \equiv \Box (F \overline{R}^l)]^F$$

We now apply the decomposition rule (2 times) and immediately delete the trivial pair $[\equiv =]_F$ with rule Triv.

$$C_5 : C_1'[A/B]_{r_1} \lor [(a \overline{W}^k)(U[A/B]_{r_2})] \equiv (a \overline{W}^k) \overline{Y}^\sigma]^F$$

$$\lor [(a(K \overline{V[A/B]_{r_4}})^{k+n}] = (a \overline{W}^k) \overline{Y}^\sigma]^F$$

We do not need to switch the latter unification constraint here as $C_4$ and $C_3$ already have the right constitution. But generally it might be necessary to employ the symmetry rule to clause $C_5$. 

---

We do not need to switch the latter unification constraint here as $C_4$ and $C_3$ already have the right constitution. But generally it might be necessary to employ the symmetry rule to clause $C_5$. 

---

2We do not need to switch the latter unification constraint here as $C_4$ and $C_3$ already have the right constitution. But generally it might be necessary to employ the symmetry rule to clause $C_5$. 

---

2We do not need to switch the latter unification constraint here as $C_4$ and $C_3$ already have the right constitutio
6.4. COMPLETENESS

We know that $\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1}$ contains only flex-flex-constraints and that $F$ and $\gamma^n$ do not occur in them. And as the latter two unification constraints of $\mathcal{C}_2$ are obviously solvable with substitution

$$\frac{(U[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2})^k}{\gamma^n, \lambda \lambda^\gamma^{k+n}, (a(K \mathbf{V}[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2})^k)}$$

we know that $\mathcal{C}_2 \vdash_{\mathcal{G}\mathcal{P}_2} \Box$, which proves the assertion.

$n > 1$: In the step case we consider the second derivation step in $\Delta_1: \Phi \ast \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}} \ast [\mathbf{A} = \mathbf{B}]^T \vdash_{\mathcal{G}\mathcal{P}_{\mathcal{A}}} \Phi \ast \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}]} \vdash_{\mathcal{G}\mathcal{P}_{\mathcal{A}}} \Phi \ast \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}]} \ast \mathcal{C}_2 \vdash_{\mathcal{G}\mathcal{P}_2} \Box$, i.e., we focus on the first step following the generalised paramodulation step. If the premise clause(s) within the application of rule $r$ are different from $\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}]}$ we can obviously switch the first two derivation steps of $\Delta_1$, such that we get the assertion trivially by employing the induction hypothesis. This even holds if the application of rule $r$ indeed uses $\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}]}$ but operates on a literal that was not affected by the initial generalised paramodulation step, i.e., on a literal different from the position where $p$ refers to. Thus, for all of the following cases let us assume that $\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}]}$ is of form $[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \land \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}$ (for respective position lists $P_1$, $P_2$ and position $p$, i.e., $p'$ and $p$ are equal; or is one removes the first element from position $p$, specifying the subterm that was modified within the initial generalised paramodulation step) and that $r$ operates in the second derivation step on literal $[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha$. Note that under this assumption the non-modified original clause $\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}$ is of form $[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}]^\alpha \land \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}$.

$r \in \{Res, Fac\}$: As both rules do not depend on the term structure of the resolution or factorisation literals, we can in both cases switch the initial derivation steps, such that the assertion easily follows by induction hypothesis. We will briefly illustrate this here for rule $Res$; the argumentation for $Fac$ is analogous. Let us assume that there is a clause $\mathcal{C}_2 \in \Phi$ that is of form $[T_2]^\beta \lor C_2^\beta$ and that

$$[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor [T_2]^\beta \lor C_2^\beta \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor ([T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}) = \mathcal{T}_2^\beta \vdash_{\mathcal{G}\mathcal{P}_2} Res$$

Then an analogous resolution proof step is possible between $\mathcal{C}_2$ and the non-modified clause $[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}$

$$[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor [T_2]^\beta \lor C_2^\beta \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor ([T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha \lor \mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2}) = \mathcal{T}_2^\beta \vdash_{\mathcal{G}\mathcal{P}_2} Res$$

We can obviously apply generalised paramodulation rule $\mathcal{G}\mathcal{P}_2$ to the latter clause (at position $p'$ in the left term of the new unification constraint) thereby generating exactly the same clause as in the above original derivation.

$$\frac{\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor C_2 \lor [T_2 = (T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2})]^\beta \lor [\mathbf{A} = \mathbf{B}]^T}{\mathcal{C}_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_2} \lor C_2 \lor ([T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}] = \mathcal{T}_2^\beta) \vdash_{\mathcal{G}\mathcal{P}_2} G\mathcal{P}_2}$$

Now the assertion follows by induction hypothesis.

$r \in \{Prim, Func, Equiv\}$: The three cases are similar and therefore we only discuss rule $Prim$ here. Thereby we consider the position $p'$ of the subterm that has been rewritten in the generalised paramodulation step. (i) If the position $p'$ in literal $[T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}]^\alpha$ refers to a proper subterm, then we get the assertion analogously to the cases for $r \in \{Res, Fac\}$ above by switching the first two derivation steps and employing the induction hypothesis. (ii) In the other case position $p'$ refers to a flexible prefix term of $T_1[\mathbf{A}/\mathbf{B}]_{\mathcal{P}_1[\{p\}] Or}$, i.e., the literal has form

$$\frac{\mathcal{A}}{[[\ldots(\ldots(H \mathbf{U}_1)\ldots \mathbf{U}_k)\ldots \mathbf{U}_n)]^\alpha} \quad (s)$$

Note that we here need to employ the conventions of Remark 4.19.
for $0 \leq k \leq n$. In case $k \equiv n$ we can replace the initial paramodulation step in $\Delta_1$ by an alternative one not employing generalised paramodulation by (1a) and get the assertion by induction hypothesis. In case $k \leq n$ we first apply positive extensionality rule $\text{Func}$ for $(k \cdot n)$-times to the clause $[A = B]^T$ leading to $[A \ Y^{k-n} = B \ Y^{k-n}]^T$. This time we can employ (1b) to replace the initial paramodulation step in $\Delta_1$ by an alternative one. Again, the assertion follows by induction hypothesis.

$r \in \mathcal{U} \mathcal{V}$ If position $p'$ in literal $[T_1[A/B]_{\mathcal{P}_1(p')}]^\alpha$ refers to a flexible prefix term as illustrated in (s) above, we get the assertion analogously to the previous case by (1a) and induction hypothesis, or by (1b) in combination with an appropriate modification of clause $[A = B]^T$ with rule $\text{Func}$. If on the other hand position $p$ refers to a proper subterm of $[T_1[A/B]_{\mathcal{P}_1(p')}]^\alpha$, then we get the result by employing derivation that is analogous to one already employed in Remark 5.2 (which shows that paramodulation into unification constraints is derivable).

$r \in Cn f$: We again differentiate between the following two cases: (i) position $p'$ in $[T_1[A/B]_{\mathcal{P}_1(p')}]^\alpha$ refers to proper subterm of the literals and (ii) position $p'$ in $[T_1[A/B]_{\mathcal{P}_1(p')}]^\alpha$ refers to a prefix term of the atom (like in (s) above). In case (i) the assertion follows by induction hypothesis after switching the first two derivation steps. This is possible as derivation step $r$ is obviously also applicable to the initial clause first, such that an subsequent application of rule $\text{Para}$ leads to $\mathcal{C}_2$. In case (ii) the argumentation is analogous to the case (ii) for $r \in \{\text{Prim}, \text{Func}, \text{Equn}\}$ discussed above, i.e., we get the assertion by employing either (1a) and induction hypothesis or by (1b) in combination with an $n$-times application of rule $\text{Func}$ to literal $[A = B]^T$ and induction hypothesis.

\[ \begin{array}{c}
\text{Theorem 6.13 (Completeness of } E\mathcal{R}E_{fe} \text{). The calculus } E\mathcal{R}E_{fe} \text{ is complete with respect to Henkin models.} \\
\text{Proof:} \text{ Let } \mathcal{I}_f \text{ be the set of } \Sigma \text{-sentences which cannot be refuted by the calculus } E\mathcal{R}E_{fe} \quad (\mathcal{I}_f := \{ \Phi \in \text{cwff}_{\Sigma} \mid \forall \Phi_{cl} \forall e \in e_{\Phi} \square \}). \text{ Then we show that } \mathcal{I}_f \text{ is a saturated abstract consistency class for Henkin models with primitive equality (cf. Def. 3.18). This entails completeness with the model existence theorem for Henkin models with primitive equality (cf. Theorem 3.29).} \\
\text{First we have to verify that } \mathcal{I}_f \text{ validates the abstract consistency properties } \nabla_\alpha, \nabla_\alpha^* \nabla_\alpha, \nabla_\alpha^*, \nabla_\lambda, \nabla_\lambda^* \nabla_\lambda, \nabla_\lambda^* \nabla_\lambda, \text{ and that } \mathcal{I}_f \text{ is saturated. For all of these cases the proofs are identical to the corresponding argumentations in Theorem 4.16. The only difference is that we employ the lemmata 6.6 and 6.8 instead of 4.12 and 4.15. Thus, all we need to ensure is the validity of the additional abstract consistency properties } \nabla_\tau^* \text{ and } \nabla_\tau^* \text{ for primitive equality.} \\
\n\begin{align*}
\nabla_\tau^\alpha & \quad \neg (A =^\alpha A) \not\in \Phi, \\
\nabla_\tau & \quad \text{If } F[A]_p \in \Phi \text{ and } A = B \in \Phi, \text{ then } \Phi \vdash F[B]_p \in \mathcal{I}_f.
\end{align*}
\]

$(\nabla_\tau^\alpha)$ We have that $[A =^\alpha A]^F \not\vdash^{Triv} \square$, such that $\neg (A =^\alpha A) \not\in \Phi$.

$(\nabla_\tau^*)$ Analogously to 5.27 we show the contrapositive of the assertion. Therefore we assume, that there is derivation $\Delta_1 : \Phi_{cl} \vdash F[B]_p^T \vdash e_{\Phi_{cl}} \square$, and show, that there exists $\Delta_2 : \Phi_{cl} \vdash F[A]_p^T \vdash [A = B]^T \vdash e_{\Phi_{cl}} \square$. Consider the following $E\mathcal{R}E_{fe}$-derivation:

\[
\frac{[F[A]_p]^T \quad [A = B]^T}{[F[B]_p]^T} \quad G_{Para}
\]

By lemma 6.12(2) we know that the generalised paramodulation rule $G_{Para}$ is admissible for calculus $E\mathcal{R}E_{fe}$, such that must be a pure $E\mathcal{R}E_{fe}$ derivation $\Delta_2 : \Phi_{cl} \vdash F[A]_p^T \vdash [A = B]^T \vdash e_{\Phi_{cl}} \square$ that avoids rule $G_{Para}$.

\[ \square \]

\footnote{Note that we here need to employ the conventions of Remark 4.19}
6.5 Theorem Equivalence

We now prove that $\text{ERUE}_c$ and $\text{ERUE}_j$ are theorem equivalent. Theorem equivalence of $\text{ERUE}_c$ and $\text{ERUE}$ is then only presented as a conjecture.

Lemma 6.14 (Proper Clauses in $\text{ERUE}$ and $\text{ERUE}_j$). For each non-proper clause $\mathcal{C}$ and clause set $\Phi$, such that $\Phi \vdash_{\text{ERUE}} \mathcal{C}$, we have that $\Phi \vdash_{\text{ERUE}_j} \mathcal{C}$.

Proof: Analogously to lemmata 4.20 and 5.28. Resolution on unification constraints does not cause additional problems. \qed

As $\Box$ is also a proper clause we immediately get the following corollary:

Corollary 6.15 (Theorem Equivalence of $\text{ERUE}_c$ and $\text{ERUE}_j$). The calculi $\text{ERUE}_c$ and $\text{ERUE}_j$ are theorem equivalent.

Conjecture 6.16 (Theorem Equivalence of $\text{ERUE}$ and $\text{ERUE}_c$ (or $\text{ERUE}_j$)). The calculi $\text{ERUE}$ and $\text{ERUE}_c$ (or $\text{ERUE}_j$) are theorem equivalent.

Again the author expects that the proof for this conjecture will be analogous to the proof of the theorem equivalence for calculi $\text{ER}_c$ and $\text{ER}$ (or $\text{EP}_c$ and $\text{EP}$).
Chapter 7

The LEO-System

In this chapter we present the theorem prover LEO, which realises the extensional higher-order resolution approach $\mathcal{ER}$. LEO is implemented in CLOS [Ste90], a object-oriented extension of LISP, and its input language is POST, which is also used in the $\Omega$MEGA-system [BCF+97].

Despite the theoretical completeness results on $\mathcal{ER}$, the most recent implementation (LEO03) of LEO is not Henkin complete. The main lesson learned from the implementation and the case studies is that developing a theoretically Henkin complete approach to mechanise classical type theory is one thing, but implementing it, such that non-trivial theorems can be proven in practice, is certainly a much more challenging task.

The difference between unification and proof search disappears in extensional higher-order theorem proving and all rules have to be integrated at the same level in order to realise arbitrary calls from the overall proof search to unification and from unification again to the overall proof search. Implementing calculus $\mathcal{ER}$ in such a theoretically Henkin complete but practically naive way is certainly not very complicated, however, the search spaces generated in such a system would be enormous. Therefore LEO employs the extensionality rules only in a restricted way and furthermore guides their application by additional heuristics. This enables LEO to prove simple theorems which require the application of the extensionality principles, such as the theorem discussed in Subsection 7.2.4, which is currently not automatically provable in any other higher-order theorem prover. The drawback is, that these restrictions and heuristics are the main sources of LEO’s incompleteness.

LEO adapts well known ideas and techniques from first-order automated theorem proving and modifies them with respect to the special requirements and aspects of the extensional higher-order resolution approach. More precisely, LEO is based on an extended SOS-architecture that provides two additional stores. The two new constructions are the store of extensionally interesting clauses and the store of unification continuations; the ideas of these concepts will be illustrated in detail below.

The most important feature of the LEO-system is that it employs, like traditional first-order resolution, eager unification as a kind of filter, i.e., LEO tries to get rid of newly generated clauses with a non-solvable unification constraint as early as possible. The main problem thereby is the undecidability of higher-order (pre-)unification and the general need for recursive calls from unification to the general refutation process in order to unify terms also with respect to the extensionality principles. A compromise between completeness and practicability is needed for LEO in order to realise the eager unification idea and hence LEO should be viewed as higher-order theorem prover specialised in extensionality reasoning rather than a general purpose automated theorem-prover for classical type theory.

The LEO-system has been implemented by the author mainly during a stay at Carnegie Mellon University (see [Ben97]). The implementation is still quite prototypical and leaves much room for improvements.

LEO’s main strength is not to compete with other theorem-provers, such as TPS [ABF+96, AINP90], but rather more to complement their weakness in the treatment of the extensionality
principles. TPS, for instance, can in contrast to LEO explore rather deep search spaces and has its strengths, e.g., in the handling of the primitive substitution principles or the handling of definitions [BA98]. *Real mathematics* will only be mechanisable by combining specialised automated higher-order and first-order theorem provers guided and controlled, e.g., by a cognitively more adequate proof planning approach (see [Bun88, CrS98, Mel95, Mel94]) or an intelligent agent mechanism like [BS99, BS98a].

We shall now discuss LEO’s extended SOS-architecture and its main loop in a greater detail. A case study will then be sketched and typical examples that can be solved with the LEO system will be given.

### 7.1 Basic Data-Structures and Algorithms

**KEIM-Toolbox** The implementation of LEO employs the KEIM-toolbox [HKK+94] for deduction systems. This toolbox, which also underlies the mathematical assistant system ΩMEGA [BCF+97], provides many useful data structures (e.g., higher-order terms, literals, clauses, and substitutions) and basic algorithms (e.g., application of substitution, subterm replacement, copying, and renaming). Thus, the usage of KEIM allows for a rather quick implementation of new higher- or first-order theorem proving systems. In addition to the code provided by the KEIM-package, LEO consists of 6900 lines of LISP code. Whereas the KEIM-toolbox indeed turned out to be very useful to support a prototypical development of a system like LEO, the experience with LEO also showed, that the datastructures offered by KEIM are in many cases too extensive and costly. Especially the implementation of LEO’s latest version, LEO3, which employs the fully polymorphic data structures offered by KEIM3 (the latest version of KEIM) is roughly 5-10 times slower than the former version LEO1, which was based on the non-polymorphic data structures of KEIM2.

**Higher-Order Unification** The author (in cooperation with Karsten Konrad) implemented higher-order unification by rather closely realising the rules of [SG89, Sny91]. In all branching cases (branching may only happen in the FlexRigid or FlexFlex case) this algorithm employs the variable binding mechanism offered by the KEIM-package. This mechanism avoids copying of terms and maintains variable instantiations by pointers instead of explicitly instantiating, i.e., modifying, the term data structures. Unfortunately the current implementation does not allow for an explicit maintenance of the variable bindings itself, such that a heuristically guided higher-order unification algorithm cannot be realised in this setting (the aim was to realise a most general, heuristically guided backtracking in the unification search tree, where an user definable heuristics selects — probably in dependence of the current proof goal — the particular partial binding to be examined first when branching in the FlexRigid-case). As KEIM is written in LISP, the implemented unification algorithm can furthermore not exploit concurrency like, e.g., employed in the unification algorithm of LEO’s brother HOT [Kon98], which is written in the programming language Oz [Sh98, SSW94]. HOT translates and realises the main ideas of calculus 8E in a tableaux setting.

**Higher-Order Term indexing** In most state of the art theorem provers for first-order logic, term indexing techniques [Gra95] are successfully employed as strong filters in unification, subsumption, and the computation of resolution partners. The basis for higher-order term indexing has been laid by the masters thesis of Lars Klein [Kle97], supervised by Michael Kohlhase, which adapts first-order term indexing ideas as presented in [Gra95] to λ-terms. The key idea of this adaptation is to employ the simplification part of higher-order unification, which is in contrast to general higher-order unification decidable, as the overall filtering criterion. When sending a request for a given λ-term T to this higher-order term indexing approach, it filters out those terms T’ given in its database, for which pure simplification can not detect a counterargument to its unifiability with T. This obviously realises an imperfect filter, as each delivered T’ may still fail to unify with T. But because of the undecidability of general higher-order unification it is easy to see that a perfect eager unification filter cannot be achieved.
Subsumption  The subsumption filter [Tam98, BHJB92, Fit96], which is well known from resolution based first-order theorem proving, checks whether a given clause \( \mathcal{C} : N_1 \lor \ldots \lor N_m \) is more general than another clause \( \mathcal{D} : M_1 \lor \ldots \lor M_n \). To be more precise, subsumption checks whether there exists a substitution \( \sigma \), such that the particular literals of \( \mathcal{C} \sigma \) are entailed in \( \mathcal{D} \). If so, then the more specialised clause \( \mathcal{D} \) can be removed from the search space without affecting completeness. In practice, many resolution based first-order theorem prover spend most of their computation time in filtering out subsumed clauses in order to minimise the search space. As matching is involved, term indexing techniques are thereby employed to lower the computation costs and to speed up the subsumption tests.

LEO employs a higher-order subsumption test that is, apart from the technical details, very similar to the ones employed in first-order provers. Instead of first-order matching, the criteria for comparing the single literals is higher-order simplification matching, i.e., matching with respect to the deterministic higher-order simplification rules. As matching is known to be decidable up to fourth-order logic [Pad95], it is theoretically possible to develop and employ a much stronger subsumption filter in LEO. This has been avoided so far, as the respective matching algorithms seem to be rather complex. But at least higher-order pattern unification or matching [MI91], which is decidable in linear time [Qia93], should be employed in LEO. Consequently, the higher-order subsumption filter that is currently employed in LEO is — just as the higher-order unification filter — a quite imperfect one, as not all subsumed clauses can be determined in the search space.

But note that even if higher-order matching would be generally decidable for all orders (which is still an open and challenging question), we still could not develop a perfect higher-order subsumption filter for LEO, as this would require a higher-order extensional subsumption approach instead of one that is based solely on syntactical higher-order matching. The problem of extensional higher-order subsumption will be illustrated in detail in Section 7.3.

