Cumulative Habilitation Script

Faculty 6 - Natural Sciences and Technology I

Saarland University

HD Dr. Christoph Benzmüller

Frankfurt am Main, 8. September 2006

5 Cumulative Habilitation Script

5.1 Introduction

Since more than a decade my main research interests are:

1. The (collaborative) development of large and integrated mathematics assistance systems in the emerging mathematical semantic web. These systems shall fruitfully support education (e.g., e-learning) and research in mathematics, formal methods, and engineering.

2. The study of the theoretical foundations (model theory and proof theory) of higher-order logic.

3. The improvement of automated theorem proving techniques in higher-order logic.

Higher-order logic constitutes the base representation framework of many leading mathematics assistance systems (e.g., Isabelle[1], HOL[2], PVS[3], and our own OMEGA system[4]). Furthermore, many mathematical textbooks naturally employ higher-order logic constructs. Therefore it is not surprising that the currently fast evolving repositories of formalized mathematics contain a significant amount of higher-order logic encodings. Thus, topics (2) and (3) can be characterized as important subtopics for the overall research goal (1).

The envisioned all-embracing assistance systems for mathematics cover a wide range of typical characteristics an ambitious, integrated AI system shall have. Therefore, research goal (1) in addition to (2) and (3) requires the combination of techniques from several subfields of AI including knowledge representation and reasoning, cognitive architectures and multi-agent systems, human computer interaction and user interfaces, machine learning, intelligent tutor systems, and dialog systems and natural language processing.

My PhD thesis has concentrated on tasks (2) and (3). Parallel to my PhD work and in particular adjacent to it I have performed and supervised research (e.g., as PostDoc and Research Fellow in Germany, the UK, and the USA, as head of the OMEGA project of Jörg Siekmann, and as principal investigator of two projects in the SFB 378 in Saarbrücken) in the wider range of research topics as required for goal (1). These research activities are documented by a wide range of journal, conference, and workshop publications (see the selected recent publications in Section 1, my complete list of publications in Section 3, and the selected publications for this cumulative habilitation document as given below) as well as by my activities as organizer and PC member of various related conferences and workshops, as scientific coordinator of the EU RTN Calculemus (2000-2004) and my recent editorship of a special issue in the Journal of Applied Logic on mathematics assistance systems. The following text, which addresses the challenge of building mathematics assistance systems and which I present here as a personal research statement, has been adopted from my editorial of this special issue (see [J13-06] in Section 3).

What is an assistance system for mathematics and what is it good for?

The notion of an assistance system for mathematics adopted here characterizes an integrated environment of tools supporting a wide range of typical research, publication and knowledge management activities. Examples of mathematical activities are computing, proving, solving, modeling, verifying, structuring, maintaining, searching, inventing, paper writing, explaining, illustrating, and possibly others. Clearly, some of them require a high amount of human ingenuity while others do not. An assistance system for mathematics should support activities for which practical and robust solutions exist, that is, at the moment predominantly those which require less human ingenuity.

[1] `www.cl.cam.ac.uk/Research/HVG/Isabelle`
[2] `hol.sourceforge.net`
[3] `pvs.csl.sri.com`
[4] `www.ags.uni-sb.de/~omega/`

Meanwhile an impressive range of mathematical support tools is actually available, for instance, computer algebra systems (e.g., MAPLE and MATHEMATICA), interactive proof assistants (e.g., ISABELLE/HOL and COQ), automated theorem provers (e.g., VAMPIRE and OTTER), model checkers (e.g., SMV), partially integrated hybrid systems (e.g. OMEGA), search engines (e.g., GOOGLE), and publishing and typesetting packages (e.g., LATEX). The integration of one or several of these tools within a uniform environment leads to our notion of an integrated mathematics assistance system. The overall idea, however, is not to replace the mathematician (or engineer or teacher) but instead to support a fruitful symbiosis of human and machine intelligence in which the computer takes over tedious routine parts thus setting precious resources free for the human user.

An obvious and very prominent approach to the development of an assistance system for mathematics is the integration of off-the-shelf tools, for instance, automated theorem provers, decision procedures, and computer algebra systems, into interactive proof assistants. An important issue in this approach is the provision of transformational mappings between the different representations employed in the combined tools. Furthermore, the maintenance and effective management of formalized bits of mathematical knowledge in structured (and probably distributed and shared) knowledge bases has to be addressed. Syntactic and semantic search facilities are required for retrieving knowledge from these knowledge sources. Bridging the gap between informal multimodal mathematical texts and fully formalized representations is just as important as the combination with powerful publication and typesetting packages. In order to reduce the duplication and multiplied encoding effort as currently still required in computer-supported mathematics, we need a smooth and formal transition from technical developments within an assistance system back and forth to high-quality publications. Another important issue is the development of powerful, uniform look-and-feel as well as effective user interfaces which preferentially support a human-oriented rather than a machine-oriented interaction with the system. They should hide the minute representational and operational details of the integrated tools. Many support tools and the mathematical knowledge sources can ideally be shared between different assistance systems through the development of a mathematical semantic web.

And who needs assistance systems for mathematics?

Computer algebra systems and publishing tools, for example, are already routinely employed in mathematical research and practice today. Furthermore, interactive proof assistants and model checkers are nowadays used in industrial applications for formal software and hardware verification and quality assurance. On the other hand mathematics has existed for thousands of years without computer support and it is perfectly valid to doubt, as many working mathematicians actually do, that the immediate impact of the envisioned assistance systems will be overwhelming for the frontiers of mathematical research.

In recent years, however, we can observe a small but increasing number of success stories in computer aided mathematics. For example, the four color theorem has been proven in 1976 by Appel and Haken with significant computer support. This proof had a dubious status for a long time because a verification of it (by hand) seemed impossible. Recently, however, a formal verification within the assistance system COQ was reported by Georges Gonthier at Microsoft Research. Another success story is the verification of a proof of the prime number theorem with the system ISABELLE by Jeremy Avigad at Carnegie Mellon University in 2004.

Presumably the most important recent example is the computer supported proof of Kepler's conjecture by Thomas Hales at Pittsburgh University. Kepler's conjecture is a problem in discrete geometry which has been unsolved for nearly 400 years. The submission of his results to the Annals of Mathematics resulted in an interesting and controversial debate. Robert D. MacPherson, the editor in chief of the Annals of Mathematics, gave a presentation at the symposium 'The nature of mathematical proof' of the British Royal Society in London in Fall 2004 in which he revealed how difficult it is to review results of this nature: a refereeing board of 12 mathematicians had finally given up to fully verify the proof after 4 years! They could still validate Hales' reduction of the original problem to a wide range of subproblems. However, they were not able to verify (nor

to refute) the many subcriteria that Hales solved with significant computer algebra support. This happened for the first time in the history of mathematics! As Hilbert's famous perpetual call from the heart exemplifies: "Da ist das Problem, suche die Lösung. Du kannst sie durch reines Denken finden, denn in der Mathematik gibt es keinen Ignorabimus"[5], mathematicians have always held the belief that in principle we know – although we may err – if something is the case or not.

While mathematicians have thus given up on verifying the proof, Hales has started the Flyspeck project. The aim of this project is to reconstruct, formalize, and fully verify Hales complete proof in the assistance system HOL-LIGHT. This is an a posteriori attempt to apply assistance systems in a research frontier of mathematics and due to the complexity of the problem and the comparative mathematical and practical immaturity of today's mathematical assistance systems this endeavor will certainly require several years of persistent work.