7.2 Extended SOS Architecture and Main Loop

7.2.1 Problems

The most important aspects and problems for an implementation of the LEO-system are:

1. **Extensional higher-order (pre-)unification** The necessary combination of the traditional syntactical higher-order unification rules with the new extensionality rules\(^1\) \textit{Eque} and \textit{Leib} is probably the most challenging aspect in the implementation of \( \mathcal{ER} \), as a non-restricted application of this rules obviously leads to a search space explosion. But because of the goal directed application of the extensionality principles in \( \mathcal{ER} \), even such a general integration would still have advantages over an unrestricted usage of the infinitely many extensionality rules in the traditional constrained resolution approach (cf. the discussion in Section 1.4). Anyhow, the challenge in LEO is to find suitable heuristics to restrict the application of the extensionality rules in calculus \( \mathcal{ER} \) and to avoid too many recursive calls from within higher-order unification to the overall search process.

2. **Eager unification is essential but undecidable** In contrast to Huet’s original constraint resolution approach, eager unification becomes essential in \( \mathcal{ER} \) and cannot be generally delayed. The reason is, that whenever both extensionality principles are involved in the search for a proof, then calculus \( \mathcal{ER} \) needs to employ recursive calls to the overall search procedure from within a unification attempt, i.e., eager unification is essential in \( \mathcal{ER} \) and LEO. Therefore we need to develop a mechanism, which allows to interrupt eager unification attempts. For example, when reaching a user-definable unification search depth limit the unification process terminates and is resumed later.

3. **Primitive substitution is infinitely branching** The primitive substitution problem is well known in resolution based higher-order theorem proving and is thus not a specific

\(^1\) We here consider the modified rule \textit{Fanz} as belonging to syntactical higher-order unification.
7.2. Extended SOS Architecture and Main Loop

The problem of calculus $\mathcal{ER}$. Informally the problem is that one generally has to ensure that free predicate variables can be instantiated with arbitrary formulae. In order to illustrate the problem, assume that the single unit clause $[P \ a]^F$ (e.g., obtained from the negation and normalisation of the theorem $\exists P, a \ P a$.) is given in the search space, where $P_{\rightarrow a}$ is a free variable. The only way to refute this unit clause is to guess an appropriate instantiation for $P$; in our case $\lambda x_i$, $\sim P^l x$. With this instantiation we get another unit clause $[\sim (P^l a)]^F$, which normalises to $[P^l a]^F$. Now the empty clause can easily be derived by resolving between $[P \ a]^F$ and $[P^l a]^F$.

Fortunately, a blind guess of instantiating formulae can be avoided as it is sufficient to subsequently instantiate the free variable heads with partial bindings imitating the logical connectives in the signature (which in some sense subsequently enumerates all formulas schemes). But as experienced in the TPS-project (cf. [ABF96]) it can nevertheless be very fruitful for some proof attempts to instantiate free variables immediately with special formulæ, such as Leibniz Equality.

It is easy to see that the primitive substitution principle causes an enormous explosion of the search space when applied in an unrestricted and non-guided way.

Apart from these challenging aspects of the implementation of LEO, there are many other, mostly technical problems (see [Ben97]).

7.2.2 Extended SOS Architecture

![Architecture of LEO](image)

Figure 7.1: LEO’s main architecture

LEO’s basic architecture adapts the well known set of support approach (cf. [McC94]) with respect to the special requirements and challenges of calculus $\mathcal{ER}$. The four cornerstones of LEO’s architecture (see the figure 7.1) are:

1. choose lightest from SOS
2. Integrate light. to USABLE
3. resolve with USABLE
4. factorize lightest
5. paramodulate with USABLE
6. primitive substitution on lightest
7. extensionality treatment on EXT
8. pre-unification on CONT
9. process results (tautology deletion)
10. pre-unification on Processed
11. store continuation object
12. check if extensionally interesting
13. Integrate Unified into SOS
USABLE The set of all usable clauses, which initially only contains satisfiable clauses, i.e., the clauses stemming from the assumptions of the theorem to prove (same idea as in first-order resolution [McC94]).

SOS The set of support, which initially only contains the clauses belonging to the negated theorem (same idea as in first-order resolution [McC94]).

EXT The set of all extensionally interesting clauses, i.e., clauses which are assumed to be unifiable, when the new extensionality rules Equi and Leib are taken into account (but probably not by pure higher-order (pre-)unification alone). Initially this set is empty.

CONT The set of all continuations created by the higher-order unification algorithm when reaching the search depth limit in a particular branch of the unification search tree. The idea of this objects is to allow for the continuation of interrupted unification attempts at a later time.

7.2.3 LEO's heuristically guided Main Loop

As a suitable realisation of \( \mathcal{R} \) we suggest a computation consisting of the steps 1–13 as described below (the Initialise step is applied only at the very beginning of the proof attempt and is not part of the main loop). The main loop, whose data-flow is graphically illustrated in Figure 7.1, is executed until an empty clause, i.e., a clause consisting only of flexflex-unification constraints, is detected. The described algorithm is an abstract and slightly idealised presentation of the working principles of the currently actual version (LEO3) of LEO. Furthermore, this description of LEO’s working mechanism abstracts from many technical details and mentions only the most important user definable flags and heuristics that influence its problem solving behaviour.

Initialise The proof problem formalised in POST-syntax is read from a file. Then the specified assumptions and the assertion are pre-classified, i.e., the assumptions become positive unit (pre-)clauses and the assertion becomes a negative unit (pre-) clause. The assumption clauses are passed to USABLE and the assertion clause to SOS. Thereby the SOS store is automatically sorted (with respect to a heuristics H1, that is based on the clauses weights; this weights are computed from the accumulated weights of the term-symbols and from the clause age, i.e., the number of the loop in which the clause has been created).

Step 1 (Choose Lightest) LEO chooses the lightest, i.e., topmost, clause from SOS. If this clause is a pre-clause, i.e., not in proper clause normal-form, then LEO applies clause normalisation to it and integrates the resulting proper clauses into SOS while employing subsumption (depending on the flag-setting forward and/or backward subsumption; see also step 13). Within the clause normalisation process the positive primitive equations are replaced by respective Leibniz equations. Negative primitive equations are not expanded but immediately encoded as extensional unification constraints. Furthermore, identical literals are automatically factorised.

In case the lightest clause was not proper, LEO chooses the next clause from SOS and proceeds with this as described. Otherwise the lightest clause is passed to the store Lightest.

Step 2 (Insert to USABLE) LEO inserts the lightest clause into USABLE while employing forward and/or backward subsumption depending on the flag-setting.

Step 3 (Resolve) The lightest clause is resolved against all clauses in USABLE and the results are stored in Resolved.

Step 4 (Paramodulate) Paramodulation is applied between all clauses in USABLE and the lightest clause, and the results are stored in Paramod. (This step is currently not realised in LEO3)
Step 5 (Factorise) The lightest clause is factorised and the resulting clauses are stored in Factorised.

Step 6 (Primitive Substitution) LEO applies the primitive substitution principle to the lightest clause. The particular logical connectives to be imitated in this step are specified by a flag. The resulting clauses are stored in Prim-Subst.

Step 7 (Extensionality Treatment) The heuristically sorted (with a heuristics H2) store EXT contains extensionally interesting clauses (i.e., clauses with unification constraints that may have additional pre-unifiers, if the extensionality rules are taken into account). LEO chooses the topmost clause and applies the compound extensionality treatment to all extensionally interesting literals of it. Assume the chosen clause has form \( C_1 : D \lor [L_{\alpha \rightarrow \beta} = R_{\alpha \rightarrow \beta}]^F \), where the latter literal is extensionally interesting (determined by a heuristics H3), then the compound extensionality treatment proceeds as follows:

1. First rule Func is applied exhaustively to \( C_1 \). This means that Func is subsequently applied until the unification constraint has form \( \left[ \left( L_{\alpha \rightarrow \beta}, s^{i1}, \ldots, s^{in} \right) = \left( R_{\alpha \rightarrow \beta}, s^{i1}, \ldots, s^{in} \right) \right]^F \), where both hand sides are of primitive type \( \tau \in \{ t, o \} \).
2. If \( \tau \equiv o \), then the compound extensionality treatment applies rule Equiv, and if \( \tau \equiv t \), then rule Leib is applied.

The resulting clauses are stored in Ext-Mod.

Step 8 (Continue Unification) The heuristically sorted (with a heuristics H4) store CONT contains continuations of interrupted higher-order pre-unification attempts from the previous loops (cf. step 10). If the actual unification search depth limit (specified by a flag, whose value can be dynamically increased during proof attempts) allows for a deeper search in the current loop, then the additional search for unifiers will be performed. The resulting, instantiated clauses are passed to Uni-Cont and the new continuations are sorted (wrt. heuristics H4) and integrated into CONT. (This step is currently not realised in LEO3)

Step 9 (Collect Results) In this step LEO collects all clauses, that have been generated within the current loop, from the stores Resolved, Paramod, Factorised, Prim-Subst, Ext-Mod, and Uni-Cont, eliminates obvious tautologies, and stores the remaining clauses in Processed.

Step 10 (Pre-Unify) LEO tries to pre-unify the clauses in Processed. Thus, it applies the pre-unification rules exhaustively, thereby spanning a unification tree until it reaches the unification search depth limit specified by a special flag. The unification search depth limit specifies how many subsequent FlexRigid-branchings may at most occur in each path through the unification search tree. The pre-unified, i.e., instantiated clauses, are passed to Unified. The main idea of this step is to filter out all those clauses with syntactically non-solvable unification constraints (modulo the allowed search depth limit). But note that there are exemptions, which are determined in steps 11 and 12. That means, that not all syntactically non-unifiable clauses are removed from the search space as this would, e.g., also remove the extensionally interesting clauses.

Step 11 (Safe Continuations) Each time a pre-unification attempt in step 10 is interrupted by reaching the unification search depth limit, a respective continuation is created. This object stores the state of the interrupted unification search process, i.e., it contains the particular unification constraints as given at the point of interruption together with the remaining literals of the clause in focus and some information on the interrupted unification process (in principle a continuation is just a new clause with some additional information). Continuations allow to continue the interrupted unification process at any later time. The set of all such continuations is integrated (employing sorting heuristics H4) in the sorted store CONT. (This step is not realised in LEO3.)
CHAPTER 7. THE LEO-SYSTEM

Step 12 (Safe Extensionally Interesting Clauses) In the pre-unification process in step 10 LEO analyses the unification pairs in focus (with heuristics H3) in order to estimate whether this unification constraint and thus this clause is extensionally interesting, i.e., probably solvable with respect to both extensionality principles. The motivation thereby is obvious: we want to avoid recursive calls to overall search procedure for all generated unification pairs and instead employ such calls only selectively. We present two exemplifying criteria for H3: (i) never apply the extensionality treatment to two constants of type $\iota$ that clash (which are quite a lot unification pairs in practice) and (ii) never apply the extensionality treatment to $\text{flex-flex}$-constraints.\footnote{Whereas the criteria (i) and (ii) mentioned here are fair, LEO currently employs additional criteria which are not so obvious and which are most likely not fair, e.g., never apply the extensionality treatment to $\text{flex-flex}$-pairs.} All extensionally interesting clauses are passed to EXT, which is heuristically sorted with a heuristics H5. While integrating the clauses to EXT forward and/or backward subsumption is applied in order to minimise the number of clauses in this store (cf. the discussion of syntactical and extensional subsumption below).

Step 13 (Integrate to SOS) In the last step LEO integrates all pre-unified clauses in Unified into the sorted (with heuristics H1) store SOS. Forward and/or backward subsumption is employed depending on the flag-setting.

7.2.4 An illustrated Example

Let us illustrate LEO’s working with the following example (this example is graphically illustrated in Figure 7.5 in Subsection 7.5.2 and also mentioned as example $E_3^{ext}$ in Section 8.1):

$$ p \land q \Rightarrow p (a \land b) $$

where $p, q, a, b$ are constants. Despite its simplicity, this theorem cannot be solved automatically by any other theorem proving system known to the author (apart from the tableaux based higher-order theorem prover Holy which is based on the tableaux calculus $\text{HTE}$ [Koh98]; $\text{HTE}$ [Koh98] translates the calculus $\mathcal{E}$ presented in this thesis to a tableaux context). Negation and clause normalisation introduces the following three clauses:

$$ \mathcal{C}_1 : [p a] \quad \mathcal{C}_2 : [p b] \quad \mathcal{C}_3 : [p (a \land b)] $$

In Figure 7.5 these clauses are represented by the nodes 5, 7 and 12.

In order to derive the empty clause LEO proceeds as follows: First, it inserts $\mathcal{C}_3$ into SOS, and the two others into USABLE. In the first loop $\mathcal{C}_3$ is chosen as the lightest clause (step 1), inserted into USABLE (step 2), and resolved against $\mathcal{C}_1$ and $\mathcal{C}_2$ (step 3), thereby yielding the clauses $\mathcal{C}_4 : [p (a \land b) = a] \quad \mathcal{C}_5 : [p (a \land b) = b]$. Paramodulation (step 4) is not employed in LEO3 and factorisation (step 5) as well as primitive substitution (step 6) are not applicable. Thus, the clauses passed to Processed (step 7) are $\mathcal{C}_4$ and $\mathcal{C}_5$. The stores EXT (step 8) is still empty and continuation of (pre-)unification (step 9) is not employed in LEO. Next LEO tries to pre-unify both clauses in Processed (step 10). After decomposition the unification process ends up with a clash in both cases: $\mathcal{C}_6 : [(a \land b) = a]$ and $\mathcal{C}_7 : [(a \land b) = b]$ such that no clause is passed to Unified. As continuation of (pre-)unification (step 9) is not employed in LEO no clause is passed to CONT: the maximal unification search depth limit (usually 5 $\text{FlexRigid}$ branching) is not reached in both unification processes. Then the non-unifiable clauses $\mathcal{C}_6$ and $\mathcal{C}_7$ are analysed whether they are extensionally interesting (step 12). As unification constraints between terms of Boolean type are generally classified as extensionally interesting, both clauses are subsequently stored in EXT. Unified is empty and thus no clause is passed to SOS (step 13), such that SOS is empty at the end of the first loop.

Nothing happens and no new clause is generated in the next loops until LEO is allowed to perform its extensionality treatment. The loops in which the extensionality treatment is applied are specified by a flag, and here we assume that the extensionality treatment is employed in every 6th loop. When reaching this loop LEO, e.g., chooses $\mathcal{C}_6$ from EXT (in step 7) and applies its
7.2. EXTENDED SOS ARCHITECTURE AND MAIN LOOP

compound extensionality treatment, thereby generating clause \( C_8 : [(a \land b) \equiv a]^F \), which passes
the unification filter (in step 9) as it contains no further unification constraints, and which is
inserted to Processed. From Processed \( C_8 \) is finally passed (in step 10) to SOS.

In the 7th loop clause \( C_8 \) is chosen from SOS and immediately normalised (step 1), leading to
\( C_9 : \,[a]^T \), \( C_{10} : \,[a]^T \lor \,[b]^T \), and \( C_{11} : \,[a]^F \lor \,[b]^F \). These clause are then integrated to SOS, and as
subsumption is applied, the clause \( C_{10} \) is removed.

In the following two loops LEO subsequently chooses \( C_9 \) and \( C_{11} \) as new lightest clauses, and
proceeds with their processing as described. The only interesting clause generated in these two loops
is \( C_{12} : \,[b]^F \) (by resolution between \( C_9 \) and \( C_{11} \)). This clause is itself integrated to USABLE in one
of the following loops.

In the 12th loop LEO then chooses clause \( C_7 \) from EXT and applies its compound extensionality
treatment to it (in step 7), thereby generating clause \( C_{13} : \,[ (a \land b) \equiv b]^F \), which is passed to SOS
at the end of this loop.

Clause normalisation of \( C_{13} \) at the beginning of loop 13 leads to \( C_{14} : \,[b]^T \), \( C_{15} : \,[a]^T \lor \,[b]^T \) and
\( C_{16} : \,[a]^F \lor \,[b]^F \) both are immediately integrated (modulo subsumption) into SOS. \( C_{14} \) is chosen
as the new lightest clause, and as the complementary clause \( C_{12} \) is already contained in USABLE,
the empty clause is derived in this loop by resolution and (trivial) unification.

7.2.5 Realisation of the Challenges and the main Sources of Incompleteness

Within the current version of LEO the basic challenges mentioned in 7.2.1 are realised as follows:

1. **Extensional higher-order (pre-)unification** Instead of a full integration of the new
extensionality rules into the employed higher-order pre-unification algorithm, we decided to
separate the extensionality treatment from higher-order pre-unification (see steps 7, 10, and
12 above). Thereby the pre-unification filter simply passes extensionality interesting clauses
to the store EXT in order to prevent from being removed from the the search space. The store
EXT is itself heuristically sorted, such that the (heuristically) most interesting clauses (this
notion clearly depends on the quality of the criterions employed by the sorting heuristics
H5) are considered first in the extensionality treatment. This mechanism allows to delay
the extensionality treatment in order to avoid an early search space explosion. It is obvious,
that because of the possibly delay of the extensionality treatment, the sorting heuristics H5
for EXT strongly influences LEO’s overall performance on examples, which indeed require the
application of the extensionality principles. The extensionality treatment itself is exhaustive,
I.e., it is applied to all extensionally interesting literals of the chosen clause from store EXT
in one.

Whereas the original motivation for the separation of the extensionality treatment was to
avoid a complete re-implementation of the pre-unification algorithm provided by the KEIM
package and to examine first in a prototypical implementation of LEO whether the ER
approach is in principle able to solve some simple theorems requiring the extensionality
principles, it turned out that this separation and the possibility to delay the extensionality
treatment is very fruitful if not even essential in practice.

Another interesting experience gained from practice is, that when allowing an extensionality
treatment only in each, e.g., 6th loop, and when restricting primitive substitution to the
logical connective \( \sim \), LEO refutes many of the examples mentioned in Section 7.4 as follows:
In the first 2 or 3 loops LEO performs a couple of interesting resolution, factorisation and
primitive substitution steps. In the following loops he then often produces less interesting
clauses which are often already subsumed by already given ones. Sometimes the SOS even
becomes empty, just like in the example discussed in the previous section. But, meanwhile
the store EXT contains some extensionally interesting clauses and from this LEO chooses then
in the 6th loop the most promising one, applies the extensionality treatment thereby often
generating new interesting facts which then subsequently positively influence the amount of
interesting clauses derived in the following loops. This illustrates that LEO indeed applies the extensionality principles in a goal-directed way, which contrasts with the traditional approaches, as they would — provided that the extensionality axioms are added to the search space — always operate on the extensionality axioms and/or their consequences.

2. **Eager unification is essential but undecidable** The current version of LEO employs higher-order (pre-)unification only with respect to a certain search depth, which is specified by a flag. When reaching this depth LEO simply terminates its search in the current path and backtracks to the next one. It has already been pointed out by other authors (see for instance [Wol93, Pau94, BS94, Nad87]) that non-terminating higher-order unification constraints rarely occur in practice, such that the syntactical unifiability of many clauses can indeed be decided even when restricting the unification search depth to a certain limit. And for the rather simple examples discussed here, a unification search depth of 5 allowed nested branchings with rule *FlexRigid* turns out to be sufficient. But not that LEO’s architecture and main loop already presents with the (currently not realised) steps 8 and 11 a general solution to the problem. This solution consists in the generation of continuations for interrupted unification processes. The generated continuations are heuristically sorted and stored in **CONT**, from where they can be chosen in a later loop of LEO. Thus, the idea is to realise a delayed continuation of interrupted unification processes with respect to a steadily increasing unification search depth in order to overcome the problems induced by the undecidability of higher-order (pre-)unification. The Kiem-toolbox already provides data structures and algorithms for unification continuations. Even though they are not yet employed in LEO — mainly as there was no need for increasing the unification search depth within the currently considered example domains (e.g., simple examples about sets as discussed in Subsection 7.4) and as it seems to be more promising to concentrate on the improvement of other aspects in LEO prototypical implementation first.

3. **The primitive substitution principle is infinitely branching** When using its default settings LEO3 only imitates the logical connective ¬ when applying the primitive substitution principle in step 6. Analogously to the restriction of the unification search depth, this restriction turns out to be sufficient for proving simple examples about sets discussed in Subsection 7.4. Whereas the additional imitation of the logical connective ∨ only slightly influences the time LEO needs to solve these problems, the situation is quite different when imitating also universal quantifiers, e.g., all quantifiers up to order 5.