In the long run, however, the envisioned fully integrated assistance systems will support this new style of mathematics not a posteriori but from the very start, ideally with far less effort as currently still required and also at a more human-friendly interaction level.

Is there some low hanging fruit?

Yes, there is. Even in case of a failure of the ambitious Flyspeck project, the existing systems are already successfully used in less ambitious mathematics such as formal verification in computer science. In particular students who want to learn mathematics or engineers who want to apply mathematics – both groups are typically confronted with far less ambitious mathematical problems than Hales – may well and actually do already benefit from current mathematics assistance systems. In fact, proof assistants and model checkers have been widely used in applications for software and hardware verification. Also e-learning environments with integrated support tools increasingly attract attention in academia as well as in public applications.

Why is it so difficult to build an integrated assistance system for mathematics?

The challenge is to attack the scientific and technological gap between the targeted ideal mathematics assistance environments and the many weaknesses and shortcomings of the current systems. This requires in particular the combination of techniques and expertise from several research areas. Research progress and good research training in this multidisciplinary area can currently probably be best achieved by joining forces in research networks. One example is the European CALCULEMUS research training network (2000-2004), which has put an emphasis on the training of young researchers in the areas of computer algebra and deduction systems.

Actually, there are relatively few research groups which have sufficient expertise, background and critical mass to cover the whole spectrum of relevant research issues to build an all embracing assistance system for mathematics. This problem is actually analogous to the development of large and all-encompassing AI systems in general; in fact, these assistance systems can be seen as an instance of an ambitious, integrated and general AI system, which researchers claim also in other more common subfields of AI.[6] However, a broad research expertise is only one of the many essential requirements. Availability of human resources, in particular, talented and enthusiastic PhD students with strong implementational skills is another. In fact, most of the existing attempts at large and integrated assistance systems have been predominantly achieved with the help of generations of PhD students and postdocs.[7] Such a student-based development strategy imposes several challenges, not least of which is the software maintenance problem, which is particularly

[5] Engl.: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no ignorabimus.

[6] In their invited talks at this years AAAI-05 conference in Pittsburgh both Ronald J. Brachman and Marvin Minsky argued for building and analyzing large, integrated AI systems. I should think that the envisioned all-embracing assistance systems for mathematics actually cover a wide range of these typical characteristics an ambitious, integrated AI system will have as well.

[7] An example is Peter Andrews' TPS system, which is based on the contributions of a row of students such as Dale Miller, Frank Pfenning, Dan Nesmith, Sunil Issar, Hongwei Xi, Matthew Bishop, and Chad Brown. Another example is our own OMEGA project with its long sequence of PhDs and postdocs.

difficult for those groups which do not have the support of an experienced and long-term employed software engineer to control and guarantee a persistent high quality software development along uniform conventions. Probably even harder is the organization of a smooth knowledge transfer in order to pass crucial system expertise from one generation of students to the next. PhD students and researchers in the area of mathematics assistance systems need in addition to scientific talent and implementational skills a broad research interest, excellent communication skills, social competence and teamwork spirit.

An important challenge is to identify the best of todays achievements and to integrate them into a single best practice environment. In order to achieve significant progress in our research area the best research strategy is debatable. Two options are "Let the best system win" and "Cooperate, modularize, and exchange components". I personally advocate the latter – however, time will tell.

5.2 Selected Publications

The following selected publications well document my personal research activities on Higher-Order Logics and Mathematics Assistance Systems in the last decade. The given percentages are estimations of my personal contribution to each paper.

Higher-Order Semantics

[50%] C. Benzmüller and C. Brown, **A Structured Set of Higher-Order Problems**. TPHOLs 2005, no.3606 in LNAI, pp.66-81, Oxford, UK, 2005. ©Springer.

[33%] C. Benzmüller, C. Brown, and M. Kohlhase. **Higher-Order Semantics and Extensionality**. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004. ©JSTOR.

Higher-Order Proof Theory

[40%] C. Benzmüller, C. Brown, and M. Kohlhase, **Cut-Simulation in Impredicative Logics.** IJCAR'06, no.4130 in LNAI, pp.220-314, Seattle, USA, 2006. ©Springer.

Higher-Order Theorem Proving

[100%] C. Benzmüller. **Comparing Approaches to Resolution based Higher-Order Theorem Proving**. Synthese, An International Journal for Epistemology, Methodology and Philosophy of Science, 133(1-2):203–235, 2002. ©Kluwer.

[100%] C. Benzmüller. **Extensional Higher-Order Paramodulation and RUE-Resolution**. CADE-16, no.1632 in LNAI, pp.399–413, Trento, Italy, 1999. ©Springer.

[60%] C. Benzmüller and M. Kohlhase. **Extensional Higher-Order Resolution**. CADE-15, no.1421 in LNAI, pp.56–71, Lindau, Germany, 1998. ©Springer.

Intergration of Reasoning Systems

[60%] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber, **Can a Higher-Order and a First-Order Theorem Prover Cooperate?** LPAR-11, no.3452, pp.415-431, Montevideo, Uruguay, 2005. ©Springer.

[50%] C. Benzmüller and V. Sorge. **OANTS – An Open Approach at Combining Interactive and Automated Theorem Proving.** In Symbolic Computation and Automated Reasoning, pp.81–97, 2000. ©A.K.Peters.

[50%] C. Benzmüller, M. Bishop and V. Sorge. **Integrating TPS and OMEGA**. *Journal of Universal Computer Science*, 5:188–207, 1999. ©Springer.

Mathematics Assistance Systems

[40%] J. Siekmann, C. Benzmüller, and S. Autexier, **Computer Supported Mathematics with OMEGA.** Special Issue on Mathematics Assistance Systems, Journal of Applied Logic. ©Elsevier. In print, 2006.

[40%] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Norma and M. Pollet, **Proof Development in OMEGA – The Irrationality of Square Root of 2**. In Thirty Five Years of Automating Mathematics, pp.271–314, 2003. ©Kluwer Applied Logic Series, Volume 28.

Tutorial Dialog with Mathematics Assistance Systems

[60%] C. Benzmüller and Q.B. Vo, **Mathematical Domain Reasoning Tasks in Tutorial Natural Language Dialog on Proofs**. AAAI-05, Pittsburgh, Pennsylvania, 2005. USA. ©AAAI Press / The MIT Press.

[40%] M. Buckley and C. Benzmüller, **An Agent-based Architecture for Dialogue Systems.** Perspectives of System Informatics (PSI'06), Novosibirsk, Akademgorodok, Russia, 2006. ©Springer LNAI. In print.

[25%] C. Benzmüller, H. Horacek, H. Lesourd, I. Kruijff-Korbayova, M. Schiller, M. Wolska, **DiaWOz-II - A Tool for Wizard-of-Oz Experiments in Mathematics.** KI 2006, Bremen, Germany, 2006. ©Springer LNAI. In print.

A Structured Set of Higher-Order Problems

Christoph E. Benzmüller and Chad E. Brown

Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany
www.ags.uni-sb.de/{~chris, ~cebrown}

Abstract. We present a set of problems that may support the development of calculi and theorem provers for classical higher-order logic. We propose to employ these test problems as quick and easy criteria preceding the formal soundness and completeness analysis of proof systems under development. Our set of problems is structured according to different technical issues and along different notions of semantics (including Henkin semantics) for higher-order logic. Many examples are either theorems or non-theorems depending on the choice of semantics. The examples can thus indicate the deductive strength of a proof system.

1 Motivation: Test Problems for Higher-Order Reasoning Systems

Test problems are important for the practical implementation of theorem provers as well as for the preceding theoretical development of calculi, strategies and heuristics. If the test theorems can be proven (resp. the non-theorems cannot) then they ideally provide a strong indication for completeness (resp. soundness). Examples for early publications providing first-order test problems are [21,29,23]. For more than decade now the TPTP library [28] has been developed as a systematically structured electronic repository of first-order test problems. This repository together with the yearly CASC theorem prover competitions [24] significantly supported the improvement of first-order and propositional reasoning systems. Unfortunately, a respective library of higher-order test problems is not yet available.

This paper presents a small set of significant test problems for classical higher-order logic that may guide the development of higher-order proof systems. These test problems are relevant for both automated and interactive higher-order theorem proving. Even some of our simpler theorems may be difficult to prove interactively. Examples are our problems 15(a): $p_{o\to o}\,(a_o \wedge b_o) \Rightarrow p\,(b \wedge a)$ and 16: $(p_{o\to o}\,a_o) \wedge (p\,b_o) \Rightarrow (p\,(a \wedge b))$.

Most of the examples presented here are chosen to be a simple representative of some particular technical or semantical point. We also include examples illustrating real challenges for higher-order theorem provers. Our work is relevant in the first place for theorem proving in classical higher-order logic. However, many of our examples also carry over to other logics such as intuitionistic higher-order logic. Most of the presented test problems evolved from experience gained in the development of the higher-order theorem provers TPS [5] and LEO [10,7]. Some of the examples and (many others) have been also discussed in other publications on classical higher-order logic, e.g. [15,17,6,1,4]. The novel contribution of this paper is not the test problems per se, but the connection of these examples with the particular model classes in which they are valid (resp. invalid) and their assemblage into a comprehensive set.

J. Hurd and T.F. Melham (Eds.): TPHOLs 2005, LNCS 3603, pp. 66–81, 2005.

We structure many of our examples along two dimensions. The examples are theorems or non-theorems depending on these dimensions.

Extensionality provides one dimension in which we can vary semantics. Assuming Henkin semantics, for instance, most of our examples denote theorems. If we choose a weaker semantics, for instance, by omitting Boolean extensionality, then some test problems become non-theorems providing a test case for soundness with respect to this more general notion of semantics (in which fewer propositions are valid). By varying extensionality, we have defined a landscape of eight higher-order model classes and developed abstract consistency methods and model existence results in [8,9]. This landscape of higher-order model classes and the corresponding abstract consistency framework provides much needed support for the theoretical analysis of the deductive power of calculi for higher-order logic. The test problems we introduce in this paper provide quick and easy test criteria for the soundness and completeness of proof systems with respect to these model classes. Testing a proof system with our examples should thus precede a formal, theoretical soundness and completeness analysis with the abstract consistency methodology introduced in [8,9].

Set comprehension provides another dimension along which one can vary semantics. In [14] different model classes are defined depending on the logical constants which occur in the signature. Since many sets are only definable in the presence of certain logical constants, this provides a way of varying the sets which exist in a model. In this paper, we provide examples of theorems which are only provable if one can use certain logical constants for instantiations. In implementations of the automated theorem provers TPS and LEO the problem of instantiating set variables corresponds to the use of *primitive substitutions* described in [14,2,3].

Section 2 introduces the syntax of classical higher-order logic following Church [15]. Section 3 presents some first test problems for pre-unification and quantifier dependencies. In Section 4 we review a landscape of higher-order semantics that distinguishes higher-order models with respect to various combinations of Boolean extensionality, three forms of functional extensionality and different signatures of logical constants. Section 5 provides test problems that are structured according to the introduced landscape of model classes. Section 6 presents some more complex test problems.

2 Classical Higher-Order Logic

As in [15], we formulate higher-order logic ($\mathcal{HOL}$) based on the simply typed λ-calculus. The set of simple types $\mathcal{T}$ is freely generated from basic types o and ι using the function type constructor $\to$.

For formulae we start with a set $\mathcal{V}$ of (typed) variables (denoted by $X_\alpha, Y, Z, \ldots$) and a signature Σ of (typed) constants (denoted by $c_\alpha, f_{\alpha\to\beta}, \ldots$). We let $\mathcal{V}_\alpha$ (Σ_α) denote the set of variables (constants) of type α. A signature Σ of constants may include logical constants from the set $\overline{\Sigma}$ defined by

$$\{\top_o, \bot_o, \neg_{o\to o}, \wedge_{o\to o\to o}, \vee_{o\to o\to o}, \Rightarrow_{o\to o\to o}, \Leftrightarrow_{o\to o\to o}\}$$

$$\cup\{\Pi^\alpha_{(\alpha\to o)\to o} \mid \alpha\in\mathcal{T}\}\cup\{\Sigma^\alpha_{(\alpha\to o)\to o} \mid \alpha\in\mathcal{T}\}\cup\{=^\alpha_{\alpha\to\alpha\to o} \mid \alpha\in\mathcal{T}\}.$$

Other constants in a signature are called parameters. The constants Π^α and Σ^α are used to define $\forall$ and $\exists$ (see below) without introducing a binding mechanism other than λ. The set of $\mathcal{HOL}$-formulae (or terms) over Σ are constructed from typed variables and constants using application and λ-abstraction. We let $\mathit{wff}_\alpha(\Sigma)$ be the set of all terms of type α and $\mathit{wff}(\Sigma)$ be the set of all terms. We use $\mathbf{A}, \mathbf{B}, \ldots$ to denote terms in $\mathit{wff}_\alpha(\Sigma)$.

We use vector notation to abbreviate k-fold applications and abstractions as $\mathbf{A}\overline{\mathbf{U}^k}$ and $\lambda\overline{X^k}.\mathbf{A}$, respectively. We also use Church's dot notation so that . stands for a (missing) left bracket whose mate is as far to the right as possible (consistent with given brackets). We use infix notation $\mathbf{A} \vee \mathbf{B}$ for $((\vee\mathbf{A})\mathbf{B})$ and binder notation $\forall X_\alpha.\mathbf{A}$ for $(\Pi^\alpha(\lambda X_\alpha.\mathbf{A}_o))$. While one can consider $\wedge$, $\Rightarrow$ and $\Leftrightarrow$ to be defined (as in [8]), we consider these members of the signature Σ. We also use binder notation $\exists X.\mathbf{A}$ as shorthand for $\Sigma^\alpha(\lambda X.\mathbf{A})$ if Σ^α is a constant in Σ. We let $(\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha)$ denote the Leibniz equation $\forall P_{\alpha\to o}.(P\mathbf{A}) \Rightarrow .P\mathbf{B}$.

Each occurrence of a variable in a term is either free or bound by a λ. We use $free(\mathbf{A})$ to denote the set of free variables of $\mathbf{A}$ (i.e., variables with a free occurrence in $\mathbf{A}$). We consider two terms to be equal (written $\mathbf{A} \equiv \mathbf{B}$) if the terms are the same up to the names of bound variables (i.e., we consider α-conversion implicitly). A term $\mathbf{A}$ is closed if $free(\mathbf{A})$ is empty. We let $\mathit{cwff}_\alpha(\Sigma)$ denote the set of closed terms of type α and $\mathit{cwff}(\Sigma)$ denote the set of all closed terms. Each term $\mathbf{A} \in \mathit{wff}_o(\Sigma)$ is called a proposition and each term $\mathbf{A} \in \mathit{cwff}_o(\Sigma)$ is called a sentence.