For more complicated examples it is certainly not sufficient to restrict the primitive substitution rule to the connective ¬ as, e.g., example THM15b discussed in [ABI96] illustrates. The proof of this example in the Trs-system requires the primitive substitution of primitive or Leibniz equality (in LEO the latter would in fact mean to subsequently build up the respective instantiation by single primitive substitution steps imitating ∨ and ¬ in a row).

In order to improve LEO’s treatment of the primitive substitution principle one can introduce a delay mechanism for primitive substitution similar to the stores **EXT** and **CONT**. The idea is to start a proof attempt by, e.g., only imitating ¬ and ∨, and then to subsequently add primitive substitutions of universal quantifiers in a delayed manner.

We briefly summarise the main sources of incompleteness in LEO3, i.e., the current version of LEO:

1. the restricted (pre-)unification depth in step 10 of the main loop,
2. the restricted application of the primitive substitution principles in step 6,

---

3Peter Andrews mentioned that his experience with the Trs-system is similar: Most of the examples examined within the Trs-system can be proven when restricting the unification depth in this sense. A unification depth of more than 15 FlexRigid branchings seems to be rarely needed in practice.

4Note that continuations are basically just ordinary clauses with additional information on the interrupted unification process (e.g., the particular search depth).
3. the heuristics employed for determining extensionally interesting clauses in step 12, which
have not been proven to ensure a Henkin completeness yet,

4. the compound extensionality treatment in step 7, which has not been proven to ensure a
Henkin completeness yet,

5. the heuristics employed to sort the stores SOS, EXT (and CONT) which have not been proven
to ensure fairness yet.

7.3 New Insights gained from LEO

7.3.1 Extensionality and Term indexing

Syntactical higher-order term indexing (see Subsection 7.1) must not be employed in the compu-
tation of resolution partners as otherwise no resolution step on only extensionally but not
syntactically unifiable literals would be suggested.

This drawback is illustrated by the example discussed in Subsection 7.2.4: the key steps in this
refutation — namely the resolution steps between \( C_3 : [p \land q] \) and \( C_1 : [p \land q] \) or \( C_2 : [p \land q] \)
at the very beginning — will not be performed in case syntactical higher-order term indexing is
employed as a filter within the computation of possible resolution partners. Thus, LEO would fail
to prove this theorem.

What we would need in extensional higher-order theorem proving is term indexing modulo the
extensionality principles, which is obviously undecidable.

7.3.2 Extensionality and Subsumption

With respect to our higher-order subsumption filter — we face a problem, similar to the one
sketched for the higher-order term indexing filter above. LEO’s subsumption filter is a quite weak
one, as it only looks for matchable literals on the basis of higher-order simplification, i.e., only
employs first-order matching, when examining whether one clause subsumes another one. Analogous
to the situation in higher-order term indexing, this filter does not take the extensionality principles
into account. Clearly, this does not affect the completeness of our approach and the only effect
of employing a weak subsumption filter when inserting clauses to a clause store is, that probably
much more clauses are inserted as needed to ensure the completeness of the system. Especially the
store EXT is affected, which maintains the extensionality interesting unification constraints. As the
compound extensionality treatment is strongly delayed in LEO, it is consequently very important
to avoid as many as possible subsumed clauses in store EXT. We illustrate this problem by the
following two clauses \( (p, q \land r \land s) \):

\[
C_1 : [[(\lambda X. p \land q) X] = (\lambda X. q \land r \land s) X] \land (\lambda X. r \land q \land s) X] \land (\lambda X. r \land p \land s) X] F
\]

\[
C_2 : [[(\lambda X. p \land q) X \land (\lambda X. q \land r) X] = (\lambda X. r \land p \land s) X] \land (\lambda X. r \land q \land s) X] \land (\lambda X. r \land p \land s) X] F
\]

It is obvious that with respect to a syntactical higher-order matching approach neither one of
both clauses is subsumed by the other. Even though, when applying the compound extensionality
treatment and normalisation to them, they introduce identical proper clauses (modulo the names
of the new Skolem constant \( s \)) into the search space:

\(
C_1 \sim C_3 : [p \land q] F \lor [q \land s] F \lor [r \land s] F \quad C_4 : [p \land q] F \lor [q \land s] F \lor [r \land s] F
\)

\(
C_2 \sim C_5 : [p \land q] F \lor [q \land s] F \lor [r \land s] F \quad C_6 : [p \land q] F \lor [q \land s] F \lor [r \land s] F
\)

The problem is that \( C_1 \) and \( C_2 \) subsume each other only with respect to a notion of extensional
higher-order subsumption, but not with respect to a notion that is only based on syntactical
higher-order matching or simplification. And, as extensional higher-order matching or unification
is undecidable extensional higher-order subsumption seems not to be feasible in practice.
A practical solution that has yet not been examined in LEO, could be to employ the extensionality treatment before inserting a extensionally interesting clause into the store \( EXT \) and to insert the resulting clauses (clauses \( C_3 \ldots C_8 \) in our example) instead of the extensionally interesting clauses itself. One could then try to develop a higher-order subsumption approach that abstracts from the names of the Skolem constants introduced by the extensionality treatment and employ it as a filter for store \( EXT \). In our example, this could prevent LEO from inserting all the clauses \( C_3 \ldots C_8 \) into \( EXT \) and to filter out, for instance, \( C_3, C_4 \) and \( C_5 \). In such a modified approach, the idea of the store \( EXT \) would be to delay and control the insertion of the clauses obtained from the compound extensionality treatment rather than to delay the extensionality treatment of the extensionally interesting clauses itself.

### 7.4 Case Study

**Boolean Properties of Sets** A case study on the examples provided by the article *Boolean Properties of Sets* [TS89] has demonstrated that LEO sometimes outperforms state of the art theorem provers when reasoning about simple examples from set theory. Article [TS89] presents 97 quite trivial theorems about sets. To provide an impression, we present the examples 28, 80, 99, 104 and 111 (the examples in [TS89] are numbered, but not each natural number is associated with an example — this explains the example numbers greater than 97).

**Examples:**

- **set28** If \( X \subseteq Y \) and \( Y \subseteq X \), then \( X = Y \)
- **set80** \((X \cap Y) \cup (X \setminus Y) = X\)
- **set99** \((X \setminus Y) \setminus Z = X \setminus (Y \setminus Z)\)
- **set104** \(X\) misses \(\emptyset\)
- **set111** \((X \cap Y)\) misses \((X \setminus Y)\)

To formalise these theorems and to mechanise their proofs in a first-order theorem prover one has to axiomatise set theory in the system. In a case study carried out in the ILF-project [Dah97], Tarski Grothendieck set theory was employed. The results of this case study are presented in detail at http://www.irm.mathematik.hu-berlin.de/ILF/zesp/mizarstat.html and are summarised by Figure 7.2. We want to point out that examples 99, 104 and 111 could not be proven by any of the employed first-order theorem provers.

Figure 7.2 also relates the ILF case study to the respective case study carried out with LEO. Instead of employing Tarski Grothendieck set theory the above examples were encoded in this case study in classical type theory. This encoding fully exploits the expressiveness of the higher-order language based on the \( \lambda \)-calculus and describes sets by characteristic functions. For instance, the set \( \{ X \mid p \ X \} \) is encoded by the \( \lambda \)-expression \( \lambda X. \ast p \ X \) or simply \( p \) (by \( \eta \)-reduction). Based on this idea one can for instance define the notions \( \subseteq, \cap, \cup, \setminus \), meets and misses as follows:

\[
\begin{align*}
\subseteq_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda X_\alpha. \lambda S_\alpha. S \ X \\
\cap_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \lambda X_\alpha. X \in M \Rightarrow X \in N \\
\cup_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \lambda X_\alpha. X \in M \lor X \in N \\
\setminus_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \lambda X_\alpha. X \in M \land \neg X \in N \\
\setminus N_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \lambda X_\alpha. (M \setminus N) \cup (N \setminus M) \\
\text{meets}_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \exists X_\alpha. X \in M \land X \in N \\
\text{misses}_{\alpha \rightarrow \alpha \rightarrow \alpha} & := \lambda M_\alpha. \lambda N_\alpha. \neg (M \text{ meets } N)
\end{align*}
\]

LEO’s (version 1.0) detailed performance in seconds when solving the theorems of the article *Boolean Properties of Sets* [TS89] is presented in Figure 7.3. The reader may wonder how some
7.5. ADDITIONAL FEATURES OF LEO

problems can be solved without consuming any time. This comes from the fact, that the initial
definition expansion (which is done by ΩMEGA) and initial clause normalisation was not measured
in this case study. And surprisingly some examples, which seem to be hard to solve for the first-
order provers (e.g., set104), can be proven just by normalising the expanded formula. A simple
example is the theorem ∀X, X ⊈ ∅, which expands and normalises to [⊥]T (LEO immediately
detects such contradictory clauses).

LEO was able to prove 95 of the 97 theorems and the only reason why the system cannot prove
theorems 56 and 57 is, that after applying its extensionality treatment in one of the first loops in
the proof attempt, LEO simply generates too many — in their sum contradictory — first-order clauses
to be refuted in its prototypical implementation. By simply combining LEO with a sophisticated
first-order theorem prover like SPASS (e.g. within the OMEGA-system), such that LEO could pass
all the essentially first-order clauses generated by its extensionality treatment to this first-order
reasoner, this two problems should easily be solvable as well. LEO’s task in such a combined system
would be to concentrate on the extensionality aspects and the first-order reasoner would perform
brute-force search on the results in order to check if the extensionality reasoning performed so far
by LEO is already sufficient to ensure the contradiction by straightforward first-order reasoning.

<table>
<thead>
<tr>
<th>Solved theorems (of 97)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waldmeister (pure equality prover, only Th 72 and 99 have been tried)</td>
</tr>
<tr>
<td>SPASS v0.78 (on Ultra SPARC 170)</td>
</tr>
<tr>
<td>Stheo v3.3 (“on” PVM)</td>
</tr>
<tr>
<td>CM v10-15-97 (ME Prover in Prolog)</td>
</tr>
<tr>
<td>CM v10-15-97 (with special cost function [hdef(d1, 1, 6)])</td>
</tr>
<tr>
<td>CM v9-22-97 (with definition expansion in the theorem)</td>
</tr>
<tr>
<td>Otter (auto)</td>
</tr>
<tr>
<td>Gandalf v. c-1.0b</td>
</tr>
<tr>
<td>SPASS v0.54</td>
</tr>
<tr>
<td>Stheo</td>
</tr>
<tr>
<td>All Together</td>
</tr>
<tr>
<td>LEO</td>
</tr>
</tbody>
</table>

Figure 7.2: Case study with LEO on simple example about sets

<table>
<thead>
<tr>
<th>set ex.</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>23</th>
<th>24</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec</td>
<td>.03</td>
<td>.02</td>
<td>.04</td>
<td>.05</td>
<td>.06</td>
<td>.04</td>
<td>.05</td>
<td>1.90</td>
<td>1.89</td>
<td>1.93</td>
<td>.05</td>
<td>.06</td>
<td>5.89</td>
<td>0.32</td>
</tr>
<tr>
<td>set ex.</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>44</td>
</tr>
<tr>
<td>sec</td>
<td>.12</td>
<td>.09</td>
<td>.06</td>
<td>.07</td>
<td>.12</td>
<td>.11</td>
<td>0</td>
<td>.02</td>
<td>.14</td>
<td>.08</td>
<td>.14</td>
<td>.11</td>
<td>.26</td>
<td>2.06</td>
</tr>
<tr>
<td>set ex.</td>
<td>45</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>73</td>
<td>83</td>
<td>57</td>
<td>54</td>
<td>35</td>
<td>33</td>
<td>69</td>
</tr>
<tr>
<td>sec</td>
<td>.07</td>
<td>.08</td>
<td>.03</td>
<td>.04</td>
<td>.20</td>
<td>1.98</td>
<td>.21</td>
<td>.17</td>
<td>.17</td>
<td>.04</td>
<td>1.56</td>
<td>.02</td>
<td>.02</td>
<td>.03</td>
</tr>
<tr>
<td>set ex.</td>
<td>64</td>
<td>65</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
</tr>
<tr>
<td>sec</td>
<td>.13</td>
<td>.03</td>
<td>.14</td>
<td>.04</td>
<td>.05</td>
<td>.17</td>
<td>.17</td>
<td>.57</td>
<td>.98</td>
<td>.93</td>
<td>.92</td>
<td>.91</td>
<td>.92</td>
<td>.93</td>
</tr>
<tr>
<td>set ex.</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
</tr>
<tr>
<td>sec</td>
<td>.08</td>
<td>.18</td>
<td>.09</td>
<td>.09</td>
<td>2.36</td>
<td>.14</td>
<td>.17</td>
<td>.14</td>
<td>.14</td>
<td>.19</td>
<td>.27</td>
<td>.16</td>
<td>.24</td>
<td>.05</td>
</tr>
<tr>
<td>set ex.</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>104</td>
<td>110</td>
<td>111</td>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
</tr>
<tr>
<td>sec</td>
<td>.16</td>
<td>.54</td>
<td>.47</td>
<td>2.03</td>
<td>25.69</td>
<td>.10</td>
<td>.06</td>
<td>0</td>
<td>.24</td>
<td>.02</td>
<td>.03</td>
<td>.07</td>
<td>.18</td>
<td>.50</td>
</tr>
<tr>
<td>set ex.</td>
<td>114</td>
<td>115</td>
<td>115</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>sec</td>
<td>.18</td>
<td>1.67</td>
<td>1.64</td>
<td>25</td>
<td>.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.3: LEO’s performance in seconds on simple examples about sets
<table>
<thead>
<tr>
<th>LEO-Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta-normalise</td>
<td>$\beta$-normalises a clause</td>
</tr>
<tr>
<td>cd</td>
<td>changes the current directory</td>
</tr>
<tr>
<td>decompose</td>
<td>applies the decomposition rule Dec to a clause</td>
</tr>
<tr>
<td>delete-clause</td>
<td>deletes a clause from the current environment</td>
</tr>
<tr>
<td>end-report</td>
<td>closes the report stream</td>
</tr>
<tr>
<td>execute-log</td>
<td>subsequently executes the commands in a report file</td>
</tr>
<tr>
<td>exit</td>
<td>quits the current interpreter session</td>
</tr>
<tr>
<td>exit</td>
<td>applies the compound extensionality treatment to a clause</td>
</tr>
<tr>
<td>factorise</td>
<td>applies the factorisation rule Fac to a clause</td>
</tr>
<tr>
<td>help</td>
<td>lists a short description of all commands</td>
</tr>
<tr>
<td>imitate</td>
<td>applies the FlexRigid rule to a clause (only with imitation bindings)</td>
</tr>
<tr>
<td>new-log</td>
<td>opens a new report-file</td>
</tr>
<tr>
<td>para</td>
<td>applies the paramodulation rule Para to two clauses</td>
</tr>
<tr>
<td>pre-unifiers</td>
<td>computes and displays the syntactical pre-unifiers of a clause</td>
</tr>
<tr>
<td>pre-unify</td>
<td>pre-unifies a clause</td>
</tr>
<tr>
<td>prim-subst</td>
<td>applies the primitive substitution Prim rule to a clause</td>
</tr>
<tr>
<td>proj-imi</td>
<td>applies the FlexRigid rule to a clause (only with projection bindings)</td>
</tr>
<tr>
<td>project</td>
<td>calls LEO in automatic mode to the current system state</td>
</tr>
<tr>
<td>prove</td>
<td>reads a problem file containing a proof problem in POST syntax,</td>
</tr>
<tr>
<td></td>
<td>applies pre-classification and stores the clauses either in the</td>
</tr>
<tr>
<td></td>
<td>set of support or the set of usable clauses</td>
</tr>
<tr>
<td>resolve</td>
<td>applies the resolution rule Res to two clauses</td>
</tr>
<tr>
<td>set-flag</td>
<td>modifies the setting of a global flag</td>
</tr>
<tr>
<td>set-tactic</td>
<td>modifies the setting of a global tactic</td>
</tr>
<tr>
<td>show-clause</td>
<td>displays a clause</td>
</tr>
<tr>
<td>show-clauses</td>
<td>displays the content of clause stores</td>
</tr>
<tr>
<td>show-derivation</td>
<td>displays a linearised derivation of a clause</td>
</tr>
<tr>
<td>show-flags</td>
<td>displays the settings of all global flags</td>
</tr>
<tr>
<td>show-problem</td>
<td>displays the given problem (POST input)</td>
</tr>
<tr>
<td>show-proof</td>
<td>displays a found proof in linearised form</td>
</tr>
<tr>
<td>show-tactics</td>
<td>displays the available tactics (heuristics)</td>
</tr>
<tr>
<td>show-vars</td>
<td>displays the settings of all global variables</td>
</tr>
<tr>
<td>step-log</td>
<td>interactsively executes the commands stored in a log file</td>
</tr>
<tr>
<td>subsumes</td>
<td>determines whether a clause subsumes another clause</td>
</tr>
<tr>
<td>write-derivation</td>
<td>writes a derivation of a clause in a file</td>
</tr>
<tr>
<td>write-proof</td>
<td>writes a proof in LOUT format in a file</td>
</tr>
<tr>
<td>write-proof</td>
<td>writes a proof in a file</td>
</tr>
</tbody>
</table>

Figure 7.4: Some interactive commands provided by the LEO-system
7.5 Additional Features of LEO

7.5.1 The Interactive Theorem Prover LEO

LEO can also be employed as an interactive theorem prover for extensional higher-order resolution. The main idea thereby is to provide a system that enables the user to analyse and illustrate the working principles of extensional higher-order resolution step by step. Especially in LEO's first implementation stages the interactive features turned out to be a very useful tool for experimenting with the calculus $\mathcal{ER}$. Figure 7.4 presents the most important interactive commands of LEO.

7.5.2 LOUI as a Graphical Interface for LEO

LOUI is a generic graphical user interface developed by Stephan Hess within his masters thesis [Hes99] and it is described in detail in [SHB+98, SHB+99]. Among the systems which already employ LOUI as an interface or which are currently connected to it are: the mathematical assistant $\Omega$MEGA [BCF+97], the induction theorem prover INKA [AHMS99, HS96], the higher-order proof planner $\lambda$CLAM [RSG98], and the LEO system described in this thesis. Apart from other useful information, LOUI displays a proof graph (more precisely, a proof tree with co-references to already given nodes in the tree) and a linearised proof protocol.

LEO's connection to LOUI allows for the graphical presentation of extensional higher-order resolution derivations and proofs with the gain for the user, that the structure of interactively or automatically created derivations become much more transparent to him.

Figure 7.5 presents LOUI's visualisation for the proof of the small but challenging example $(p_{0 \rightarrow 0} a_0 \land p_{0 \rightarrow 0} b_0 \Rightarrow p_{0 \rightarrow 0} (a_0 \land b_0))$ discussed in Subsection 7.2.4. Figure 7.6 presents another simple example about sets, which states that the power set of the empty set is the set that contains only the empty set.

In the LOUI interface triangles represent initially given clauses stemming either directly from the assumptions or from the negated theorem. Derived clauses are presented as round nodes with links to their ancestors and their successors. Rhombi symbolise co-references to already displayed sub-derivations in order to avoid their duplication in the display.

7.5.3 Integration to $\Omega$MEGA

The LEO prover is integrated into the mathematical assistant system $\Omega$MEGA [BCF+97], and the main aspects of this integration are:

- LEO and $\Omega$MEGA rely on the same data structures and run in a single LISP process. Hence, LEO can be viewed as $\Omega$MEGA's logical engine, which can be employed to solve minor sub-problems and to support $\Omega$MEGA's proof planner [CrS98, Me95, Me94].

- LEO can be employed as an integrated higher-order theorem prover just like any other of the (currently) 10 external systems, that have been integrated to $\Omega$MEGA.

- LEO can be employed as an interactive theorem prover for extensional higher-order resolution in $\Omega$MEGA. This is supported by the LOUI-interface, which provides menu entries supporting an easy selection for each of LEO's interactive commands (cf. Figure 7.4).