We denote substitution of a term $\mathbf{A}_\alpha$ for a variable X_α in a term $\mathbf{B}_\beta$ by $[\mathbf{A}/X]\mathbf{B}$. Since we consider α-conversion implicitly, we assume the bound variables of $\mathbf{B}$ avoid variable capture.

Two common relations on terms are given by β-reduction and η-reduction. A β-redex $(\lambda X.\mathbf{A})\mathbf{B}$ β-reduces to $[\mathbf{B}/X]\mathbf{A}$. An η-redex $(\lambda X.\mathbf{C}X)$ (where $X \notin free(\mathbf{C})$) η-reduces to $\mathbf{C}$. For $\mathbf{A}, \mathbf{B} \in \mathit{wff}_\alpha(\Sigma)$, we write $\mathbf{A}{\equiv_\beta}\mathbf{B}$ to mean $\mathbf{A}$ can be converted to $\mathbf{B}$ by a series of β-reductions and expansions. Similarly, $\mathbf{A}{\equiv_{\beta\eta}}\mathbf{B}$ means $\mathbf{A}$ can be converted to $\mathbf{B}$ using both β and η. For each $\mathbf{A} \in \mathit{wff}(\Sigma)$ there is a unique β-normal form (denoted $\mathbf{A}{\downarrow_\beta}$) and a unique $\beta\eta$-normal form (denoted $\mathbf{A}{\downarrow_{\beta\eta}}$). From this fact we know $\mathbf{A}{\equiv_\beta}\mathbf{B}$ ($\mathbf{A}{\equiv_{\beta\eta}}\mathbf{B}$) iff $\mathbf{A}{\downarrow_\beta} \equiv \mathbf{B}{\downarrow_\beta}$ ($\mathbf{A}{\downarrow_{\beta\eta}} \equiv \mathbf{B}{\downarrow_{\beta\eta}}$).

A non-atomic formula in $\mathit{wff}_o(\Sigma)$ is any formula whose β-normal form is $(c\overline{\mathbf{A}^n})$ where c is a logical constant. An atomic formula is any other formula in $\mathit{wff}_o(\Sigma)$.

Many of the example problems in this paper employ equality, e.g. $\neg(a = \neg a)$. We have different options for the encoding of equality. We can either use primitive equality (i.e., equality as a logical constant) or use some definition of equality in terms of other logical constants. A common definition is Leibniz equality ($\forall P_{\alpha\to o}.(P\mathbf{A}) \Rightarrow .P\mathbf{B}$), but others are possible (see Exercise **X5303** in [4]). In many examples we will denote equality by $\stackrel{*}{=}$ (e.g., $\neg(a \stackrel{*}{=} \neg a)$). For each different interpretation of equality, we obtain a different example. We will discuss conditions under which different choices lead to theorems and which choices lead to non-theorems.

For some types, one can also define equality extensionally. For example, one can use equivalence instead of equality at type o. Similarly, at any type $\alpha \to o$, we introduce $\stackrel{set}{=}$ to denote set equality, i.e., $\stackrel{set}{=}$ is an abbreviation for

$$\lambda U_{\alpha\to o}\lambda V_{\alpha\to o}\forall X_\alpha.UX \Leftrightarrow VX.$$

In some cases, the use of an extensional definition of equality yields a theorem which can be proven without *assuming* extensionality. We will *not* use the notation $\stackrel{*}{=}$ to refer to any extensional definition of equality. Interpreting $\stackrel{*}{=}$ extensionally would significantly change some of the discussion below.

3 Test Problems for Pre-unification and Quantifier Dependencies

Higher-order pre-unification (see [26]) and higher-order Skolemization (see [22]) are important basic ingredients for building an automated higher-order theorem prover. They are largely independent of the chosen semantics for higher-order logic with one exception: β versus $\beta\eta$. As noted in [18] the unification problem relative to β-conversion is different from the unification problem relative to $\beta\eta$-conversion.

3.1 Pre-unification

Implementing a sound, complete and efficient pre-unification algorithm for the simply typed λ-calculus is a highly non-trivial task. Since higher-order pre-unification extends standard first-order unification all first-order test problems in the literature also apply to the higher-order case.

Some specific higher-order test problems can be obtained from the literature on higher-order unification and pre-unification, for example [26,25]. We will now illustrate how further challenging test examples can be easily created using Church numerals.

Church numerals are usually employed in the context of the untyped λ-calculus to encode the natural numbers. This encoding can be partly transformed in a simply typed or polymorphic typed λ-calculus. This includes the definition of successor, addition and multiplication which we employ in or test problems.

Iteration is the key concept to encode natural numbers as Church numerals. For each type α, we can define the Church numeral $\overline{n}^{\alpha}$ by $(\lambda F_{\alpha\to\alpha}\lambda Y_{\alpha}.(F^n Y))_{(\alpha\to\alpha)\to(\alpha\to\alpha)}$ where $(F^n Y)$ is shorthand for $\underbrace{(F\ (F \ldots (F}_{n-times}\ Y)))$. We will often write $\overline{n}$ instead of $\overline{n}^{\alpha}$, leaving the dependence on the type implicit. Omitting types[1], the successor function $\overline{s}$ can be defined as $\lambda N\lambda F\lambda Y.F(NFY)$, addition $\overline{+}$ as $\lambda M\lambda N\lambda F\lambda Y.MF(NFY)$ and multiplication $\overline{\times}$ as $\lambda M\lambda N\lambda F\lambda Z.N(MF)Z$. To ease notation, we write $\overline{+}$ and $\overline{\times}$ in infix.

Arithmetic equations on Church numerals such as $\overline{3}\overline{\times}\overline{4} \stackrel{*}{=} \overline{5}\overline{+}\overline{7}$ or $(((\overline{1}\overline{0}\overline{\times}\overline{1}\overline{0})\overline{\times}\overline{1}\overline{0}) \stackrel{*}{=} ((\overline{1}\overline{0}\overline{\times}\overline{5})\overline{+}(\overline{5}\overline{\times}\overline{1}\overline{0}))\overline{\times}\overline{1}\overline{0}))$ provide highly suited test problems for the efficiency of β-conversion or $\beta\eta$-conversion in the proof system. Of course, in order to correctly implement β- and η-conversion, one must first properly implement α-conversion.

We obtain more challenging test problems if we employ pre-unification for synthesizing Church numerals and arithmetical operations.

Example 1. (Solving arithmetical equations using pre-unification) The following examples are provable using pre-unification for β-conversion.

[1] N, M are of type $(\alpha \to \alpha) \to (\alpha \to \alpha)$, F is of type $\alpha \to \alpha$, and Y, Z are of type α.

(a) $\exists N_{(\iota\to\iota)\to\iota\to\iota}.((N\overline{\times 1}) \stackrel{*}{=} \overline{1})$ (There are two solutions, $\overline{1}$ and $(\lambda F_{\iota\,\iota}.F)$, if one only assumes β-conversion. There is one solution assuming $\beta\eta$-conversion.)
(b) $\exists N.(N\overline{\times 4}) \stackrel{*}{=} \overline{5+7}$
(c) $\exists H.(((H\bar{2})\bar{3}) \stackrel{*}{=} \bar{6}) \wedge (((H\bar{1})\bar{2}) \stackrel{*}{=} \bar{2}))$
(d) $\exists N, M.(N\overline{\times 4}) \stackrel{*}{=} \overline{5+}M$ (There are infinitely many solutions to this problem.)

3.2 Quantifier Dependencies

In proof search with tableaux and expansion proofs, variable conditions can be used to encode quantifier dependencies. Of course, one must be careful to obtain a sound framework. For instance, the variable conditions added with each eliminated existential quantifier in the framework used in [20] allow (incorrect) proofs of the following first-order non-theorems:

Example 2. (First-order non-theorems)

(a) (Example 2.9 in [30]) $(\exists X_\iota \forall Y_\iota . q_{\iota\to\iota\to o} XY) \vee (\exists U_\iota \forall V_\iota . \neg qVU)$
(b) (Example 2.50 in [30]) $\exists Y_\iota \forall X_\iota . ((\forall Z_\iota . q_{\iota\to\iota\to o} XZ) \vee (\neg qXY))$

In [19] an attempt was made to use variable conditions in the context of resolution theorem proving (for a sorted extension of higher-order logic) instead of introducing Skolem terms. However, the system was unsound as it allowed a resolution refutation proving the following non-theorem:

Example 3. (Non-Theorem: Every function has a fixed point) $\forall F_{\alpha\to\alpha}.\exists X_\alpha.F\ X \doteq X$. The idea is that one obtains two single-literal clauses $(P_{\iota\to o}(FX))$ and $\neg(PY)$ using clause normalization and variable renaming (where X and Y can be instantiated). One then obtains the empty clause by unifying Y with (FX).

Skolem terms avoid incorrect proofs of such theorems since the Skolem terms will preserve the relationship between renamed variables in different clauses. In particular, if S is a Skolem function, we would obtain single-literal clauses $(S_{\iota\to\iota\to o} X(FX))$ and $\neg(S_{\iota\to\iota\to o} YY)$ which cannot be resolved and unified.

There is a relationship between Skolemization and the axiom of choice in the first-order case which becomes more delicate in the higher-order case. Consider formulas $\forall x_\iota \exists y_\iota \varphi(x,y)$ and $\forall x_\iota \varphi(x,(f_{\iota\to\iota} x))$. In first-order logic, the two formulas are equivalent with respect to satisfiability whenever f does not occur in φ. The equivalence follows from the fact that any first-order model (with domain $\mathcal{D}_\iota$) satisfying $\forall x \exists y \varphi(x,y)$ can be extended to interpret f as a function $g : \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$ such that $\forall x \varphi(x,(fx))$ holds. In general, the axiom of choice (at the *meta-level*) is required to conclude the function g exists. The situation is different in the higher-order case. As we shall see when we consider higher-order models, we would need to interpret f not simply as a function from $\mathcal{D}_\iota$ to $\mathcal{D}_\iota$, but as a member of a domain $\mathcal{D}_{\iota\to\iota}$. Existence of an appropriate function from $\mathcal{D}_\iota$ to $\mathcal{D}_\iota$ follows from the axiom of choice at the meta-level, but the existence of an appropriate element of $\mathcal{D}_{\iota\to\iota}$ would only follow from a choice property internal to the higher-order model.

Dale Miller has shown that a naive adaptation of standard first-order Skolemization to higher-order logic allows one to prove particular instances of the axiom of choice.

For example, naive Skolemization permits an easy proof of the following version of the axiom of choice:

Example 4. (Choice) $(\forall X \exists Y . rXY) \Rightarrow (\exists F \forall X . rX(FX))$

However, naive Skolemization does not provide a complete method for reasoning with choice. The following example is equivalent to the axiom of choice (essentially Axiom 11 in [15]) but is not provable using naive Skolemization.

Example 5. (Choice) $\exists E_{(\iota\to o)\to\iota} \forall P . (\exists Y . PY) \Rightarrow . P(EP)$

Thus standard first-order Skolemization is unsound in higher-order logic as it partly introduces choice into the proof system. Dale Miller has fixed the problem by adding further conditions (see [22]): any Skolem function symbol f^n with dependency arity n (the existentially bound variable to be eliminated by a new Skolem term headed by f is depending on n universial variables) may only occur in formulas $f^n\overline{\mathbf{A}^n}$, where none of the $\mathbf{A}^i$ contains a variable that is bound outside of the term $f^n\overline{\mathbf{A}^n}$.

4 Semantics for HOL

In [8] we have re-examined the semantics of classical higher-order logic with the purpose of clarifying the role of extensionality. For this we have defined eight classes of higher-order models with respect to various combinations of Boolean extensionality and three forms of functional extensionality. One can further refine these eight model classes by varying the logical constants in the signature Σ as in [14].

A model of $\mathcal{HOL}$ is given by four objects: a typed collection of nonempty sets $(\mathcal{D}_\alpha)_{\alpha\in\mathcal{T}}$, an application operator $@\colon \mathcal{D}_{\alpha\to\beta}\times\mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$, an evaluation function $\mathcal{E}$ for terms and a valuation function $v\colon \mathcal{D}_o \longrightarrow \{\mathtt{T},\mathtt{F}\}$. A pair $(\mathcal{D},@)$ is called a Σ-applicative structure (see [8](3.1)). If $\mathcal{E}$ is an evaluation function for $(\mathcal{D},@)$ (see [8](3.18)), then we call the triple $(\mathcal{D},@,\mathcal{E})$ a Σ-evaluation. If v satisfies appropriate properties, then we call the tuple $(\mathcal{D},@,\mathcal{E},v)$ a Σ-model (see [8](3.40 and 3.41)).

Given an applicative structure $(\mathcal{D},@)$, an assignment φ is a (typed) function from $\mathcal{V}$ to $\mathcal{D}$. An evaluation function $\mathcal{E}$ maps an assignment φ and a term $\mathbf{A}_\alpha \in \mathit{wff}_\alpha(\Sigma)$ to an element $\mathcal{E}_\varphi(\mathbf{A}) \in \mathcal{D}_\alpha$. Evaluation functions $\mathcal{E}$ are required to satisfy four properties given in [8](3.18)). If $\mathbf{A}$ is closed and $\mathcal{E}$ is an evaluation function, then $\mathcal{E}_\varphi(\mathbf{A})$ cannot depend on φ and we write $\mathcal{E}(\mathbf{A})$.

A valuation $v\colon \mathcal{D}_o \longrightarrow \{\mathtt{T},\mathtt{F}\}$ is required to satisfy a property $\mathfrak{L}_c(\mathcal{E}(c))$ for every logical constant $c \in \Sigma$ (see [8](3.40)). For each logical constant c, $\mathfrak{L}_c(\mathsf{a})$ is defined to hold if a is an object of a domain $\mathcal{D}_\alpha$ satisfying the characterizing property of the logical constant c. For example, $\mathfrak{L}_\neg(\mathsf{n})$ holds for $\mathsf{n} \in \mathcal{D}_{o\to o}$ iff for every $\mathsf{a} \in \mathcal{D}_o$, $v(\mathsf{n}@\mathsf{a})$ is $\mathtt{T}$ iff $v(\mathsf{a})$ is $\mathtt{F}$. Likewise, $\mathfrak{L}_{=^\alpha}(\mathsf{q})$ holds for $\mathsf{q} \in \mathcal{D}_{\alpha\to\alpha\to o}$ if for every $\mathsf{a},\mathsf{b} \in \mathcal{D}_\alpha$, $v(\mathsf{q}@\mathsf{a}@\mathsf{b})$ is $\mathtt{T}$ iff a equals b.