Within the $\Omega$MEGA system, LEO and TPS [ABI+96, AINP90], which is the only other higher-order theorem prover integrated to $\Omega$MEGA (the integration of TPS and $\Omega$MEGA is described in [BBS99, BS98b]) complement each other. The TPS system can explore rather deep search spaces and employs a clever mechanism for selectively expanding definitions [BA98]. But the extensionality treatment in TPS is rather weak (e.g., TPS can not solve our little problem illustrated in Figure 7.5 and Subsection 7.2.4).
EXAMPLE: $p_{0 \rightarrow 0} a_0 \land p_{0 \rightarrow 0} b_0 \Rightarrow p_{0 \rightarrow 0} (a_0 \land b_0)$

Figure 7.5: LOUT-Visualisation of LEO’s proof for example $E_3^{st}$
7.5. ADDITIONAL FEATURES OF LEO

**Example:** \( \emptyset(\emptyset) = \{\emptyset\} \)

DEF.:

\[ \emptyset := \lambda A_{a \rightarrow a} \lambda B_{a \rightarrow a} \forall x X_{a} \land B \implies A \land X \]

\[ \emptyset := \lambda Y_{a} \land \lambda Y_{b} \land Y \land X \]

\( Y := \lambda Y_{a} \land F_{a} \land X \)

To show:

\[ \neg (\lambda B_{a \rightarrow a} \land (\forall X_{a} \land B \land X \implies \bot)) = \]

\[ (\lambda B_{a \rightarrow a} \land B = \lambda Y_{a} \land \bot)) = \]

\[ (\lambda B_{a \rightarrow a} \land (\forall X_{a} \land (B \land X \implies \bot)) = \]

\[ (\lambda B_{a \rightarrow a} \land B = \lambda Y_{a} \land \bot)) = \]

1. \[ \square \]
2. \[ e_{a \rightarrow o}^{1} X_{o}^{a} = e_{a \rightarrow o}^{1} s^{k_{1}} \]  \[ F \]
3. 4. \[ e_{a \rightarrow o}^{1} X_{a}^{2} \]  \[ F \]
5. \[ e_{a \rightarrow o}^{1} X_{a}^{2} = e_{a \rightarrow o}^{1} s^{k_{1}} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} X_{a}^{h}] F \]
6. \[ e_{a \rightarrow o}^{1} X_{a}^{2} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} X_{a}^{h}] F \]
7. \[ e_{a \rightarrow o}^{1} X_{a}^{2} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} X_{a}^{h}] F \]
8. 9. \[ (\lambda B_{a \rightarrow a} \land (\forall X_{a} \land (B \land X \implies \bot)) = \]
10. 11. \[ e_{a \rightarrow o}^{1} s^{k_{1}} \]  \[ F \]
12. \[ e_{a \rightarrow o}^{1} e_{a \rightarrow o}^{2} \]  \[ X_{a}^{2} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} s^{k_{1}}] F \]
13. \[ e_{a \rightarrow o}^{1} e_{a \rightarrow o}^{2} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} s^{k_{1}}] F \]
14. \[ e_{a \rightarrow o}^{1} = e_{a - o}^{1} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} s^{k_{1}}] F \]
15. \[ e_{a \rightarrow o}^{1} X_{a}^{2} \]  \[ F \]  \[ \lor \]  \[ [e_{a \rightarrow o}^{1} s^{k_{1}}] F \]

**Figure 7.6:** Lout-Visualisation of LEO’s proof for example \( E_{o}^{ext} \).
Chapter 8

Examples

This chapter presents various examples that illustrate the basic ideas of the calculi $\mathcal{ER}$, $\mathcal{EP}$, and $\mathcal{ERhE}$ and compare refutations in the different calculi with each other. These examples especially demonstrate the importance of an appropriate extensionality treatment in higher-order theorem proving.

8.1 Extensionality in $\mathcal{ER}$

This section discusses five simple, but with respect to their mechanisation nevertheless challenging examples for an higher-order automated theorem prover. As already mentioned in Subsection 7.2.4, example $E_5^{xt}$ is currently not automatically provable in any higher theorem prover apart from Leo. Note that all refutations are quite trivial and straightforward in the extensional higher-order resolution approach $\mathcal{ER}$.

$E_1^{xt}$ $a_o \equiv b_o \Rightarrow (\forall p_{o\rightarrow o} \cdot Pa \Rightarrow Pb)$

The non-trivial direction of the extensionality property for truth values: if $a_o$ is equivalent to $b_o$, then $a_o$ is Leibniz equal to $b_o$. In the following refutation $p_{o\rightarrow o}$ is a new Skolem constant.

$$
\begin{align*}
Cnf(\neg E_1^{xt}) : & \quad C_1 : [p \cdot a]^T & \quad C_2 : [p \cdot b]^T & \quad C_3 : [a]^T \lor [b]^T & \quad C_4 : [b]^T \lor [a]^T \\
Res(C_1, C_2) : & \quad C_5 : [p \cdot a \equiv p \cdot b]^F & \quad C_6 : [a \equiv b]^F \\
Dec(C_3), Triv : & \quad C_7 : [a]^T \lor [b]^T & \quad C_8 : [a]^T \lor [b]^T
\end{align*}
$$

The rest of the refutation with clauses $C_3, C_4, C_7,$ and $C_8$ is obvious.

$E_2^{xt}$ $p_{o\rightarrow o} \cdot (a_o \land b_o) \Rightarrow p (b \land a)$

Any property which holds for $a \land b$ also holds for $b \land a$. This statement can also be read as whenever $a \land b$ is in a set of truth values $p_{o\rightarrow o}$, then also $b \land a$ is in $p_{o\rightarrow o}$.

$$
\begin{align*}
Cnf(\neg E_2^{xt}) : & \quad C_1 : [p \cdot (a \land b)]^F & \quad C_2 : [p \cdot (b \land a)]^F \\
Res(C_1, C_2) : & \quad C_3 : [p \cdot (a \land b) \equiv p \cdot (b \land a)]^F & \quad C_4 : [a \land b \equiv b \land a]^F \\
Dec(C_4), Triv : & \quad C_5 : [a]^F \lor [b]^F & \quad C_6 : [a]^T \lor [b]^T & \quad C_7 : [a]^T & \quad C_8 : [b]^T
\end{align*}
$$

The rest of the refutation is obvious: resolve $C_5$ against $C_7$ and $C_8$.

$E_3^{xt}$ $(p_{o\rightarrow o} \cdot a_o \land p \cdot b_o) \Rightarrow p (b \land a)$

If the truth values $a_o$ and $b_o$ are in set $p_{o\rightarrow o}$, then $a_o \land b_o$ is in $p_{o\rightarrow o}$ as well.
8.1. Extensionality in ER

\[
\begin{align*}
\text{Cnf}(\neg \mathbf{E}^{\text{ext}}_3) : & \quad C_1 : [p \ a]^T \\
\text{Res}(C_3, C_1) : & \quad C_4 : [p \ (a \land b) = p \ a]^F \\
\text{Res}(C_3, C_2) : & \quad C_5 : [p \ (a \land b) = p \ b]^F \\
\text{Dec}(C_4), \text{Triv} : & \quad C_6 : [a \land b = a]^F \\
\text{Dec}(C_5), \text{Triv} : & \quad C_7 : [a \land b = b]^F \\
\text{Equiv}(C_6), \text{Cnf} : & \quad C_8 : [a]^F \lor [b]^F \\
\text{Equiv}(C_7), \text{Cnf} : & \quad C_9 : [a]^T \lor [b]^T \\
& \quad \quad C_{10} : [a]^T \\
& \quad \quad C_{11} : [a]^F \lor [b]^F \\
& \quad \quad C_{12} : [a]^T \lor [b]^T \\
& \quad \quad C_{13} : [b]^T
\end{align*}
\]

The rest of the refutation is obvious: Resolve \(C_8\) against \(C_{10}\) and \(C_{13}\).

\[
\mathbf{E}^{\text{ext}}_4 \quad (\forall X, \forall P, \forall P (m \rightarrow X) \Rightarrow P (n \rightarrow X)) \\
\quad \Rightarrow (\forall Q (\forall \lambda \mapsto \pi), Q (\lambda X, m X) \Rightarrow Q (\lambda X, n X))
\]

This formula is an instance of the \(\xi\)-rule \((\forall X, \forall X, X \equiv n \mapsto X) \Rightarrow (\lambda X, m X) \equiv (\lambda X, n X)\) for Leibniz equality; for details on the \(\xi\)-rule, e.g., see [HS86].

\[
\begin{align*}
\text{Cnf}(\neg \mathbf{E}^{\text{ext}}_4) : & \quad C_1 : [P (m X)]^F \lor [P (n X)]^T \\
& \quad C_2 : [q (\lambda X, m X)]^F \\
& \quad C_3 : [q (\lambda X, n X)]^F
\end{align*}
\]

Unfortunately the idea to resolve \(C_2\) and \(C_3\) immediately against \(C_1\) does not lead to successful refutation as the resulting unification constraints are not solvable. Therefore we choose another way and resolve between \(C_2\) and \(C_3\) (in the following derivation \(p \rightarrow \pi, q (\rightarrow \pi) \rightarrow \alpha, s_i\) are new Skolem constants):

\[
\begin{align*}
\text{Res}(C_2, C_3) : & \quad C_4 : [q (\lambda X, m X) = q (\lambda X, n X)]^F \\
\text{Dec}(C_4), \text{Triv} : & \quad C_5 : [m s = n s]^F \\
\text{Func}(C_5) : & \quad C_6 : [m s = n s]^F \\
\text{Leib}(C_6), \text{Cnf} : & \quad C_7 : [p (m s)]^T \\
& \quad C_8 : [p (n s)]^F
\end{align*}
\]

We made a detour to the pre-unification part of the calculi and modified the clauses \(C_2\) and \(C_3\) in an extensionally appropriate way and \(C_2\) and \(C_3\) have now their counterparts in \(C_7\) and \(C_8\). But in contrast to \(C_2\) and \(C_3\) the new clauses can successfully be resolved against \(C_1\).

In our refutation \(q (\rightarrow \pi) \rightarrow \alpha, s_i, p \rightarrow \pi\) are new Skolem constants.

\[
\mathbf{E}^{\text{ext}}_5 \quad (\forall X, \forall P, \forall P (m \rightarrow X) \Rightarrow P (n \rightarrow X)) \Rightarrow (\forall Q (\forall \lambda \mapsto \pi), Q m \Rightarrow Q n)
\]

This is an instance of the non-trivial direction of the functional extensionality axiom for type \(\tau \rightarrow \tau\) (in the following derivation \(p \rightarrow \pi, q (\rightarrow \pi) \rightarrow \alpha, s_i\) are new Skolem constants):

\[
\begin{align*}
\text{Cnf}(\neg \mathbf{E}^{\text{ext}}_5) : & \quad C_1 : [P (m X)]^F \lor [P (n X)]^T \\
& \quad C_2 : [q m]^F \\
& \quad \quad C_3 : [q n]^F \\
\text{Res}(C_2, C_3) : & \quad C_4 : [q m = q n]^F \\
\text{Dec}(C_4), \text{Triv} : & \quad C_5 : [m n]^F \\
\text{Func}(C_5) : & \quad C_6 : [m s = n s]^F \\
\text{Leib}(C_6), \text{Cnf} : & \quad C_7 : [p (m s)]^T \\
& \quad \quad C_8 : [p (n s)]^F
\end{align*}
\]

The rest of the refutation is (like above) obvious: resolve \(C_7\) and \(C_8\) against \(C_1\).

In our refutation \(q (\rightarrow \pi) \rightarrow \alpha, s_i, p \rightarrow \pi\) are new Skolem constants.

\[
\mathbf{E}^{\text{ext}}_6 \quad p(\emptyset) = \{\emptyset\}
\]
A (similar) proof for this example in calculus $\mathcal{ER}$ is also illustrated by Figure 7.6.

$Cnf(\neg E^{\tau\tau})$:

$\begin{align*}
C_1 & : [\lambda B_{\alpha\tau\eta} \forall X_{\alpha\tau\eta} \lambda Y_{\alpha\tau\eta} (B X) \Rightarrow \bot)]^T \\
Cnf(C_1), Equiv, Cnf & : \begin{cases}
C_2 : [\epsilon_1 X]^T \lor [\epsilon_1 = \lambda Z_{\alpha\tau\eta} \bot]^T \\
C_3 : [\epsilon_1 s]^T \lor [\epsilon_1 = \lambda Z_{\alpha\tau\eta} \bot]^T \\
C_4 : [\epsilon_1 s]^T \lor [\epsilon_1 X]^T \\
\end{cases}
\end{align*}$

$Func(C_2), Equiv, Cnf f$:

$\begin{align*}
C_5 & : [\epsilon_1 X]^T \lor [\epsilon_1 Y]^T \\
Func(C_5), Unif : & \begin{cases}
C_6 : [\epsilon_1 X]^T \\
C_7 : C_3 : [\epsilon_1 s]^T \lor [\epsilon_1 e_2]^T \\
Res(C_4, C_2), Fac, Unif : & \begin{cases}
C_8 : [\epsilon_1 s]^T \\
Res(C_5, C_8) : \emptyset
\end{cases}
\end{cases}
\end{align*}$

8.2 Decomposition in $\mathcal{ER}$

Example $E^{Dec}$ below demonstrates the basic ideas of extensional higher-order resolution and illustrates why this approach can also be seen as test calculus for extensional higher-order $E$-unification. This example furthermore focuses on the role of the decomposition rule in connection with the extensionality rules and compares the slightly modified decomposition rule $Dec$ employed in this thesis with the rule $Dec'$ as used in $[BK98a]$:

$$
\frac{C \lor [h \ U^n = h \ V^n]^F}{C \lor \bigvee [U^1 = V^1]^F \lor \ldots \lor [U^n = V^n]^F} \quad Dec'
$$

Our example extends $E^{\tau\tau}$ from the previous section: suppose we have four function constants, $f_{(a \to a)} \to a \to a$, $g_{(a \to a)} \to a \to a$, $h_{a \to a}$, and $j_{a \to a}$, and we know that $f$ equals $g$ and $h$ equals $j$ (in the $E$-unification perspective we can assume that this two equations define our theory $E$). We want to prove that under this assumptions (under this theory) application $(f \ h)$ equals application $(g \ j)$. Depending on the actual encoding of the assumptions and the assertion, a respective proof is either trivial or quite complex to mechanise in classical type theory. A formulation that is not trivial to mechanise (as the application of the extensionality principles is required) and which uses Leibniz equality is:

$$
E^{Dec} \quad \forall X_{\alpha \to a} \forall Y_{\alpha \to a} (f X Y) \equiv (g X Y) \land (\forall Z_{\alpha \to a} (h Z) \equiv (j Z)) \Rightarrow (f \ h) \equiv (g \ j)
$$

When expanding the definition of $\equiv$, negating the theorem and applying pre-classification we obtain the following pre-clauses for this example:

$\begin{align*}
C_1 & : \forall P_{\alpha \to a} \forall X_{\alpha \to a} \forall Y_{\alpha \to a} \neg (P (f X Y)) \lor (P (g X Y)))^T \\
C_2 & : \forall Q_{\alpha \to a} \forall Z_{\alpha \to a} \neg (Q (h Z)) \lor (Q (j Z)))^T \\
C_3 & : \neg (\forall R_{\alpha \to a} \to a \to a) \neg (R (f \ h)) \lor (R (g \ j)))^T
\end{align*}$

Clause normalisation leads to:

$\begin{align*}
C_4 & : [P (f X Y)]^T \lor [P (g X Y)]^T \\
C_5 & : [Q (h Z)]^T \lor [Q (j Z)]^T \\
C_6 & : [r (f \ h)]^T \\
C_7 & : [r (g \ j)]^T
\end{align*}$

The reader may check that resolving $C_6$ and $C_7$ against $C_5$ does not lead to successful refutation (c.f. discussion of example $E1$ in $[BK98a]$). Instead we resolve between $C_6$ and $C_7$ and proceed by employing the difference reduction idea.

The first refutation we present here employs rule $Dec$. Below we will present a second refutation.
that alternatively use rule $Dec'$:

\[
Res(C_6, C_7) : C_8 : \forall (f \ h) = r (g \ j)^F
\]

\[
Dec'(C_8) : C_9 : \forall (f \ h) = r (g \ j)^F
\]

\[
\begin{align*}
Cnf(C_12) : C_{13} & : [p (f s)]^T \lor [q (h u)]^T \\
Cnf(C_16) & : C_{14} : [p (f s)]^T \lor [q (j u)]^T
\end{align*}
\]

Within this derivation $t_{a \rightarrow a}, s_a, u_a, p_{a \rightarrow a}, q_{a \rightarrow a}$ are new Skolem terms and $P'_{a \rightarrow a}, Q'_{a \rightarrow a}$ are new predicate variables. The rest of the refutation employs straightforward resolution between the clauses $C_3-C_{16}$ and our assumption clauses $C_4, C_5, C_{17}$ and $C_{18}$.

A alternative refutation that uses the rule $Dec'$ is a bit more tricky:

\[
Res(C_6, C_7) : C_8 : \forall (f \ h) = r (g \ j)^F
\]

\[
Dec'(C_8) : C_9 : \forall (f \ h) = g (j s)^F
\]

\[
Func(C_9) : C_{10} : \forall (f h s = g j s)^F
\]

\[
Leib(C_{10}) : C_{11} : \forall \exists P_{a \rightarrow a} \neg(P (f h s)) \lor \forall (P (g j s))^F
\]

\[
\begin{align*}
Cnf(C_11) : C_{12} & : [p (f s)]^T \\
Cnf(C_15) & : C_{13} : [p (g j s)]^F
\end{align*}
\]

\[
Res(C_2, C_4) : C_{14} : [P (f X Y)]^T \lor [p (f h s) = P (f X Y)]^F
\]

\[
Res(C_3, C_4) : C_{15} : [P' (f X Y)]^T \lor [p (g j s) = P' (g X Y)]^F
\]

Note that the unification constraints in $C_{14}$ and $C_{15}$ are solvable by first imitating $p$ and then either projecting to the argument of $P$ or imitating to the argument of $p$. In our refutation we choose for $C_{14}$ the partial projection solution $\sigma_1 := [(\lambda X_{a \rightarrow a} p) X] / [P, h / X, s / Y]$ and for $C_{15}$ the partial imitation solution $\sigma_2 := [(\lambda X_{a \rightarrow a} p) (g j s)] / [P']$. Both solutions can be computed in $\mathcal{E}R$ by applying eager unification. We propagate these solutions back to the non-unification literals of the clauses and proceed as follows:

\[
\begin{align*}
UN(C_4) : C_{16} & : [p (g h s)]^T \\
UN(C_5) & : C_{17} : [p (g j s)]^F
\end{align*}
\]

\[
Res(C_{16}, C_{17}) : C_{18} : [p (g h s) = p (g j s)]^F
\]

\[
2 \times Dec'(C_{18}) : C_{19} : [h s = j s]^F
\]

\[
Leib(C_{10}) : C_{20} : [\exists Q_{a \rightarrow a} \neg(Q (h s)) \lor (Q (j s))]^F
\]

\[
\begin{align*}
Cnf(C_20) : C_{21} & : [q (h s)]^T \\
Cnf(C_24) & : C_{22} : [q (j s)]^F
\end{align*}
\]

\[
Res(C_{21}, C_5) : C_{23} : [Q (j Z)]^T \lor [q (h s) = Q (j Z)]^F
\]

\[
Res(C_{22}, C_5) : C_{24} : [Q' (h Z')]^T \lor [q (j s) = Q' (j Z')]^F
\]

\[
Res(C_{23}, C_{24}) : [Q (j Z) = Q' (h Z')]^T \lor [q (h s) = Q (j Z)]^F \lor [q (j s) = Q' (j Z')]^F
\]

With unifier $\sigma := [(\lambda X_{a \rightarrow a} p (h s))/Q, (\lambda X_{a \rightarrow a} q) X]/Q', s / Z]$ this unification constraint is obviously solvable, such that the empty clause is obtained by applying pre-unification.

This discussion raises the question whether rule $Dec$ or rule $Dec'$ is better suited within calculus $\mathcal{E}R$. This problem can and should be clarified by experiments in practice. As an alternative to $\mathcal{E}R$ and in order to simplify the formal completeness proofs at certain points we decided in this paper to use $Dec$.