Given a model $\mathcal{M} := (\mathcal{D},@,\mathcal{E},v)$, an assignment φ and a proposition $\mathbf{A}$ (or set of propositions Φ), we say $\mathcal{M}$ satisfies $\mathbf{A}$ (or Φ) and write $\mathcal{M} \models_\varphi \mathbf{A}$ (or $\mathcal{M} \models_\varphi \Phi$) if $v(\mathcal{E}_\varphi(\mathbf{A})) \equiv \mathtt{T}$ (or $v(\mathcal{E}_\varphi(\mathbf{A})) \equiv \mathtt{T}$ for each $\mathbf{A} \in \Phi$). If $\mathbf{A}$ is closed (or every member of Φ is closed), then we simply write $\mathcal{M} \models \mathbf{A}$ (or $\mathcal{M} \models \Phi$) and say $\mathcal{M}$ is a model of $\mathbf{A}$ (or Φ). We also consider classes $\mathfrak{M}$ of Σ-models and say a proposition $\mathbf{A}$ is valid in $\mathfrak{M}$ if $\mathcal{M} \models_\varphi \mathbf{A}$ for every $\mathcal{M} \in \mathfrak{M}$ and assignment φ.

In order to define model classes which correspond to different notions of extensionality, we define five properties of models (see [8](3.46, 3.21 and 3.5)). For each Σ-model $\mathcal{M} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$, we say $\mathcal{M}$ satisfies property

$\mathfrak{q}$ iff for all $\alpha \in \mathcal{T}$ there is a $\mathsf{q}^\alpha \in \mathcal{D}_{\alpha\to\alpha\to o}$ with $\mathfrak{L}_{=^\alpha}(\mathsf{q}^\alpha)$.
η iff $(\mathcal{D}, @, \mathcal{E})$ is η-functional (i.e., for each $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and assignment φ, $\mathcal{E}_\varphi(\mathbf{A}) \equiv \mathcal{E}_\varphi(\mathbf{A}\downarrow_{\beta\eta})$).
ξ iff $(\mathcal{D}, @, \mathcal{E})$ is ξ-functional (i.e., for each $\mathbf{M}, \mathbf{N} \in \mathit{wff}_\beta(\Sigma)$, $X \in \mathcal{V}_\alpha$ and assignment φ, $\mathcal{E}_\varphi(\lambda X_\alpha . \mathbf{M}_\beta) \equiv \mathcal{E}_\varphi(\lambda X_\alpha . \mathbf{N}_\beta)$ whenever $\mathcal{E}_{\varphi,[\mathsf{a}/X]}(\mathbf{M}) \equiv \mathcal{E}_{\varphi,[\mathsf{a}/X]}(\mathbf{N})$ for every $\mathsf{a} \in \mathcal{D}_\alpha$).
$\mathfrak{f}$ iff $(\mathcal{D}, @)$ is functional (i.e., for each $\mathsf{f}, \mathsf{g} \in \mathcal{D}_{\alpha\to\beta}$, $\mathsf{f} \equiv \mathsf{g}$ whenever $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for every $\mathsf{a} \in \mathcal{D}_\alpha$).
$\mathfrak{b}$ iff υ is injective.

For each $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ and each signature Σ we define $\mathfrak{M}_*(\Sigma)$ to be the class of all Σ-models $\mathcal{M}$ such that $\mathcal{M}$ satisfies property $\mathfrak{q}$ and each of the additional properties $\{\eta, \xi, \mathfrak{f}, \mathfrak{b}\}$ indicated in the subscript $*$ (see [8](3.49)). We always include β in the subscript to indicate that β-equal terms are always interpreted as identical elements. We do not include property $\mathfrak{q}$ as an explicit subscript; $\mathfrak{q}$ is treated as a basic, implicit requirement for all model classes. See [8](3.52) for a discussion on why we require property $\mathfrak{q}$. (We also briefly explore models which do not satisfy property $\mathfrak{q}$ in the context of Example 8 and again in Subsection 5.3.) Since we are varying four properties, one would expect to obtain 16 model classes. However, we showed in [8] that $\mathfrak{f}$ is equivalent to the conjunction of ξ and η. Note that, for example, $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$ is a larger class of models than $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$, hence fewer propositions are valid in $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$ than are valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$. In our examples we try to indicate the largest of our model classes in which the proposition is valid. Implicitly, this means the proposition is also valid in smaller (more restricted) model classes and may not be valid in larger (less restricted) ones.

5 Test Problems for Higher-Order Theories

Unless stated otherwise, we assume the signature includes $\overline{\Sigma}$ (see p. 67) and write $\mathfrak{M}_*$ for $\mathfrak{M}_*(\Sigma)$. Many of the examples could be considered in the context of smaller signatures. In the following discussion, we only consider smaller signatures in order to make particular points. (Note that if the signature becomes too small, Leibniz equality, for example, is no longer expressible.)

5.1 Properties of Equality

There are many useful first-order test problems on equality reasoning in the literature. For instance, in [12] the following clause set is given to illustrate the incompleteness of the RUE-NRF resolution approach as introduced in [16]:

$$\{g(f(a)) = a, f(g(X)) \neq X\}$$

Here, X is a free variable (i.e., implicitly universially quantified) and f, g are unary function symbols. In [12] it is shown that this inconsistent clause set cannot be refuted in the first-order RUE-NRF approach.

We now present some higher-order test problems addressing properties of equality. Some of them apply to many possible notions of equality while others describe specific properties of individual notions or relate different notions to each other.

Example 6. Equality is an equivalence relation in $\mathfrak{M}_\beta$. These particular examples should be theorems even if one replaces $\stackrel{*}{=}$ with an extensional definition of equality (e.g., $\Leftrightarrow$ at type o or $\stackrel{set}{=}$ at any type $\alpha \to o$).

(a) $\forall X_\alpha . X \stackrel{*}{=} X$
(b) $\forall X_\alpha \forall Y_\alpha . X \stackrel{*}{=} Y \Rightarrow Y \stackrel{*}{=} X$
(c) $\forall X_\alpha \forall Y_\alpha \forall Z_\alpha . (X \stackrel{*}{=} Y \wedge Y \stackrel{*}{=} Z) \Rightarrow X \stackrel{*}{=} Z$

Example 7. Equality obeys the congruence property (substitutivity property) in $\mathfrak{M}_\beta$.

(a) $\forall X_\alpha \forall Y_\alpha \forall F_{\alpha\to\alpha} . X \stackrel{*}{=} Y \Rightarrow (FX) \stackrel{*}{=} (FY)$
(b) $\forall X_\alpha \forall Y_\alpha \forall P_{\alpha\to o} . (X \stackrel{*}{=} Y) \wedge (PX) \Rightarrow (PY)$

Example 8 relates the Leibniz definition of equality to primitive equality.

Example 8. $(a_\alpha \doteq^\alpha b_\alpha) \Rightarrow (a =^\alpha b)$.

One could legitimately debate whether Example 8 should be a theorem. On the one hand, if Example 8 is not a theorem, then one should not consider Leibniz equality to be a definition of *real* equality. Semantically, Henkin's first (quite natural) definitions of models allowed models in which Leibniz equality (e.g., at type ι) does *not* evaluate to equality of objects in the model. Such a model $\mathcal{M}$ is constructed in [1]. This model $\mathcal{M}$ is a Σ-model in the sense of this paper (if one assumes $=^\alpha \notin \Sigma$ for every type α), but is not in any model class $\mathfrak{M}_*(\Sigma)$ since property $\mathfrak{q}$ fails. There is a slight technical problem with saying $\mathcal{M}$ provides a counter-model for Example 8 since one cannot express Example 8 without $=^\iota \in \Sigma$. As in [14], one can distinguish between *internal* and *external* uses of equality (as well as $\Rightarrow$ and $\forall$) and determine that $\mathcal{M}$ *is* (in a sense that can be made precise) a countermodel for Example 8.

If a model satisfies property $\mathfrak{q}$, then Example 8 is valid for any type α. If a logical system is intended to be complete for one of our model classes $\mathfrak{M}_*(\Sigma)$, then Example 8 should be a theorem. For the complete natural deduction calculi in [8], there is an explicit rule which derives primitive equality from Leibniz equality. In some sense, requiring property $\mathfrak{q}$ semantically corresponds to explicitly requiring that Example 8 be provable.

Also, if $=^\alpha \in \Sigma$, then Example 8 (for this particular type α) is valid in any Σ-model. A proof *using* primitive equality could instantiate the Leibniz variable $P_{\alpha\to o}$ with $(\lambda Z_\alpha . a = Z)$. The important point is that $=$ must be available for instantiations during proofs (not simply for expressing the original sentence).

Extensionality is the distinguishing property motivating our different model classes. For both, functional and Boolean extensionality, we distinguish between a trivial and a non-trivial direction.

Example 9. The trivial directions of functional and Boolean extensionality are valid in $\mathfrak{M}_\beta$.

(a) $\forall F_{\alpha\to\beta}\forall G_{\alpha\to\beta}.F \stackrel{*}{=} G \Rightarrow (\forall X_\alpha.(FX) \stackrel{*}{=} (GX))$
(b) $\forall A_o\forall B_o.A \stackrel{*}{=} B \Rightarrow (A \Leftrightarrow B)$

The other directions are not valid in $\mathfrak{M}_\beta$. They become theorems only relative to more restricted model classes in our landscape.

Example 10. (discussed in [15]; Axiom 10 in [17]) $\forall A_o\forall B_o.(A \Leftrightarrow B) \Rightarrow A \stackrel{*}{=} B$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$. This is the non-trivial direction of Boolean extensionality.

Example 11. ([15,17], Axiom $10^{\beta\alpha}$) $\forall F_{\alpha\to\beta}\forall G_{\alpha\to\beta}.(\forall X_\alpha.(FX) \stackrel{*}{=} (GX)) \Rightarrow F \stackrel{*}{=} G$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}}$. This is the non-trivial direction of functional extensionality. (Property $\mathfrak{q}$ is also relevant to this example as is discussed in [8].)

5.2 Extensionality

We next present examples that illustrate distinguishing properties of the different model classes with respect to extensionality. In the preceding sections we have already mentioned several test problems that are independent of the "amount of extensionality" and which are theorems in $\mathfrak{M}_\beta$. We additionally refer to all first-order test problems as, for instance, provided in the TPTP library.

η-equality is usually realized as part of the pre-unification algorithm in a higher-order reasoning system. It is important to note that η-equality should not be confused with full extensionality. In literature on higher-order rewriting, for instance [25], the notion of extensionality is usually only associated with η-conversion which is far less than full extensionality.

Example 12. $(p_{(\iota\to\iota)\to o}(\lambda X_\iota.f_{\iota\to\iota}X)) \Rightarrow (p_{(\iota\to\iota)\to o}f)$ is essentially 21 from [15] which expresses η-equality using Leibniz equality. It is valid in $\mathfrak{M}_{\beta\eta}$ but not in $\mathfrak{M}_\beta$.

Property ξ together with η gives us full functional extensionality.

Example 13. Validity of $(\forall X_\iota.(f_{\iota\to\iota}X) \stackrel{*}{=} X) \wedge p(\lambda X_\iota X) \Rightarrow p(\lambda X_\iota.fX)$ only depends on ξ, not on η. It is thus valid in $\mathfrak{M}_{\beta\xi}$ (but not in model classes which do not require either ξ or $\mathfrak{f}$).

Example 14. $(\forall X_\iota.(f_{\iota\to\iota}X) \stackrel{*}{=} X) \wedge p(\lambda X_\iota X) \Rightarrow pf$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}}$, but not in model classes which do not require $\mathfrak{f}$.

As in Example 11, property $\mathfrak{q}$ is important for validity of Example 13 in $\mathfrak{M}_{\beta\xi}$ and validity of Example 14 in $\mathfrak{M}_{\beta\mathfrak{f}}$.

Example 15. ([7]) (a) $p_{o\to o}\ (a_o \wedge b_o) \Rightarrow p\ (b \wedge a)$ and (b) $a_o \wedge b_o \wedge (p_{o\to o}a) \Rightarrow (pb)$ are valid iff we require Boolean extensionality as in $\mathfrak{M}_{\beta\mathfrak{b}}$.

Example 16. $(p_{o\to o}\ a_o) \wedge (p\ b_o) \Rightarrow (p\ (a \wedge b))$ is a theorem of $\mathfrak{M}_{\beta\mathfrak{b}}$ which is slightly more complicated to mechanize in some calculi; see [7] for more details.

Example 17. $\neg(a = \neg a)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$. As discussed in [7] this example motivates specific inference rules for the mechanization of primitive equality.

The following is a tricky example introduced in [14].

Example 18. $(h_{o\to\iota}((h\top) \stackrel{*}{=} (h\bot))) \stackrel{*}{=} (h\bot)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$, but not in model classes which do not require property $\mathfrak{b}$.

Many people do not immediately accept that Example 18 is a theorem. A simple informal argument is helpful. Either $(h\top) \stackrel{*}{=} (h\bot)$ is true or false. If the equation holds, then Example 18 reduces to $(h\top) \stackrel{*}{=} (h\bot)$ which we have just assumed. If the equation is false, then Example 18 reduces to $(h\bot) \stackrel{*}{=} (h\bot)$, an instance of reflexivity.

Example 19 combines Boolean extensionality with η-equality.

Example 19. $p_{(\iota\to\iota)\to o}(\lambda X_\iota.f_{o\to\iota\to\iota}(a_{(\iota\to\iota)\to o}(\lambda X_\iota.fb_oX)\wedge b)X) \Rightarrow p(f(b\wedge.a(fb)))$ is valid in $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, but is not valid if properties $\mathfrak{b}$ and η are not assumed.

By DeMorgan's Law, we know $X \wedge Y$ is the same as $\neg(\neg X \vee \neg Y)$. In Example 20, we vary the notion of "is the same as" to obtain several examples which are only provable with some amount of extensionality. Note that if we only assume property ξ, we can only conclude the η-expanded form of $\wedge$ is equal to $(\lambda X\lambda Y.\neg(\neg X \vee \neg Y))$.

Example 20. Consider the following examples.

(a) $\forall X\forall Y.X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$ is valid in $\mathfrak{M}_\beta$.
(b) $\forall X\forall Y.X \wedge Y \stackrel{*}{=} \neg(\neg X \vee \neg Y)$ is valid in $\mathfrak{M}_{\beta\mathfrak{b}}$.
(c) $(\lambda U\lambda V.U \wedge V) \stackrel{*}{=} (\lambda X\lambda Y.\neg(\neg X \vee \neg Y))$ is valid in $\mathfrak{M}_{\beta\xi\mathfrak{b}}$.
(d) $\wedge \stackrel{*}{=} (\lambda X\lambda Y.\neg(\neg X \vee \neg Y))$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$.