### 8.3 Leibniz Equality and Alternative Definitions in $\mathcal{E}R$

The examples discussed in this section focus on the equivalence of Leibniz equality and alternative definitions for equality in classical type theory.

The definitions for equality that are compared are:
Leibniz Equality

$$\equiv^\alpha := \lambda X_{\alpha}, \lambda Y_{\alpha}. \forall P_{\alpha \rightarrow \alpha} . PX \Rightarrow PY$$

Reflexivity Definition

$$\equiv^\alpha := \lambda X_{\alpha}, \lambda Y_{\alpha}. \forall Q_{\alpha \rightarrow \alpha \rightarrow \alpha} . (\forall Z_{\alpha} . (Q Z Z)) \Rightarrow (Q X Y)$$

Modified Leibniz Equality

$$\equiv^\alpha := \lambda X_{\alpha}, \lambda Y_{\alpha}. \forall P_{\rightarrow \rightarrow \rightarrow} . ((a_o \lor \neg a_o) \land P X) \Rightarrow ((b_o \lor \neg b_o) \land P Y)$$

Leibniz equality employs the substitutivity principle to define equality, whereas the second alternative definition, which is presented and discussed in Andrews textbook [And86] at page 155, employs the reflexivity principle. The third (artificial) definition illustrates that there are infinitely many modifications of Leibniz equality (and analogously of the Reflexivity Definition) which all denote the same relation in Henkin semantics, namely equality. The examples furthermore illustrates that it is impossible to decide whether a given formula denotes the equality relation as this requires to decide whether two arbitrary formulae are equivalent.

E\(_{1}^\alpha\) (i) \((u \equiv^\alpha v) \Rightarrow (u \equiv^\alpha v)\) and (ii) \((u \equiv^\alpha v) \Rightarrow (u \equiv^\alpha v)\)

In case (i) we use the following refutation (\(q_{\alpha \rightarrow \alpha \rightarrow \alpha}\) is a new Skolem constant):

$$\text{CNF}(i) : \quad \begin{align*}
C_1 : & [P u]F \lor [P v]T \\
C_2 : & [q Z Z]T \\
C_3 : & [q u v]F
\end{align*}$$

Res\((C_1, C_3) : \quad \begin{align*}
C_4 : & [P u]F \lor [P v = q u v]F
\end{align*}$$

\(\text{UNP}(C_4)\) with \([(\lambda X_{\alpha}, q u X) / P]\) : \quad \begin{align*}
C_5 : & [q u v]F
\end{align*}

Res\((C_2, C_5), \text{Triv} : \quad \square$$

In case (ii) we proceed as follows (\(p_{\alpha \rightarrow \alpha \rightarrow \alpha}\) is a new Skolem constant):

$$\text{CNF}(i) : \quad \begin{align*}
C_1 : & [Q z z]F \lor [Q u v]T \\
C_2 : & [p u]T \\
C_3 : & [p v]F
\end{align*}$$

Prim\((C_1), \text{Subst} : \quad \begin{align*}
C_4 : & [Q' z z]T \lor [Q' u v]F
\end{align*}$$

Res\((C_1, C_5) : \quad \begin{align*}
C_5 : & [Q z z]F \lor [Q u v = p v]F
\end{align*}$$

\(\text{UNP}(C_5)\) with \([(\lambda X_{\alpha}, q u X) / P]\) : \quad \begin{align*}
C_6 : & [p z]F
\end{align*}

Res\((C_2, C_6), \text{Triv} : \quad \square$$

Alternatively one can directly prove \(\equiv^\alpha = \equiv^\alpha\), but the proof is a little more bulky.

E\(_{2}^\alpha\) (i) \((u \equiv^\alpha v) \Rightarrow (u \equiv^\alpha v)\) and (ii) \((u \equiv^\alpha v) \Rightarrow (u \equiv^\alpha v)\)

In case (i) we proceed as follows:

$$\text{CNF}(i) : \quad \begin{align*}
C_1 : & [P u]F \lor [P v]T \\
C_2 : & [p u]T \lor [u]T \\
C_3 : & [p u]T \lor [u]F \\
C_4 : & [p v]F \lor [a]T \\
C_5 : & [p v]F \lor [a]F
\end{align*}$$

Res\((C_2, C_3), F_{ac,2 \times \text{Triv} : \quad \begin{align*}
C_6 : & [p u]T \\
\text{Res}(C_3, C_5), F_{ac,2 \times \text{Triv} : \quad \begin{align*}
C_7 : & [p v]F
\end{align*}

Res\((C_1, C_6) : \quad \begin{align*}
C_8 : & [p v]F \lor [P u = p u]F
\end{align*}

Res\((C_8, C_7) : \quad \begin{align*}
\end{align*}

\(\text{UNP}(C_9)\) with \([p / P]\) : \quad \square$$
In case (ii) we employ the following derivation:

\[
\begin{align*}
\text{CNF}(i) : \\
C_1 & : [a] F \lor [P \ u] F \lor [b] F \lor [P \ v] T \\
C_2 & : [a] F \lor [P \ u] F \lor [b] T \lor [P \ v] T \\
C_3 & : [a] T \lor [P \ u] F \lor [b] F \lor [P \ v] T \\
C_4 & : [a] T \lor [P \ u] F \lor [b] T \lor [P \ v] T \\
C_5 & : [P \ u] T \\
C_6 & : [P \ v] F \\
Res(C_1, C_2), 3 \times Fac, 4 \times Triv : \\
C_7 & : [a] F \lor [P \ u] F \lor [P \ v] T \\
Res(C_3, C_4), 3 \times Fac, 4 \times Triv : \\
C_8 & : [P \ u] F \lor [P \ v] T \\
Res(C_7, C_8), 2 \times Fac, 3 \times Triv : \\
C_9 & : [P \ u] F \lor [P \ v] T \\
Res(C_9, C_5) : \\
C_{10} & : [P \ v] T \lor [P \ u = p \ u] F \\
Res(C_{10}, C_6) : \\
C_{11} & : [P \ u = p \ u] F \lor [P \ v = p \ v] F \\
\text{UNH}(C_{11}) \text{ with } [p/P] : \\
& \Box
\end{align*}
\]

Alternatively one can directly prove that \(\vdash \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \ldots \alpha \). Again the proof is a little more bulky.

### 8.4 Reasoning about Sets with Leibniz Equality

The case study carried out with LEO on the examples presented in the Mizar article *Boolean Properties of Sets* [TS89] has already been discussed in Section 7.4. This case study makes use of the expressiveness of classical type theory and encodes sets as characteristic functions instead of employing, e.g., Tarski-Grothendieck set theory [Try89].

The particular proofs generated by LEO are in most cases quite short and elegant. Very interesting is, that some proofs are found immediately by definition expansion and clause normalisation.

LEO’s performance together with the detailed proofs is reported at [http://www.ags.uni-sb.de/projects/deduktion/projects/hot/leo/](http://www.ags.uni-sb.de/projects/deduktion/projects/hot/leo/).

Currently the author experiments with the examples of the article Basic Properties of Sets [Byl89].

### 8.5 Positive Extensionality Rules in \(\mathcal{EP}\) and \(\mathcal{ERUE}\)

The examples discussed in this section demonstrate that the positive extensionality rules (or extensionality axioms which we want to avoid) are unavoidable in order to reach Henkin completeness approaches for primitive equality. As discussed in detail in chapter 2.8 none of these examples can be proven in \(\mathcal{EP}\) or \(\mathcal{ERUE}\) without employing the additional extensionality rules.

\[\mathcal{EP}^{\text{para}}\]

\[C_1 : [a = \neg a]^T\]

**Refutation in \(\mathcal{EP}\) and \(\mathcal{ERUE}\)**

\[
\begin{align*}
\text{Eq}^\text{v}(C_1), \text{CNF} : & \quad C_2 : [A]^F \lor [A]^T \\
Fac(C_2), \text{UNIT}, Fac(C_3), \text{UNIT} : & \quad C_4 : [A]^F \\
Res(C_4, C_5), \text{UNIT} : & \quad C_6 : [A]^T
\end{align*}
\]

As rule \(\text{Para}\) is not needed this refutation is also possible in calculus \(\mathcal{ERUE}\).

\[\mathcal{EP}^{\text{para}}\]

\[C_1 : [G \ X \Rightarrow \ldots]^T\]
Refutation in $\mathcal{EP}$ and $\mathcal{ERU}$

$$\begin{align*}
\text{Fun}_c^c(C_1) : & \quad C_2 : [G \times Y \vdash p \ Y]^T \\
\text{Equi}^{(c)}(C_1) : & \quad C_3 : [G \times Y]^F \lor [p \ Y]^T \\
\text{Prim}(C_3), \text{Subst} : & \quad C_5 : [G \times Y]^F \lor [p \ Y]^T \\
\text{Prim}(C_4), \text{Subst} : & \quad C_6 : [G \times Y]^F \lor [p \ Y]^F \\
\text{Fac}(C_3), \text{UNI} : & \quad C_7 : [p \ Y]^T \\
\text{Fac}(C_5), \text{UNI} : & \quad C_8 : [p \ Y]^F \\
\text{Res}(C_7, C_8), \text{UNI} : & \quad C_9 : \Box
\end{align*}$$

As rule $\text{Para}$ is not needed this proof is also possible in calculus $\mathcal{ERU}$.

$\text{E}_{3}^{\text{Para}} \ C_1 : [m = \lambda X_{\alpha} \ (\exists X_{\alpha} \ X \land \neg X)]^T$

Refutation in $\mathcal{EP}$ and $\mathcal{ERU}$

$$\begin{align*}
\text{Fun}_c^c(C_1) : & \quad C_2 : [\text{m} Y_{\alpha} = (\exists X_{\alpha} \ X \land \neg X)]^T \\
\text{Equi}^{(c)}(C_2) : & \quad C_3 : [M Y_{\alpha}^F \lor [\text{m}]^T] \\
\text{Prim}(C_3), \text{Subst} : & \quad C_5 : [H Y_{\alpha}^F \lor [\text{m}]^F] \\
\text{Fac}(C_4), \text{UNI}, \text{Fac}(C_5), \text{UNI} : & \quad C_6 : [\text{m} Y_{\alpha}^F \lor [\text{m}]^F] \\
\text{Res}(C_5, C_6), \text{UNI} : & \quad C_9 : \Box
\end{align*}$$

where $s_{\alpha}$ is a Skolem constant for $X$. As rule $\text{Para}$ is not needed this proof is also possible in calculus $\mathcal{ERU}$.

$\text{E}_{4}^{\text{Para}} \ C_1 : [m = \lambda X_{\alpha} \neg(m \ X)]^T$

Refutation in $\mathcal{EP}$ and $\mathcal{ERU}$

$$\begin{align*}
\text{Fun}_c^c(C_1) : & \quad C_2 : [\text{m} Y = \neg(m Y)]^T \\
\text{Equi}^{(c)}(C_2) : & \quad C_3 : [\text{m} Y]^F \lor [\text{m} Y]^T \\
\text{Fac}(C_3), \text{UNI}, \text{Fac}(C_4), \text{UNI} : & \quad C_5 : [\text{m} Y]^T \\
\text{Res}(C_5, C_6), \text{UNI} : & \quad C_7 : \Box
\end{align*}$$

As rule $\text{Para}$ is not needed this proof is also possible in calculus $\mathcal{ERU}$.

$\text{E}_{5}^{\text{Para}} \ C_1 : [P_{q_r s}^{\to \to \to \to \to \to \to r} \ r]^T \\
C_2 : [q \ X_{\alpha} = \neg(r X)]^T$

where $P_{(s_{1} \to \to \to \to \to \to \to r), X_{i}}$ are free variables and $q_{\alpha} \to \to \to \to \to \to r_{\alpha} \to \to \to \to \to r$ are function constants.

Refutation in $\mathcal{EP}$

$$\begin{align*}
3 \times \text{Fun}_c^c(C_1) : & \quad C_3 : [P q_{Y_{\alpha}^{1}} Y_{\alpha}^{2} Y_{\alpha}^{3} = p_{r} Y_{\alpha}^{1} Y_{\alpha}^{2} Y_{\alpha}^{3}]^T \\
\text{Fun}_c^c(C_2) : & \quad C_4 : [q_{X Z} = \neg(r X Z)]^T \\
\text{Para}(C_4, C_5) : & \quad C_5 : [p_{r} Y_{\alpha}^{1} Y_{\alpha}^{2} Y_{\alpha}^{3}]^F \lor [P q_{r} Y_{\alpha}^{1} Y_{\alpha}^{2} Y_{\alpha}^{3}]^F
\end{align*}$$

By pre-unifying $C_5$ one can compute the following unifier for its unification constraint: $[(\forall X_{\alpha} \to \to \to \to \to r), V_{\alpha}, W_{\alpha}, T_{\alpha}, U \lor W = \neg(r \lor W)]/P, V / Y_{\alpha}^{1}, W / Y_{\alpha}^{2}]$. Thus by eager unification applied to $C_5$ with get:

$$\text{UNI}(C_3), \text{Subst} : C_9 : [r \ X Z = \neg(r X Z)]^F$$

The rest of the refutation is analogous to $\text{E}_{1}^{\text{Para}}$.

Refutation in $\mathcal{ERU}$

$$\begin{align*}
3 \times \text{Fun}_c^c(C_1) : & \quad C_3 : [P q_{Y_{\alpha}^{1}} Y_{\alpha}^{2} Y_{\alpha}^{3} = p_{r} Y_{\alpha}^{1} Y_{\alpha}^{2} Y_{\alpha}^{3}]^T \\
\text{Fun}_c^c(C_2) : & \quad C_4 : [q_{X Z} = \neg(r X Z)]^T \\
\text{Equi}^{(c)}(C_3) : & \quad C_5 : [P q_{Y_{\alpha}^{1}} Y_{\alpha}^{2} Y_{\alpha}^{3}]^F \lor [P q_{r} Y_{\alpha}^{1} Y_{\alpha}^{2} Y_{\alpha}^{3}]^T \\
\text{Fun}_c^c(C_4), \text{CNF} : & \quad C_9 : [q_{X Z}]^F \lor [r X Z]^F
\end{align*}$$

The rest of the refutation is straightforward resolution on $C_5, \ldots, C_8$. 

8.6 Comparing \( \mathcal{EP} \) and \( \mathcal{ERUE} \)

**Properties of Primitive Equality**  The following examples demonstrate that primitive equality denotes an extensional congruence relation in all Henkin models (i.e., the intended equality relation). The single properties to be checked are reflexivity \( (\mathcal{E}^F_1) \), symmetry \( (\mathcal{E}^F_2) \), transitivity \( (\mathcal{E}^F_3) \), congruence \( (\mathcal{E}^F_4) \) and extensionality \( (\mathcal{E}^F_5) \).

\[ \mathcal{E}^F_1 \forall A \alpha A = A. \]
Negation and clause normalisation leads to \( (a) \) a new Skolem constant:
\[ C_1 : [a = a]^F \]

**Refutation in \( \mathcal{EP} \):** immediately by unification.

**Refutation in \( \mathcal{ERUE} \):** immediately by unification.

\[ \mathcal{E}^F_2 \forall A \alpha \bullet (A \Rightarrow (B \Rightarrow A)). \]
Negation and clause normalisation leads to \( (a, b, c, p \alpha \Rightarrow) \) are new Skolem constants:
\[ C_1 : [a \Rightarrow b]^T \quad C_2 : [b \Rightarrow a]^F \]

**Refutation in \( \mathcal{EP} \):**

\[ \text{Lib}(C_2), \text{CNF} \quad C_3 : [p \Rightarrow a]^T \quad C_4 : [p \Rightarrow b]^F \]
\[ \text{Para}(C_1, C_3), \text{Triv} \quad C_5 : [p \Rightarrow b]^T \]
\[ \text{Res}(C_4, C_3), \text{Triv} \quad \square \]

**Refutation in \( \mathcal{ERUE} \):**

\[ \text{Res}(C_1, C_2), \text{Triv} : \square \text{ (possible because of symmetry convention in \( \mathcal{ERUE} \))} \]

\[ \mathcal{E}^F_3 \forall A \alpha \forall B \alpha \forall C \alpha (A \Rightarrow B \land (B \Rightarrow C)) \Rightarrow (A \Rightarrow C). \]
Negation and clause normalisation leads to \( (a, b, c, p \alpha \Rightarrow) \) are new Skolem constants:
\[ C_1 : [a = b]^T \quad C_2 : [b = c]^T \quad C_3 : [a = c]^F \]

**Refutation in \( \mathcal{EP} \):**

\[ \text{Lib}(C_3), \text{CNF} \quad C_4 : [p \Rightarrow a]^T \quad C_5 : [p \Rightarrow c]^F \]
\[ \text{Para}(C_1, C_4), \text{Triv} : C_6 : [p \Rightarrow b]^T \]
\[ \text{Para}(C_2, C_6), \text{Triv} : C_7 : [p \Rightarrow c]^T \]
\[ \text{Res}(C_5, C_7), \text{Triv} : \square \]

**Refutation in \( \mathcal{ERUE} \):**

\[ \text{Res}(c_3, c_2) : C_4 : [(a = c) = (b = c)]^F \]
\[ 2 \times \text{Dec}(c_4), \text{Triv} : C_5 : [a = b]^F \]
\[ \text{Res}(c_1, c_5), \text{UNF} : C_6 : \square \]

\[ \mathcal{E}^F_4 \forall F \alpha \Rightarrow \forall A \alpha \forall B \alpha (A \Rightarrow B) \Rightarrow (F \Rightarrow (F \Rightarrow B)). \]
Negation an clause normalisation leads to \( (a, b, c, d, f \alpha \Rightarrow, p \beta \Rightarrow) \) are new Skolem constants:
\[ C_1 : [a = b]^T \quad C_2 : [d \Rightarrow f]^T \quad C_3 : [f \Rightarrow f]^F \]

**Refutation in \( \mathcal{EP} \):**

\[ \text{Lib}(C_2), \text{CNF} \quad C_3 : [p \Rightarrow(f \Rightarrow a)]^T \quad C_4 : [p \Rightarrow(f \Rightarrow b)]^F \]
\[ \text{Para}(C_1, C_3), \text{Triv} : C_5 : [p \Rightarrow(f \Rightarrow b)]^T \]
\[ \text{Res}(C_3, C_4), \text{Triv} : \square \]

**Refutation in \( \mathcal{ERUE} \):**

\[ \text{Dec}(c_2), \text{Triv} : C_3 : [a = b]^F \]
\[ \text{Res}(c_1, c_3), \text{UNF} : C_4 : \square \]
\( E^\alpha_d \forall F_{\alpha \rightarrow \beta} \forall G_{\alpha \rightarrow \beta} (\forall A_{\alpha} \ (F \ A = \beta \ G \ A)) \Rightarrow (F = G) \)

Negation and clause normalisation leads to \((f_{\alpha \rightarrow \beta}, g_{\alpha \rightarrow \beta}, p_{\beta \rightarrow \alpha}, s_{\beta} \) are new Skolem constants):

\[ C_1 : \lfloor f = g \rfloor^T \quad C_2 : \lfloor f = g \rfloor^F \]

Refutation in \( \mathcal{EP} \):

\[
\begin{align*}
\text{Func}(C_2) : \\
\text{Leib}(C_3), \text{CNF} : \\
\text{Para}(C_1, C_2), \text{UNIT} : \\
\text{Res}(C_6, C_3), \text{UNIT} : \\
\end{align*}
\]

Refutation in \( \mathcal{ERUC} \):

\[
\begin{align*}
\text{Func}(C_2) : \\
\text{Res}(c_1, c_3), \text{UNIT} : \\
\end{align*}
\]

Note that the \( \mathcal{ERUC} \) refutations are in all cases shorter and more elegant.

Especially \( E^\alpha_d \) shows that in extensional higher-order paramodulation it might be useful to allow paramodulation also on unification constraints in order to get shorter and simpler proofs without recursive calls from within the unification process. On the other hand it seems to be quite complicated to guide an approach that simultaneously paramodulates into unification constraints and employs recursive calls to the overall refutation procedure from within unification.

**Primitive Equality and Leibniz Equality** We analyse in calculi \( \mathcal{EP} \) and \( \mathcal{ERUC} \) whether Leibniz equality and primitive equality denote the same relation.

\( E^\alpha_1 \) We prove that \( =^\alpha \rightarrow =^\alpha \rightarrow =^\alpha \), i.e., that \( (\lambda X_{\alpha} \lambda Y_{\alpha}, \forall P_{\alpha \rightarrow \alpha} \ P \ X \Rightarrow P \ Y) =^\alpha \rightarrow =^\alpha \rightarrow =^\alpha \)

Refutation in \( \mathcal{EP} \) (\( \alpha_1, \beta_0 \) and \( p_{\alpha \rightarrow \alpha}, q_{\alpha \rightarrow \alpha}, r_{\alpha \rightarrow \alpha} \) are Skolem constants):

\[
\begin{align*}
\text{CNF}(E^\alpha_1) : \\
2 \times \text{Func}(C_1) : \\
\text{Equiv}(C_2), \text{CNF} : \\
\text{Res}(C_5, C_3), \text{Triv} : \\
2 \times \text{Leib}(C_6) : \\
\text{CNF}(C_7) : \\
\text{Prim}(C_8) : \\
\text{Subst}(C_{13}) : \\
\text{Fac}(C_{13}) : \\
2 \times \text{Dec}(C_{14}), \text{Triv} : \\
2 \times \text{FlexRigid}(C_{15}) : \\
2 \times \text{Subst}(C_{16}) : \\
3 \times \text{Triv}(C_{17}) : \\
4 \times \text{Para}(C_8, C_9, C_{10}) : \\
\end{align*}
\]

Straightforward Resolution on \( C_{19}, C_{20}, C_{21}, C_{21} : \)

Refutation in \( \mathcal{ERUC} \)
We replay the derivation above and instead of the paramodulation steps between \( \mathcal{C}_{18} \) and \( \mathcal{C}_8, \mathcal{C}_9, \mathcal{C}_{10} \) we resolve between \( \mathcal{C}_{18} \) and the unification constraint \( \mathcal{C}_6 \).

We prove that \( \models \alpha \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha \), i.e., that \( \forall Q(\alpha \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha) \) (\( Q(\lambda X, \lambda Y, \forall P \rightarrow \alpha, \alpha X \Rightarrow P Y) \Rightarrow (Q = \alpha) \))

Refutation in \( \mathcal{EP} \) \( (Q(\alpha \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha) \) is a Skolem constant):

\[
\begin{align*}
\text{CNF}(\mathcal{E}^\alpha_2) : \quad & \mathcal{C}_1 : [q (\lambda X, \lambda Y, \forall P \rightarrow \alpha, \alpha X \Rightarrow P Y)]^T \\
& \mathcal{C}_2 : [q = \alpha]^F \\
\text{Res}(\mathcal{C}_1, \mathcal{C}_2) : \quad & \mathcal{C}_3 : [(q (\lambda X, \lambda Y, \forall P \rightarrow \alpha, \alpha X \Rightarrow P Y)) \Rightarrow (q = \alpha)]^F \\
\text{Dec}(\mathcal{C}_3, \text{Triv}) : \quad & \mathcal{C}_4 : [(\lambda X, \lambda Y, \forall P \rightarrow \alpha, \alpha X \Rightarrow P Y) \Rightarrow (\lambda X, \lambda Y, \forall P \rightarrow \alpha, \alpha X \Rightarrow P Y)]^F
\end{align*}
\]

\( \mathcal{C}_4 \) is identical to \( \mathcal{C}_1 \) in the refutation for \( \mathcal{E}^\alpha_1 \) above.

Refutation in \( \mathcal{ERIE} \)

We employ the previous initial derivation and then an refutation that is analogous to the \( \mathcal{ERIE} \)-refutation for \( \mathcal{E}^\alpha_1 \).

Reasoning about Sets with Primitive Equality

\( \mathcal{E}^\alpha_{1^{st}} \) Let the set of odd numerals be defined as the set of non-even numerals. Then the power set of the set of odd numerals greater than 100 is equal to the power set of the set of even numerals greater that 100:

\[
\{X \mid \text{odd } X \land \text{num } X\} = \{X \mid \neg \text{ev } X \land \text{num } X\} \Rightarrow \\
\phi \{X \mid \text{odd } X \land \text{num } X \land X > 100\} = \phi \{X \mid \neg \text{ev } X \land \text{num } X \land X > 100\}
\]

where the power set is defined by \( \phi := \lambda M_{\alpha \rightarrow \alpha} \lambda N_{\alpha \rightarrow \alpha} \forall X_{\alpha} \cdot N X \Rightarrow M X \cdot \). Negation and clause normalisation leads to:

\[
\begin{align*}
\mathcal{C}_1 : [(\lambda X, \alpha \land \text{num } X) = (\lambda X, \neg \text{ev } X \land \text{num } X)]^T \\
\mathcal{C}_2 : [(\lambda X, \alpha \land \text{num } X) \land X > 100] = (\lambda X, \neg \text{ev } X \land \text{num } X \land X > 100)]^F
\end{align*}
\]

The probably simplest refutation in calculus \( \mathcal{EP} \) is:

\[
\begin{align*}
\text{Func}^\beta(\mathcal{C}_1) : \quad & \mathcal{C}_3 : [(\neg \text{ev } Y \land \text{num } Y) = (\neg \text{ev } Y \land \text{num } Y)]^F \\
\text{Func}(\mathcal{C}_2) : \quad & \mathcal{C}_4 : [\forall X_n n X \Rightarrow ((\neg \text{ev } X \land \text{num } X) \land X > 100)]^F \\
\text{Equiv}(\mathcal{C}_4) : \quad & \mathcal{C}_5 : [\forall X_n n X \Rightarrow ((\neg \text{ev } X \land \text{num } X) \land X > 100)]^T \\
& \mathcal{C}_6 : [\forall X_n n X \Rightarrow ((\neg \text{ev } X \land \text{num } X) \land X > 100)]^F
\end{align*}
\]

The following \( \mathcal{ERIE} \)-refutation has a more goal directed character and is at least not more
complicated than the above paramodulation proof:

\[
\begin{align*}
\text{Func}(\mathcal{C}_1) & : & \mathcal{C}_3 : [ (odd \ Y \land \neg num \ Y) = (\neg ev \ Y \land \neg num \ Y) ]^T \\
\text{Func}(\mathcal{C}_2) & : & \mathcal{C}_4 : [ (\forall X, n \ X \Rightarrow (odd \ X \land num \ X) \land X > 100) \\
& & = (\forall X, n \ X \Rightarrow ((\neg ev \ X \land num \ X) \land X > 100)) ]^P \\
\text{Dec}(\mathcal{C}_4), \text{Triv} & : & \mathcal{C}_5 : [ (\lambda X, n \ X \Rightarrow (odd \ X \land num \ X) \land X > 100) \\
& & = (AX, n \ X \Rightarrow ((\neg ev \ X \land num \ X) \land X > 100)) ]^P \\
\text{Func}(\mathcal{C}_5) & : & \mathcal{C}_6 : [ (n \ s \Rightarrow (odd \ s \land num \ s) \land s > 100) ]^P \\
& & = (n \ s \Rightarrow ((\neg ev \ s \land num \ s) \land s > 100)) ]^P \\
2 \times \text{Dec}(\mathcal{C}_5), 4 \times \text{Triv} & : & \mathcal{C}_7 : [ (odd \ s \land num \ s) = (\neg ev \ s \land num \ s) ]^P \\
\text{Res}(\mathcal{C}_5, \mathcal{C}_3), \text{UNL} & : & \\
\end{align*}
\]

If we slightly modify, i.e., complicate, the example by switching the first two conjuncts in the definition of odd numerals, such that \( \mathcal{C}_1 : [ \lambda X, (num \ X \land odd \ X) = \lambda X, (\neg ev \ X \land num \ X) ]^T \) and \( \mathcal{C}_3 : [ (odd \ X \land num \ X) = (\neg ev \ X \land num \ X) ]^T \), both refutations obviously need to employ additional recursive calls from within unification in order to show that \( (\neg ev \ X \land odd \ X) = (odd \ X \land num \ X) ]^P \) is a solvable extensional unification problem. More precisely in an analogous paramodulation refutation to above additional recursive calls are necessary to justify the paramodulation steps leading to \( \mathcal{C}_8 \) and \( \mathcal{C}_{10} \) (note that \text{Triv} is not applicable in both cases to justify the resolution step immediately). These clauses would look like

\[
\begin{align*}
\mathcal{C}_8 & : [ (\forall X, n \ X \Rightarrow (\neg ev \ X \land num \ X) \land X > 100) ]^T \\
& \lor [ (num \ X \land odd \ X) = (odd \ X \land num \ X) ]^P \\
\end{align*}
\]

and

\[
\begin{align*}
\mathcal{C}_{10} & : [ (\forall X, n \ X \Rightarrow (\neg ev \ X \land num \ X) \land X > 100) ]^T \\
& \lor [ (num \ X \land odd \ X) = (odd \ X \land num \ X) ]^P \\
\end{align*}
\]

In both cases the unification constraints can be eliminated only by recursive calls to overall proof procedure.

A RUE-resolution refutation analogous to the above one would result in \( \text{Res}(\mathcal{C}_5, \mathcal{C}_3) : \mathcal{C}_7 : [ (odd \ s \land num \ s) = (\neg ev \ s \land num \ s) = (num \ s \land odd \ s) = (\neg ev \ s \land num \ s) ]^P \) such that decomposition and subsequent elimination of trivial pairs leads to \( \mathcal{C}_8 : [ (num \ X \land odd \ X) = (odd \ X \land num \ X) ]^P \). Then an recursive call to the overall proof procedure leads to the refutation.

This modified example demonstrates that calculating \( \mathcal{E} \) in some cases unavoidably needs to perform recursive calls to the overall refutation process in order to justify single paramodulation steps. These side computations obviously follow the difference reduction idea. Thus, in practice the paramodulation approach requires both: appropriate heuristics in order to guide a proof along the paramodulation, i.e., term rewriting idea, and appropriate heuristics that guide the side computations based on the difference reduction idea.

The following example demonstrates that in some situations the term rewriting idea even seems to be completely inappropriate and useless. Therefore we further modify and complicate our example above. The idea is to restructure the single conjuncts in the assertion, such that rewriting with the assumption equation at the right subterm of the assertion becomes impossible. This final modification additionally illustrates that an efficient paramodulation approach is probably not easy to mechanise in practice, completely independent from the question if suitable reduction orderings are available or not.

\( \text{Ex}^{\text{def}} \) We complicate problem \( \text{Ex}^{\text{def}} \) by switching the conjuncts in the assertion:

\[
\begin{align*}
\{ X \} & odd \ X \land num \ X = \{ X \} \neg ev \ X \land num \ X \Rightarrow \\
\wp \{ X \} & (odd \ X \land X > 100 \land num \ X) = \wp \{ X \} (\neg ev \ X \land X > 100) \land num \ X \\
\end{align*}
\]

Negation and clause normalisation leads to:

\[
\begin{align*}
\mathcal{C}_1 & : [ \lambda X, (odd \ X \land num \ X) = \lambda X, (\neg ev \ X \land num \ X) ]^T \\
\mathcal{C}_2 & : [ (\lambda N, \forall X, N \ X \Rightarrow ((odd \ X \land X > 100) \land num \ X)) \\
& = (\lambda N, \forall X, N \ X \Rightarrow ((\neg ev \ X \land X > 100) \land num \ X)]^P \\
\end{align*}
\]
8.6. COMPARING EP AND EReUE

We first try to proceed analogous to example \( E_{1}^{ext} \):

\[
\text{Func}(C_1) : \quad C_3 : [(\text{odd } X \land \text{num } X) = (\neg ev \ X \land \text{num } X)]^T
\]

\[
\text{Func}(C_2) : \quad C_4 : [(\forall X. n \ X \Rightarrow (\text{odd } s \land s > 100) \land \text{num } s)] =
\]

\[
(\forall X. n \ X \Rightarrow ((\neg ev \ s \land s > 100) \land \text{num } s) ]^P
\]

Following the paramodulation based refutation for \( E_{1}^{ext} \) we perform a recursive call to the refutation process with rule \textit{Equiv} and then try to rewrite the resulting clauses in an appropriate way:

\[
\text{Equiv}(C_4) : \quad C_5 : [(\forall X. n \ X \Rightarrow ((\text{odd } X \land X > 100) \land \text{num } X))]^T \lor
\]

\[
[\forall X. n \ X \Rightarrow (((\neg ev \ X \land X > 100) \land \text{num } X))]^P
\]

\[
C_6 : [(\forall X. n \ X \Rightarrow ((\text{odd } X \land X > 100) \land \text{num } X)) ]^P \lor
\]

\[
[\forall X. n \ X \Rightarrow (((\neg ev \ X \land X > 100) \land \text{num } X))]^P
\]

Surely, there are many different rewrite steps possible with \( C_3 \) on these clauses, but as the reader may convince himself, none of these rewrite steps lead to unification constraints that are solvable (even recursive calls to the overall proof procedure cannot help and unfortunately only overwhelm the search space with useless clauses). The problem is that the right paramodulation step is simple not possible because of the improperly modified term-structure. The only possible way to proceed that the author explored is to employ a difference reducing refutation as motivated by the respective \textit{EReUE} refutation in example \( E_{1}^{ext} \) above:

\[
\text{Dec}(C_6), \text{Triv} : \quad C_5 : [(\forall X. n \ X \Rightarrow ((\text{odd } X \land X > 100) \land \text{num } X))]
\]

\[
= (\forall X. n \ X \Rightarrow (((\neg ev \ X \land X > 100) \land \text{num } X))]^P
\]

\[
\text{Func}(C_5) : \quad C_6 : [(n \Rightarrow (\neg ev \ X \land X > 100) \land \text{num } X))]^P
\]

\[
\text{Dec}(C_6), \text{Triv} : \quad C_7 : [(\forall s \Rightarrow (\text{odd } s \land s > 100) \land \text{num } s)]^P
\]

Instead of immediately resolving between \( C_7 \) and \( C_3 \) as in the corresponding derivation for \( E_{1}^{ext} \) we employ the extensionality rules to both clauses and proceed with a straightforward resolution proof.

\[
\text{Equiv}(C_7) \land \text{CNF}, \text{Fac, UNI} : \quad C_8 : [\text{odd } s]^T \lor [ev \ s]^P
\]

\[
C_9 : [\text{odd } s]^T \lor [s > 100]^T
\]

\[
C_{10} : [\text{odd } s]^T \lor [\text{num } s]^T
\]

\[
C_{11} : [s > 100]^T \lor [ev \ s]^P
\]

\[
C_{12} : [s > 100]^T
\]

\[
C_{13} : [s > 100]^T \lor [\text{num } s]^T
\]

\[
C_{14} : [\text{num } s]^T \lor [ev \ s]^P
\]

\[
C_{15} : [\text{num } s]^T \lor [s > 100]^T
\]

\[
C_{16} : [\text{num } s]^T
\]

\[
C_{17} : [\text{odd } s]^P \lor [s > 100]^P \lor [\text{num } s]^P \lor [ev \ s]^T
\]

Straightforward resolution refutation with \( C_8, C_{21}, C_{22}, C_{23} \)

An alternative Refutation in \textit{RUE-resolution} approach that illustrates the difference reduction idea even better is the following:

\[
\text{Res}(C_3, C_7) : \quad C_5 : [(\text{odd } X \land \text{num } X) = (\neg ev \ X \land \text{num } X)] =
\]

\[
[((\text{odd } s \land s > 100) \land \text{num } s)) = ((\neg ev \ s \land s > 100) \land \text{num } s)]^P
\]

\[
\text{Equiv}(C_5), \text{CNF} : \quad C_7 : [(\text{odd } X \land \text{num } X) = (\neg ev \ X \land \text{num } X)]^T \lor
\]

\[
[((\text{odd } s \land s > 100) \land \text{num } s)) = ((\neg ev \ s \land s > 100) \land \text{num } s)]^P
\]

\[
C_8 : [(\text{odd } X \land \text{num } X) = (\neg ev \ X \land \text{num } X)]^P \lor
\]

\[
[((\text{odd } s \land s > 100) \land \text{num } s)) = ((\neg ev \ s \land s > 100) \land \text{num } s)]^P
\]
\(2 \times \text{Equiv}'(\mathcal{C}_7), \text{Cnf, Fac, Triv}:
\)
\(\mathcal{C}_9: [\text{odd } s]^T \lor [\text{even } s]^T \lor [\text{odd } X]^T \lor [\text{even } X]^T \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{num } X]^F\)
\(\mathcal{C}_{10}: [\text{odd } X]^T \lor [\text{even } X]^T \lor [\text{odd } s]^F \lor [s > 100]^F \lor [\text{odd } X]^F \lor [\text{num } X]^F\)
\(\mathcal{C}_{11}: [\text{odd } s]^T \lor [\text{even } s]^T \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{even } X]^F \lor [\text{num } X]^F \lor [\text{odd } X]^F\)
\(\mathcal{C}_{12}: [\text{even } s]^T \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{odd } s]^F \lor [\text{even } X]^F \lor [\text{num } X]^F \lor [\text{odd } X]^F\)

\(2 \times \text{Equiv}'(\mathcal{C}_7), \text{Cnf, Fac, Triv}:
\)
\(\mathcal{C}_{13}: [\text{num } s]^T \lor [\text{num } X]^T\)
\(\mathcal{C}_{14}: [s > 100]^T \lor [\text{num } X]^T\)
\(\mathcal{C}_{15}: [\text{odd } s]^T \lor [\text{num } X]^T \lor [\text{even } s]^F\)
\(\mathcal{C}_{16}: [s > 100]^T \lor [\text{odd } X]^T \lor [\text{even } X]^F\)
\(\mathcal{C}_{17}: [\text{odd } s]^T \lor [\text{odd } X]^F \lor [\text{even } s]^F \lor [\text{even } X]^F\)
\(\mathcal{C}_{18}: [s > 100]^T \lor [\text{even } X]^T \lor [\text{odd } X]^F \lor [\text{num } X]^F\)
\(\mathcal{C}_{19}: [\text{even } s]^T \lor [\text{num } X]^F \lor [\text{odd } s]^F \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{even } X]^F\)
\(\mathcal{C}_{20}: [\text{even } s]^T \lor [\text{odd } X]^T \lor [\text{odd } s]^F \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{even } X]^F\)
\(\mathcal{C}_{21}: [\text{even } s]^T \lor [\text{even } X]^T \lor [\text{odd } s]^F \lor [\text{num } s]^F \lor [s > 100]^F \lor [\text{odd } X]^F \lor [\text{num } X]^F\)

Straightforward resolution proof on \(\mathcal{C}_9, \ldots, \mathcal{C}_{21} \square\)

We could even resolve immediately between \(\mathcal{C}_3\) and \(\mathcal{C}_4\) and proceed with a recursive call to the overall proof procedure as illustrated above. The resulting set of clauses then again increases a bit but a straightforward resolution proof is still easy to find.

With this example we demonstrated that a pure term rewriting is most likely impossible in extensional higher-order resolution as a combined extensionality treatment that takes functional and Boolean extensionality into account seems to require the application of difference reduction techniques.

The following example illustrates that neither rule \emph{Para} nor rule \emph{Para'} capture full functional extensionality.\(^1\)

\(\text{E}_1^{\text{unc}} (\forall X. f_{\rightarrow_i} X = g_{\rightarrow_i} X) \Rightarrow (p_{(i \rightarrow_i) \rightarrow} f \Rightarrow p_{(i \rightarrow_i) \rightarrow} g)\)

This problem normalizes to:

\(\mathcal{C}_1: [f \rightarrow_i X = g \rightarrow_i X]^T \quad \mathcal{C}_2: [p \rightarrow f]^T \quad \mathcal{C}_3: [p \rightarrow g]^T\)

Note that neither with rule \emph{Para} nor with rule \emph{Para'} a fruitful rewriting step is possible (when applying rule \emph{Para'} in order to rewrite \(f\) in \(\mathcal{C}_2\) we obtain the unification constraint \([p \rightarrow f]^T = [p \rightarrow g]^T\), which is not solvable with a projection binding). Instead we have to employ difference reducing derivations and again the \text{ERLE} derivation is more elegant.

Refutation in \text{EP}

\(\text{Res}(\mathcal{C}_3, \mathcal{C}_3), \text{Dec}:
\)
\(\mathcal{C}_4: [f = g]^T\)
\(\text{Func}(\mathcal{C}_3):
\)
\(\mathcal{C}_5: [f \rightarrow_i s_i = g \rightarrow_i s_i]^T\)
\(\text{Let}(\mathcal{C}_3):
\)
\(\mathcal{C}_6: [p \rightarrow f s_i]^T \quad \mathcal{C}_7: [p \rightarrow g s_i]^T\)
\(\text{Para}(\mathcal{C}_1, \mathcal{C}_6), \text{UNIT}:
\)
\(\mathcal{C}_8: [p \rightarrow g s_i]^T\)
\(\text{Res}(\mathcal{C}_8, \mathcal{C}_7), \text{Triv}:
\)

Refutation in \text{ERLE}

\(\text{Res}(\mathcal{C}_2, \mathcal{C}_3), \text{Dec}:
\)
\(\mathcal{C}_4: [f = g]^T\)
\(\text{Func}(\mathcal{C}_4):
\)
\(\mathcal{C}_5: [f \rightarrow_i s_i = g \rightarrow_i s_i]^T\)
\(\text{Res}(\mathcal{C}_5, \mathcal{C}_1), \text{UNIT}:
\)

\(^1\)It has been claimed by an unknown referee of [Ben98], that rule \emph{Para'} in contrast to rule \emph{Para} captures full functional extensionality, which is not the case as our examples illustrate.
8.7 Raised Questions

The examples examined so far (an excerpt has been presented in this Chapter) raise various questions concerning the calculi $\mathcal{ER}$, $\mathcal{EP}$, and $\mathcal{ERUE}$:

1. Is rule $\text{FlexFlex}$ indeed admissible in all three approaches, i.e., can this rule be avoided (as already realised in LEO)? As to the best of my knowledge there is no counterexample to this conjecture.

2. Can rule $\text{Leib}$ be restricted to base types as conjectured in Remark 4.18 and as already employed in LEO’s extensionality treatment (cf. step 7 in LEO’s main loop as presented in Subsection 7.2.3)? For the examples examined so far, this restriction is indeed possible without affecting refutability.

3. Is decomposition rule $\text{Dec}$, as used in this thesis, or rule $\text{Dec}'$, as used in [BK98a], better suited for applications in practice? The difference between both rules has been illustrated in Subsection 8.2.

4. Is the traditional paramodulation rule $\text{Pam}$ (cf. Figure 5.1) or its higher-order counterpart $\text{Pam}'$ (cf. Figure 5.1) better suited in applications of calculus $\mathcal{EP}$ in practice. The connection between both rules has been illustrated in Section 5.1.

5. In 5.2 it has been remarked that paramodulation into unification constraints may shorten some proofs, but it might be quite complicated to guide this in practice. This question needs further investigation.

6. The following example demonstrates that in calculus $\mathcal{ERUE}$ various applications of rule $\text{Leib}$ can be substituted by alternative derivations applying resolution on unification constraints instead.

Remark 8.1. In $\mathcal{ERUE}$ one can prove example $\mathcal{E}_{5}^{\text{ext}}$ (cf. Section 8.1), which formulates an instance of the functional extensionality property for Leibniz equality, as follows (the idea of this refutation is due to Frank Plennig):

\[
\begin{align*}
\text{Cnf}(-E_{5}^{\text{ext}}) : \\
C_{1} & : [P (m \, X)]^{F} \lor [P (n \, X)]^{T} \\
C_{2} & : [q \, m]^{F} \\
C_{3} & : [q \, n]^{F} \\
\text{Res}(C_{2}, C_{3}), \text{Dec}, \text{Triv} : \\
C_{4} & : [m = n]^{F} \\
\text{Funct}(C_{4}) : \\
C_{5} & : [m \, s = n \, s]^{F} \\
\text{Res}(C_{1}, C_{5}) : \\
C_{6} & : [P (m \, X)]^{F} \lor [P (n \, X) = ((m \, s = (n \, s))]^{F} \\
\text{UNIV}(C_{6}, [\lambda X_{\bullet} (m \, s) = X/P]) : 
\end{align*}
\]

Hence the question arises if rule $\text{Leib}$ can be avoided completely in $\mathcal{ERUE}$. As rule $\text{Leib}$ is obviously not avoidable in calculus $\mathcal{EP}$ (there is no proof in $\mathcal{EP}$ for the above problem that avoids the application of rule $\text{Leib}$) this result would be another strong argument in favour of calculus $\mathcal{ERUE}$ and against calculus $\mathcal{EP}$.

7. The calculi do not provide sufficient restrictions of the extensionality and equality rules that allow for a fruitful guidance of the proof search in difficult examples. Thus, further investigations in order to restrict and specialise the developed approaches is important and can probably lead to much more powerful implementations as the LEO-system.

8. It should be examined whether $\mathcal{EP}$ and $\mathcal{ERUE}$ bring essential advantages at all over a pure defined equality treatment as employed in $\mathcal{ER}$. Clearly, one advantage of $\mathcal{EP}$ and $\mathcal{ERUE}$ is that the primitive substitution rule in these approaches allows to immediately instantiate a flexible literal by a primitive equation, whereas in calculus $\mathcal{ER}$ the instantiation of Leibniz equality requires two subsequent primitive substitution steps (which can be improved by extending the primitive substitution rule in $\mathcal{ER}$, such that it also instantiates the free literal head by
most general Leibniz equations, i.e., $\lambda X \forall \alpha \exists \alpha \neg (P (H_1 X)) \lor (P (H_2 X))$, where $H_1$ and $H_2$ are new free variables of appropriate type). The importance of primitive substitutions of equality is demonstrated by the Tps-example THM15b discussed in [ABI+96].
Chapter 9

Applications and Related Work

In this chapter we discuss related work and sketch some applications. The discussion is structured with respect to the different kinds of contributions.

9.1 Cooperation and Joint Work

The contributions of the Chapters 2.1 and 3 have been developed in cooperation with Michael Kohlhase. Some main aspects of this chapter were also positively influenced by discussions with Frank Pfening and Peter Andrews during a research stay at Carnegie Mellon University, Pittsburgh, USA.

As already mentioned, the resolution calculus $\mathcal{ER}$ in Chapter 4 is based on, extends, and corrects Michael Kohlhase’s approaches presented in [Koh94b] and [Koh95]. And the first completeness proof for $\mathcal{ER}$, which is presented in [BK98a] and which differs from the one presented in this thesis, has been developed in cooperation with Michael Kohlhase. The author gained important insights about $\mathcal{ER}$, the completeness proof for $\mathcal{ER}$ and the ideas for $\mathcal{ER}$ in discussions with Frank Pfening.

9.2 Abstract Consistency and Model Existence

The abstract consistency method, which has been extended in this thesis, was first developed by Hintikka and Smullyan [Hint55, Smu63, Smu68] for first-order logic and extended to higher-order logic by Andrews [And71]. The new results can clarify the relationship between syntax and semantics for a variety of higher-order deduction calculi. Up to now calculus development in higher-order logic has been guided by Andrew’s Unifying Principle for Type Theory [And71]. This model existence theorem has set the completeness standard for higher-order calculi such as [And71, Hue72, Hue73a, ALCMP84], even though it is weaker (i.e., with respect to this principle fewer formulæ are valid) than the intuitive one given by Henkin Models. The new model existence theorems, e.g., allow to analyse completeness of the traditional higher-order resolution calculi [And71, JP72, Hue72, Hue73a] or the more recent ones [Wol93, Koh94b, Koh95] with respect to Henkin semantics or, if the extensionality axioms in these approaches were avoided, with respect to the $\Sigma$-Models. Both was hardly possible before, i.e., only when using direct proofs and without the help of a proof tool.

The semantical notions in Chapter 2.1 stem from earlier attempts of Kohlhase to achieve completeness with respect to Henkin models for higher-order tableaux [Koh95, Koh98] and especially from the attempts of the author in cooperation with Kohlhase to achieve completeness for higher-order resolution [Koh94a, Ben97, BK97b, BK98a].

A model existence theorem for a logical system $\mathcal{L}$ is a theorem of the form: If a set of sentences $\Phi$ in $\mathcal{L}$ is a member of an abstract consistency class $\Gamma$, then there exists an $\mathcal{L}$-model for $\Phi$. Thus if we want to show the completeness of a particular calculus $\mathcal{L}$, we first prove that the class $\Gamma$
of sets of sentences $\Phi$ that are $\mathcal{L}$-consistent (cannot be refuted in $\mathcal{L}$) is an abstract consistency class, then the model existence theorem tells us that $\mathcal{L}$-consistent sets of sentences are satisfiable in $\mathcal{L}$. Now we assume that a sentence $\mathbf{A}$ is valid in $\mathcal{L}$, so $\neg \mathbf{A}$ does not have a $\mathcal{L}$-model and is therefore $\mathcal{L}$-inconsistent. From this it is easy to verify that $\mathbf{A}$ is a theorem of $\mathcal{L}$. Note that with this argumentation the completeness proof for $\mathcal{L}$ condenses to verifying that $\Gamma$ is an abstract consistency class, a task that does not refer to $\mathcal{L}$-models. Thus the usefulness of model existence theorems derives from the fact that it replaces the model-theoretic analysis in completeness proofs with the verification of some proof-theoretic conditions (membership in $\Gamma$). In this respect a model existence theorem is similar to a Herbrand Theorem, but it is easier to generalise to other logic systems like higher-order logic.

Another application of model existence theorems is that they allow for very simple (but non-constructive) proofs of cut-elimination theorems. In [And71] Andrews applies his Unifying Principle to cut-elimination in a non-extensional sequent calculus by proving the calculus complete (relative to $\exists$) both with and without the cut rule and concludes that cut-elimination is valid for this calculus. In the extensional case, where a cut-elimination theorem can be found in [Tak68, Tak87], we can directly model a cut-elimination proof following Andrews’ approach, using the model existence theorem for Henkin models.

Takahashi discusses in [Tak68] a non-constructive cut-elimination theorem for simple type theory with extensionality. This work is based on [Tak67], where a respective result has been shown for simple type theory without extensionality. Cut-elimination addresses the question: "Given a deductive system, a proof for $\mathbf{A} \Rightarrow \mathbf{B}$ and one for $\mathbf{B} \Rightarrow \mathbf{C}$, how can we construct a proof for $\mathbf{A} \Rightarrow \mathbf{C}$?" Takahashi analyses this problem for a Gentzen style sequent calculus [Gen35]. His system is also used by Takeuti in [Tak87]. A constructive proof of cut-elimination for intuitionistic type theory is presented by a student of Gandy in [Unk].

With respect to the connection of the primitive substitution principle and higher-order model existence the author wants to point to the theorem 2 in [BK98a]. A consequence of this restricted model existence theorem would be that depending on the maximal order $\tau$ of an input problem the primitive substitution of universal quantifiers at head positions could be restricted to those types with an order $\leq \tau$. Unfortunately at CADE-15 Peter Andrews presented a counterexample to the argumentation in the proof of theorem 2 given in [BK98a] and [BK97a] so that we had to withdraw the theorem. The author is grateful to Peter Andrews for his advice (and not only for this particular one) but also wants to point out that the lack of theorem 2 does not affect the other results presented in [BK98a] as they do not depend on theorem 2. The overall role of the primitive substitution principle in higher-order theorem proving and the question whether there are ways to restrict this principle is, at least for the author, still not obvious.

In all these applications, the leverage added by the work presented in Chapters 2, 1 and 3 is that we can now extend non-extensional results to extensional cases. However, the generalised model classes have a merit of their own, for instance in higher-order logic programming [NM94], where the denotational semantics of programs can induce non-standard meanings for the classical connectives. For instance, given a SLD-like search strategy as in $\lambda$-PROLOG [Mit93], conjunction is not commutative any more. Therefore, various authors have proposed model-theoretic semantics, where property $b$ (which expresses that the domain $D_\tau$ contains exactly $T$ and $F$) fails. For instance Wolfram uses Andrews $\nu$-complexes [Wol94] as a semantics for $\lambda$-PROLOG and Nadathur uses “labeled structures” for the same purpose in [NM94]. It is plausible to assume that the results of this thesis will be useful for further developments in this direction as well.

### 9.3 Extensional Higher-Order Resolution

Higher-order resolution has been first discussed in [And71]. Andrews calculus still avoids higher-order unification and instead provides a substitution rule, i.e., a instantiation rule for free variables, that basically allows to enumerate the Herbrand universe. Most important about [And71] is, that it adapts Smullyan’s unifying principle to higher-order logic in order to prove completeness of the developed resolution approach with respect to the semantical notion of $\nu$-complexes; see also the
9.3. EXTENSIONAL HIGHER-ORDER RESOLUTION

discussion in Section 9.2 above.

The first mechanisable higher-order resolution calculi were presented by Jensen/Pietrowiski in [JP72] and Huet in [Hue72, Hue73a]. Especially Huet's Constrained Resolution Approach is closely related to the work presented here, as the non-extensional fragment of calculus $\mathcal{ER}$ is essentially a variant of Huet's Constrained Resolution Approach. The differences of extensional higher-order resolution to Huet's constrained resolution approach have been illustrated in this thesis; especially remark 4.17 pointed to the fact, that in contrast to Huet's approach eager unification becomes essential in $\mathcal{ER}$, such that unification can not be delayed until the end of the refutation process.

The resolution calculus $\mathcal{HORES}$ in [Koh94b] influenced the development of calculus $\mathcal{ER}$ to a large extent. The particular connections between $\mathcal{ER}$ and $\mathcal{HORES}$ have been discussed in detail in Section 4.1. Furthermore, extensionality rule $\text{Equ}_K$ is motivated by an analogous rule already given in the higher-order tableau calculus [Koh95]. Whereas this calculus claims to be Henkin complete it can easily be shown by a counterexample, for instance by examples $E_1^{\tau\tau}$ and $E_2^{\tau\tau}$, that it is not. The problem is that [Koh95] does not use a sufficient interplay between the functional and the Boolean extensionality principles. Such an interplay is realised in $\mathcal{ER}$ by the modified rule $\text{Func}$ (which extends the $\alpha$- and $\eta$-unification rules employed in [Koh95]) and the rule $\text{Leib}$ which connects unification constraints with Leibniz equations in connection with rule $\text{Equ}_K$. The extensional tableau calculus $\mathcal{HTE}$ [Koh98] which extends [Koh95] translates the rules of calculus $\mathcal{ER}$ presented in this thesis in a tableau setting. In difference to [Koh95] the calculus in [Koh98] provides an extended unification rule $\text{Func}$ as discussed in this thesis (the extended unification rule $\text{Func}$ applies the functional extensionality principle also to non-$\lambda$-abstractions within higher-order unification).

Wolfram suggests in his resolution approach [Wol93] to employ general higher-order $E$-unification, but does not present a concrete higher-order $E$-unification algorithm for theories $E$ that include full extensionality principles. And it has been illustrated in this thesis, that general calls to a Henkin complete theorem prover are needed to realise extensional higher-order $E$-unification. In other words, extensional higher-order unification unavoidably needs to employ higher-order theorem proving for side computations, such that the difference between extensional higher-order unification and Henkin complete higher-order theorem proving disappears. Instead, both algorithms have to be closely integrated. In order to reach Henkin completeness the resolution approach in [Wol93] therefore either needs to add the extensionality axioms or has to address the problem for mutual recursive calls from resolution to $E$-unification and vice versa.

The difference of higher-order unification as employed in $\mathcal{ER}$ with respect to the higher-order $E$-unification approach discussed in [Smy90] is, that the latter is restricted to first-order theories only and does not take the Boolean extensionality property into account. Hence, it does not allow to unify terms like $\lambda x. \red X \land \circle X$ and $\lambda x. \circle X \land \red X$, whereas $\mathcal{ER}$ allows to tests for the unifiability of two terms with respect to an arbitrary higher-order theory including both extensionality principles. Restricted transformation based approaches to higher-order $E$-unification, where as much computation as possible is pushed to a first-order $E$-unification procedure, are discussed in [QW96, NQ91], and a restricted combinatory logic approach is presented in [DJ92]. Like [Smy90], these approaches do not take the Boolean extensionality into account and do not allow to consider arbitrary higher-order theories. The study of higher-order $E$-unification was first suggested by Siekmann in [Sic84] and a complete overview on first-order $E$-unification is provided by [BS94].

The "theorem proving modulo" approach described in [DKH98] is a way to remove computational arguments from proofs by reasoning modulo a congruence on propositions that is handled by via rewrite rules and equations. In their paper the authors present a higher-order logic as a theory modulo.

Pfenning addresses in [Pfe87] the problem of translating machine found proofs in higher-order logic into natural deduction [Pra65] or sequent style calculi [Gen55]. The proposed solution, which

\footnote{The author does not fully see yet to what extent the "theorem proving modulo" approach can handle full extensionality in the sense of this thesis.}
is based on previous works by Andrews [And80] and Miller [Mil83], first translates the machine
found proof into a so-called expansion tree proof, where it can be cleaned up and brought into a
certain standard form, before it is translated into a natural deduction or sequent calculus proof.
Pfenning's work especially focuses on the problem of extensionality and primitive equality and it
extends Miller's work [Mil83] by introducing the notion of extensional expansion proofs. The
proof translation mechanism developed by Pfenning is employed in the TRS-System in order to
translate matuting proofs [And76] into natural deduction proofs and it is applicable to other machine
oriented calculi as well. The only prerequisite is a translation from the deductive system used by
the theorem prover into extensional expansion proofs. The relation to the calculi sketched in
this thesis is thus quite obvious. By defining a respective translation algorithm from extensional
higher-order resolution into extensional expansion proofs, one can exploit Pfenning's translation
mechanism to generate natural deduction proofs.

The role and importance of extensionality in intensional type theory [Str93] is examined by
Hofmann in [Ho97]. He illustrates that the identity type in intensional type theory is not powerful
enough for the formalisation of mathematics or program development as it lacks the principles of
extensionality. Hofmann investigates to what extent extensional constructs of interest (i.e., quo-
tient types) can be added to intensional type theory without sacrificing decidability and existence
of canonical forms. He also points to the problem of adding full extensionality to intensional type
theory (which is then called extensional type theory) by proving the undecidability of typing in the
resulting system. A new approach to extensional equality in intensional type theory is presented in
[Al99].

The treatment of the functional extensionality principles in term rewriting in intuitionistic
type theory is furthermore addressed in [DCK93, DCK96].

Typical applications of extensional higher-order resolution (as well as extensional higher-order
paramodulation or RU&E-resolution) are comparisons of sets, functions or functionsals. Both in
mathematics as well as in applications like the verification of functional programs these concepts
play an important role. Hence, a subsystem that is capable of at least some basic extensionality
reasoning is very important for a mathematical assistant system like OMEGA [BCF+97] or a system
employed in program verification like PVS [ORS92] or Hol [GM93].

A rather new but nevertheless interesting and challenging application of extensional higher-
order resolution lies in the semantical construction in natural language processing. In this context
the use of higher-order unification and especially coloured higher-order unification [HK99, HK97],
for instance in the dissolving of ellipsis, has been illustrated in [DSP91, GKP91, GKP96, GK97].
The gain of full extensional higher-order theorem proving in this context is motivated by the
examples discussed in [KK98].

9.4 Primitive Equality in Higher-Order Theorem Proving

Equality is usually treated as a defined notion in approaches and systems for automated higher-
order theorem proving. This is probably the main reason why the problem of mechanising primitive
equality in higher-order logic while preserving Henkin completeness has (to the best knowledge of
the author) not been addressed in literature so far.

In contrast, the field of higher-order term rewriting and narrowing has become very active in
recent years (see [Pre98, NP98, NM98b, Nip95, Pre95, Pre94, VO94, Wo93]).

The probably most challenging tasks in this field addresses the development of confluent and
terminating term rewriting orders (see for instance [JR99, JR98, LP95]) For a summary to the
most recent approaches and developments we refer to [Pre98]. Higher-order term rewriting and
narrowing is obviously of great importance for the mechanisation of primitive equality in auto-
mated higher-order theorem proving, too. But one should not confuse higher-order term rewriting —
where the Boolean extensionality property does simply not occur — with higher-order equa-
tional reasoning. The difference has been illustrated in detail in Chapter 5, where we show that
if one is interested in problem domains which require a combined application of functional and
Boolean extensionality principles (e.g., when reasoning on sets described by characteristic func-
9.5. THEOREM PROVERS FOR HIGHER-ORDER LOGIC

In the last two decades several higher-order theorem proving systems have been built, among them are TPS [ABH96], HOL [GM93], or PVS [ORS92]. Neither one of these systems provides a full extensibility treatment in the sense of this thesis. The TPS-system, which is very likely the most powerful higher-order theorem prover currently available, at least provides a partial solution: it looks for equations between functional terms in input problems and modifies them by initially applying the extensibility principles in an appropriate way.

The ISABELLE-system [Pau94] and the TWELF-system [SP98] (TWELF is the successor of ELF [Pfe91]) provide logical frameworks, i.e., these systems provide logic languages powerful enough to allow for the specification of object logics as well as the specification of respective proof tactics in order to mechanise reasoning at the object logic level. Both systems have reached a considerable degree of automation and have been successfully applied in case studies in the area of programming languages and logics, e.g., the formal verification of Java Byte Code Verifier in Java and ISABELLE and the type preservation and value soundness properties of Mini ML with TWELF (see [Pfe96, Pfe]). Whereas the degree of automation in both systems is steadily increasing, both systems do, to best knowledge of the author, currently not address the problem of mechanising full extensibility reasoning.

Quite closely related to LEO is Konrad’s extensional higher-order theorem prover HOT [Kon98], which is based on the higher-order tableau calculus HTE [Koh98] that has already been mentioned above. HOT is implemented in Oz [Sh98, SSW94] and can thus exploit concurrency when branching within tableaux extensions. Therefore, and as HOT is based on more efficient data structures, HOT is (at least) on some examples more efficient as LEO (see [Kon98]). It seems to be interesting to reimplement LEO in Oz using the same data structures and then to compare the tableaux based prover HOT with LEO.

9.6 Examples

The presented examples can support the development of new approaches and systems for extensional higher-order theorem proving, as they point to the weaknesses of current approaches. Especially the examples discussed in Subsection 8.6 emphasize the difference between higher-order
term rewriting and higher-order equational reasoning. And it seems to be very interesting to investigate if and how these examples can be tackled in a pure term rewriting approach.
Chapter 10

Conclusion and Outlook

The role of equality and extensionality higher-order automated theorem proving is not well understood, in particular with respect to its potential for a mechanical treatment. To this end we have developed three calculi $ER$, $EP$, and $ERUE$, which improve the current situation. The practicability of calculus $ER$ has been demonstrated in case studies with the Leo-system.

Future work should be concerned with the calculi $ER$, $EP$, and $ERUE$ as stated in Section 8.7, namely to investigate the role of the FlexFlex-rule, to further restrict the application of the extensionality and equality rules, and to compare the three approaches in practical applications in order to find out, whether a primitive treatment of equality (as in $EP$ and $ERUE$) yields any advantages over a purely defined equality treatment (as in $ER$). Furthermore, the link to general higher-order $E$-unification needs further study.

Although the experiments with the MIZAR set theory were successful, I do not expect that such a classical theorem-proving approach will ever attain the problem solving expertise of human mathematicians. An integration of various heterogeneous reasoners (e.g., an integration of Tps with the extensionality reasoner Leo and first-order provers like Otter as motivated in [BBS99]) is a more promising alley. Clearly, in order to model human problem solving behaviour the crucial task will be to appropriately guide the collaboration of the integrated systems. Two options in this respect are: (i) to guide their collaboration by a proof planner (deliberative approach), and (ii) to realise the integrated systems as autonomous, specialised proof agents and to guide their collaboration on the basis of evaluations and resource allocations in a resource adaptive agent architecture (reactive approach). Also a mix of these two approaches as in the IntERRAP-architecture [Mü06] may be worthwhile to explore.

We shall concentrate on the latter option in order to realise a cooperation between Tps, Leo, and Otter in Omega, thereby exploiting the agent architecture that is already provided by the Omega-system (see [FHJ+99, BS98a, BS99]).
Bibliography


[BK98a] Christoph Benzmüller and Michael Kohlhase. Extensional higher-order resolution. In Kirchner and Kirchner [KK98a], pages 56–72.

[BK98b] Christoph Benzmüller and Michael Kohlhase. LEO – a higher-order theorem prover. In Kirchner and Kirchner [KK98a], pages 139–144.


BIBLIOGRAPHY


[Dah97] Ingo Dahn. Integration of automated and interactive theorem proving in ILF. In McCune [McC97a], pages 57–60.


[GLMS94] Christoph Goller, Reinhold Letz, Klaus Mayr, and Johann Schumann. SETHEO v3.2: Recent developments. In Bundy [Bun94], pages 778–782.


BIBLIOGRAPHY


BIBLIOGRAPHY


[RSG98] Julian Richardson, Alan Smaill, and Ian Green. Proof planning in higher-order logic with ACaml. In Kirchner and Kirchner [KK98a], pages 129–133.


[SP98] Carsten Schürmann and Frank Pfenning. Automated theorem proving in a simple meta-logic for LF. In Kirchner and Kirchner [KK98a], pages 286–300.


[Unkar] Unknown. *Unknown Title*. PhD thesis, Worcester College, Unknown Year. I have a manuscript of the thesis without a title page. (the author is a student advised by Professor Gandy).


[WGR96] Christoph Weidenbach, Bernd Gaede, and Georg Rock. Spass & flotter, version 0.42. In McRobbie and Slaney [MS96].


Index

Σ-algebra, 21
  full, 21
Σ-model, 22
Σ-valuation, 22
λ-calculus, 3
  simply typed, 3, 14
λ-conversion, 3
POST, 80
Keim, 12
Leo, 2, 12, 80
Leo clause store
  CONT, 83–85
  Uni-Cont, 85
  Ext-Mod, 85
  EXT, 83–86
  Factorised, 85
  Lightest, 84
  Paramod, 85
  Prim-Subst, 85
  Processed, 85
  Resolved, 84, 85
  SDS, 83, 84
  Unified, 85, 86
  USABLE, 83, 84
  set of support, 83
Ωmega, 80
Tps, 2, 5, 7–9
Keim-toolbox, 81
abstract consistency classes, 8, 31
  compactness of, 32
  non-atomic, 31
  properties for, 30
abstract consistency method, 5
abstract consistency properties, 30, 36
  with primitive equality, 36
abstract extension lemma, 39
admissibility, 16
  of generalised paramodulation rule, 70
  of generalised paramodulation rule, 75
application operator, 17
approximating binding, 15
assignment, 18
atom, 15
  axiom of choice, 4
  axioms of choice, 5
  base type, 14
  binding
    approximating, 15
    general, 15
    imitation, 15
    projection, 15
  bound variable, 15
calculus, 16
  EP, 11
  ER, 10, 11
  ERE, 11
calculus ratiocinator, 1
canonical projection, 18
carrier set, 17
characteristic function, 90
clause, 15
closed under subsets, 30
  compactness, 30, 32
  completeness, 5
  Henkin, 2
  of EP_c, 71
  of ER_c, 53
  of ERE_c, 78
comprehension
  principles, 3
comprehension principles, 3
congruence, 17
connective, 14
consistency
  non-atomic, 31
  constant, 14
  constraint, 16
  flex-flex, 16
  flex-rigid, 16
  convention, 55
conversion
α -, 31
βη-, 15
λ, 15
λ, 3
denotation, 19, 20
Denotatpflicht, 3
derivability, 16
of proper clauses, 55
description operator, 4, 5
difference-reduction, 29
domain, 17
eager unification, 54
empty clause, 16
equality
properties of Leibniz equality, 25
at defining, 5
defined, 11, 27, 28
Leibniz, 2, 5, 7, 11, 24, 27, 37
meta-, 5
notions of, 5
of clauses, 47
primitive, 5, 7, 8, 11, 26, 27, 37
semantical relation, 5, 7
evaluation
function, 6
in functional structures, 21
evaluation function, 18
expressiveness, 90
extension lemma, 30, 39
extensional higher-order paramodulation, 61
extensional higher-order resolution, 49, 72
extensional higher-order RUE-Resolution, 72
extensionality
additional rules for primitive equality, 11
Boolean, 2, 4, 5, 9, 11
Boolean, for Leibniz equality, 23, 27
Boolean, for primitive equality, 26, 27
for Leibniz equality, 23
for negative primitive equations, 61
for positive primitive equations, 61
for primitive and Leibniz equality, 27
for primitive equality, 26
full, for Leibniz equality, 23, 27
full, for primitive equality, 26, 27
functional, 2, 4, 5, 7, 9, 11
functional, for Leibniz equality, 23
functional, for primitive equality, 26
in models, 24
in models with primitive equality, 27
treatment, 7, 85
factorisation
generalised rules, 64
falsity, 22
fix-point free functions, 60
flexible literal heads, 10
formula, 14
frame, 17
free variable, 15
full pre-Σ-algebra, 21
fully invariant, 17
function application, 17
functional
congruence relation, 17, 40
pre-structure, 17
types, 14
functionality, 17
general binding, 10, 15
generalised factorisation rule, 64
generalised paramodulation rule, 63
generalised primitive substitution rule, 64
generalised resolution rules, 64
Henkin model, 7, 23
Henkin model with primitive equality, 26
higher-order logic, 14
Hilbert style calculus, 5
Hilbert’s program, 1
Hintikka lemma
for $\kappa_{\beta}$, 35
for $\kappa_{\beta}$, 33
for $\kappa_{\beta}$, 38
for $\kappa_{\beta}$, 35
for $\kappa_{\beta}$, 35
Hintikka set, 8, 30, 33
homomorphic extension, 20
homomorphism, 17
imitation bindings, 15
incompleteness, 80, 87, 88
of $\mathcal{E}_{\text{naive}}$, 59
of naive RUE-Resolution, 72
incompleteness theorems, 1
individuals, 6
interpretation, 6
function, 8
of constants, 17

Lego, Coq, type theory
constructive, 115
lifting lemma
for $\mathcal{E}_{\text{Pe}}, 62$
for $\mathcal{E}_{\text{Rc}}, 51$
INDEX

for TRUEc, 73
for clause normalisation, 50
lingua characteristica, 1
literal, 15
negative, 15
positive, 15
pre-, 15
proper, 15
unification constraint, 15
logical connectives, 14

maximal, 33
model, 6
Henkin, 23
Henkin, with primitive equality, 26
quotient, 23
standard, 6, 23
with primitive equality, 26

model class
Mββ, 23
Mβββ, 23
Mβb, 23
Mββb, 23

model existence, 39
model existence theorem, 40
models
functional, 8
non-functional, 8

normal form
βη, 4, 14
βη-head, 4
head-, 16, 52, 63, 74

order, 14, 112

paradoxes, 3
Russel’s, 3
parameters, 14

paramodulation, 57
first-order, 11
higher-order, 11
into unification constraints, 58

partial binding, 10

polarity, 15
pre-Σ-algebra, 21
pre-clause, 15
pre-literal, 15
pre-structure, 16, 17
singleton, 17
total, 17

primitive equality, 10, 36
primitive substitution, 10, 82, 88
generalised rules, 64
restricted, 112
projection bindings, 15

proper
clause, 15
clause in F and Ff, 71
clause in TRUE and TRUEf, 79
derivation, 50, 73
proper derivations, 62

property
b, 7, 8, 22
f, 22
q, 7, 8, 22, 23

property e, 26
proposition, 14
pure, 30

quotient
model, 23
pre-structure, 18
structure, 18, 19

reduction
βη, 15

reflexivity resolution, 58

resolution
extensional higher-order, 49, 72
gerenalised rules, 64
higher-order, 1, 8
higher-order constrained, 1
on unification constraints, 72

rule
GFac, 64
GPrim, 64
GRes1, 64
GRes2, 64
Equiv, 62
Func, 62

GPara, 63
Lib, 61
Lib, 54
Para, 57
Para, 57
Prim, 10
Sym, 73
admissible, 16
derivable, 16

satisfies, 22
saturated, 32
atomically, 32
saturation, 8
semantics
Henkin, 2, 5  
standard, 2, 5, 7  
semi-valuation, 8  
sentence, 14  
set  
of support, 12  
of variables, 14  
three, 90  
set of constants, 14  
set of support, 80  
signature, 14  
singleton pre-$\Sigma$-structure, 17  
Skolem term, 15  
Skolemisation, 9, 15  
solved  
for, 16  
variable, 16  
SOS-architecture, 80, 83  
soundness  
of $\beta$-equality, 20  
of CNF, 48  
of $\mathcal{P}_{\text{naive}}$, 59  
of extensional higher-order paramodulation, 62  
of extensional higher-order resolution, 49  
of extensional higher-order RUE-resolution, 73  
standard model, 23  
structure, 16, 18  
substitution, 15  
application of, 15  
subsumption, 12, 81  
extensional higher-order, 89  
higher-order, 89  
sufficiently pure, 30  
symmetric unification constraints, 73  
term indexing, 12  
extensional higher-order, 89  
higher-order, 12, 81  
syntactical higher-order, 89  
term rewriting, 4, 29  
higher-order, 10  
syntactical, 4  
term structure, 19  
$\beta$, 19  
$\beta\eta$, 19  
thereom equivalence  
of $\mathcal{R}_K$ and $\mathcal{R}_J$, 55  
of $\mathcal{R}_{KE}$ and $\mathcal{R}_{KE}^f$, 79  
of $\mathcal{P}$ and $\mathcal{P}_{\text{naive}}$, 71  
of $\mathcal{R}$ and $\mathcal{R}_{KE}$, 56  
of $\mathcal{P}_{\text{KE}}$ and $\mathcal{R}_{KE}$ (or $\mathcal{R}_{KE}^f$), 79  
truth, 22  
truth values, 6  
type, 14  
function, 17  
type theory  
classical, 1, 3, 5  
extensional, 114  
intensional, 114  
typed  
collection, 17  
collection of sets, 17  
mapping, 17  
typed binary relation, 17  
typed function, 17  
unification, 48  
continuation, 87  
cager, 54, 82, 88  
extensional higher-order, 82, 87  
higher-order, 1, 48, 49, 81  
higher-order $E$, 10, 49  
higher-order pre-, 1, 48  
unification constraint, 15  
unifying principle, 5  
universe of type $\alpha$, 17  
valid, 22  
valuation, 7, 22  
with equality, 26  
vale, 20  
variable, 14  
assignment, 6  
conditions, 9  
solved, 16  
variable assignment, 18  
weak derivability  
of generalised factorisation rule, 65, 74  
of generalised paramodulation rule, 68  
of generalised primitive substitution rule, 65, 74  
of generalised resolution rules, 65, 74
Symbol Index

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A/X]</td>
<td>... substitution, 15</td>
</tr>
<tr>
<td>[A/X]T</td>
<td>... application of substitution, 15</td>
</tr>
<tr>
<td>Aαβ</td>
<td>... abstract consistency class, 31, 41</td>
</tr>
<tr>
<td>Aβ</td>
<td>... abstract consistency class, 40</td>
</tr>
<tr>
<td>Aαβγ</td>
<td>... abstract consistency class, 31, 41</td>
</tr>
<tr>
<td>Aαβγδ</td>
<td>... abstract consistency class, 31, 41</td>
</tr>
<tr>
<td>PS</td>
<td>... example for incompleteness of S naïve, 59</td>
</tr>
<tr>
<td>P₂</td>
<td>... example for incompleteness of S naïve, 59</td>
</tr>
<tr>
<td>P₄</td>
<td>... example for incompleteness of S naïve, 60</td>
</tr>
<tr>
<td>P₆</td>
<td>... example for incompleteness of S naïve, 60</td>
</tr>
<tr>
<td>PSβ</td>
<td>... extensional higher-order paramodulation with full unification and unfolded clause normalisation, 49</td>
</tr>
<tr>
<td>SRβ</td>
<td>... extensional higher-order resolution with full unification and unfolded clause normalisation, 49</td>
</tr>
<tr>
<td>SRβE</td>
<td>... extensional higher-order RUE-resolution with full unification and unfolded clause normalisation, 72</td>
</tr>
<tr>
<td>Sβ</td>
<td>... extensional higher-order paramodulation with full unification, 61</td>
</tr>
<tr>
<td>SβE</td>
<td>... extensional higher-order resolution with full unification, 49</td>
</tr>
<tr>
<td>SβEβE</td>
<td>... extensional higher-order RUE-resolution with full unification, 72</td>
</tr>
<tr>
<td>S naïve</td>
<td>... naïve higher-order paramodulation, 59</td>
</tr>
<tr>
<td>S</td>
<td>... extensional higher-order paramodulation, 11, 61</td>
</tr>
<tr>
<td>Sβ</td>
<td>... extensional higher-order resolution, 10, 11, 49</td>
</tr>
<tr>
<td>SβE</td>
<td>... extensional higher-order RUE-resolution, 11, 72</td>
</tr>
<tr>
<td>S</td>
<td>... model class, 8</td>
</tr>
<tr>
<td>⊥</td>
<td>... Æípn equality, 5, 14, 37</td>
</tr>
<tr>
<td>EXT₁β</td>
<td>... functional extensionality for Leibniz equality, 9, 23, 27</td>
</tr>
<tr>
<td>EXT₂β</td>
<td>... Boolean extensionality for Leibniz equality, 9, 23, 27</td>
</tr>
<tr>
<td>Mβ</td>
<td>... model class, 8</td>
</tr>
<tr>
<td>Mαβ</td>
<td>... model class, 6, 8</td>
</tr>
<tr>
<td>Mαβγ</td>
<td>... model class, 7, 8</td>
</tr>
<tr>
<td>Mαβγδ</td>
<td>... model class, 7, 8</td>
</tr>
<tr>
<td>Mαβγδζ</td>
<td>... model class, 7, 8</td>
</tr>
<tr>
<td>Mαβγδζζ</td>
<td>... meta-equality, 5, 14</td>
</tr>
<tr>
<td>Mαβγδζζζ</td>
<td>... α-equality, 15</td>
</tr>
<tr>
<td>Mαβγδζζζζ</td>
<td>... βη-equality, 15, 17</td>
</tr>
<tr>
<td>Mαβγδζζζζζ</td>
<td>... β-equality, 15, 17</td>
</tr>
<tr>
<td>Mαβγδζζζζζζ</td>
<td>... η-equality, 15</td>
</tr>
<tr>
<td>H</td>
<td>... logical conjunctive, 14</td>
</tr>
<tr>
<td>∆</td>
<td>... primitive equality, 5, 14, 37</td>
</tr>
<tr>
<td>EXTαβ</td>
<td>... functional extensionality for primitive equality, 26, 27</td>
</tr>
<tr>
<td>EXTαβγ</td>
<td>... Boolean extensionality for primitive equality, 26, 27</td>
</tr>
<tr>
<td>GT</td>
<td>... model class, 6</td>
</tr>
<tr>
<td>Σ</td>
<td>... signature, 14</td>
</tr>
<tr>
<td>Σ=</td>
<td>... signature with primitive equality, 14</td>
</tr>
<tr>
<td>A/X</td>
<td>... application of substitution with subsequent head-normalisation, 15</td>
</tr>
<tr>
<td>T₉</td>
<td>... α-normal form, 15</td>
</tr>
<tr>
<td>T₉β</td>
<td>... β-normal form, 15</td>
</tr>
<tr>
<td>T₉η</td>
<td>... η-normal form, 15</td>
</tr>
<tr>
<td>T₉βη</td>
<td>... head-normal form, 15</td>
</tr>
<tr>
<td>T₆</td>
<td>... term structure, 19</td>
</tr>
<tr>
<td>TS(Σ)</td>
<td>... set of variables, 14</td>
</tr>
<tr>
<td>wff(Σ)</td>
<td>... Σ-terms of type α, 14</td>
</tr>
<tr>
<td>wff(Σ)</td>
<td>... set of Σ-terms, 14</td>
</tr>
<tr>
<td>wff(Σ)</td>
<td>... set of Σ-terms, 14</td>
</tr>
<tr>
<td>wff(Σ)</td>
<td>... set of propositions, 14</td>
</tr>
<tr>
<td>cuwff(Σ)</td>
<td>... set of sentences, 14</td>
</tr>
<tr>
<td>να</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νβ</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νγ</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νδ</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νξ</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>νη</td>
<td>... abstract consistency property, 30</td>
</tr>
<tr>
<td>@</td>
<td>... application operator, 17</td>
</tr>
<tr>
<td>@αβ</td>
<td>... application operator for equivalence</td>
</tr>
</tbody>
</table>