Finally we reach Henkin semantics which is characterized by full extensionality, i.e. the combination of Boolean and functional extensionality. Example 20(d) already provided one example valid only in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$.

Example 21. The following theorem in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ characterizes the fact that in all Henkin models we have exactly four functions mapping truth values to truth values.

$$((p\ \lambda X_o.X_o) \wedge (p\ \lambda X_o.\neg X_o) \wedge (p\ \lambda X_o.\bot) \wedge (p\ \lambda X_o.\top)) \Rightarrow \forall Y_{o\to o}.(p\ Y)$$

Example 22. As exploited in [11], set theory problems can be concisely and elegantly formulated in higher-order logic when using λ-abstraction to encode sets as characteristic functions. For instance, given a predicate $p_{\alpha\to o}$ the set of all objects of type α that have property p is denoted as $\lambda X_\alpha.(pX)$. We then define set operations as follows (we give only some examples):

set operation	defined by
$\in_{\alpha\to(\alpha\to o)\to o}$	$\lambda Z_\alpha\lambda X_{\alpha\to o}(XZ)$
$\{.\}_{\alpha\to(\alpha\to o)}$	$\lambda U_\alpha(\lambda Z_\alpha.Z \stackrel{*}{=} U)$
$\emptyset_{\alpha\to o}$	$(\lambda Z_\alpha\bot)$
$\cap_{(\alpha\to o)\to(\alpha\to o)\to(\alpha\to o)}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\lambda Z_\alpha.Z \in X \wedge Y \in Y)$
$\cup_{(\alpha\to o)\to(\alpha\to o)\to(\alpha\to o)}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\lambda Z_\alpha.Z \in X \vee Y \in Y)$
$\subseteq_{(\alpha\to o)\to(\alpha\to o)\to o}$	$\lambda X_{\alpha\to o}\lambda Y_{\alpha\to o}(\forall Z_\alpha.Z \in X \Rightarrow Y \in Y)$
$\wp_{(\alpha\to o)\to((\alpha\to o)\to o)}$	$\lambda X_{\alpha\to o}(\lambda Y_{\alpha\to o}.Y \subseteq X)$

We can now formulate some test problems on sets:

(a) $a_{\alpha\to o} \cup (b_{\alpha\to o} \cap c_{\alpha\to o}) \stackrel{set}{=} (a \cup b) \cap (a \cup c)$ is valid in $\mathfrak{M}_\beta$.
(b) $a_{\alpha\to o} \cup (b_{\alpha\to o} \cap c_{\alpha\to o}) \stackrel{*}{=} (a \cup b) \cap (a \cup c)$ is valid in $\mathfrak{M}_{\beta\xi\mathfrak{b}}$ but not in model classes without ξ and $\mathfrak{b}$.
(c) $\wp(\emptyset_{\alpha\to o}) \stackrel{set}{=} \{\emptyset_{\alpha\to o}\}$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ but not in model classes without $\mathfrak{f}$ and $\mathfrak{b}$. The example is not valid in $\mathfrak{M}_\beta$ due to the embedded equation introduced by the definition of a singleton set $\{.\}$.
(d) and $\wp(\emptyset_{\alpha\to o}) \stackrel{*}{=} \{\emptyset_{\alpha\to o}\}$ is valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ but not in model classes without $\mathfrak{f}$ and $\mathfrak{b}$.

These examples motivate pre-processing in higher-order theorem proving in which the definitions are fully expanded and in which the extensionality principles are employed es early as possible. After pre-processing, many problems of this kind can be automatically translated from their concise and human readable higher-order representation into first-order or even propositional logic representations to be easily checked by respective specialist systems.

5.3 Set Comprehension

One of the advantages of Church's type theory is that instead of assuming comprehension axioms one can simply use terms defining sets for set instantiations. Such set instantiations make use of logical constants in the signature Σ. As in [14] one can vary the signature of logical constants in order to vary the set comprehension assumed in Σ-models. With different amounts of set comprehension, different examples will be valid.

Generating set instantiations is one of the toughest challenges for the automation of higher-order logic. (In fact set instantiations can be employed to simulate the cut-rule as soon as one of the following prominent axioms of higher-order logic is available in the search space: comprehension, induction, extensionality, choice, description.) Set instantiations are often generated during automated search using an enumeration technique involving primitive substitutions.

For each example below, we note restrictions on the signature Σ under which the example is either valid or not valid. Since we would like to distinguish between signatures which contain primitive equality (at various types) and those which do not, we consider classes of models which do not necessarily satisfy property $\mathfrak{q}$. In particular, let $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$ be the set of all Σ-models and let $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}^{-\mathfrak{q}}(\Sigma)$ be the set of all Σ-models satisfying properties $\mathfrak{f}$ and $\mathfrak{b}$ (without requiring property $\mathfrak{q}$).

As in Example 8 one can focus on the use of logical constants in Σ for instantiations and ignore certain uses of logical constants to express the formula. For example, suppose $\mathbf{A} \in \mathit{cwff}_o(\Sigma)$, $\mathcal{M}$ is a Σ-model and $\neg \notin \Sigma$. While $(\neg\mathbf{A}) \notin \mathit{wff}_o(\Sigma)$, we can consider $(\neg\mathbf{A})$ to be a *Σ-external proposition* and define $\mathcal{M} \models \neg\mathbf{A}$ to mean $\mathcal{M} \not\models \mathbf{A}$. Intuitively, the negation is used *externally* in $(\neg\mathbf{A})$. We can inductively define the set of Σ-external propositions $\mathbf{M}$ and the meaning of $\mathcal{M} \models \mathbf{M}$ for Σ-models $\mathcal{M}$. After doing so, most of the examples below are Σ-external propositions even if Σ contains no logical constants. Only Examples 30 and 33 in this section make nontrivial uses of certain logical constants to express the propositions. Due to space considerations, we refer the reader to [14] for details.

If Σ is sufficiently small, then one can construct two trivial models in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ where $\mathcal{D}_o$ is either simply $\{\mathtt{T}\}$ or $\{\mathtt{F}\}$. (This possibility was ruled out in [8] since we assumed $\neg \in \Sigma$.)

Example 23. $\exists PP$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if either $\top \in \Sigma$ or $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ if $\Sigma \subseteq \{\bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$. (Any proof must use a set instantiation involving either $\top$, $\neg$, $\Rightarrow$, $\Leftrightarrow$ or some primitive equality.)

Example 24. $\neg\forall PP$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if either $\bot \in \Sigma$ or $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ if $\Sigma \subseteq (\overline{\Sigma} \setminus \{\bot, \neg\})$. (Any proof must use a set instantiation involving either $\bot$ or $\neg$.)

Example 25 characterizes when an instantiation satisfying the property of negation is possible. This can be either because the signature supplies negation or supplies enough constants to define negation.

Example 25. $\exists N_{o\to o}\forall P_o.NP \Leftrightarrow \neg P$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if $\neg \in \Sigma$. The example is also valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if $\bot \in \Sigma$ and $\{\Rightarrow, \Leftrightarrow\} \cap \Sigma \neq \emptyset$ since one can consider either the term $\lambda X_o.X \Rightarrow \bot$ or the term $\lambda X_o.X \Leftrightarrow \bot$. The example is not valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$.

One possibility we did not cover in Example 25 is if Σ is $\{\bot, =^o\}$. Consider the term $(\lambda X_o.X =^o \bot)$. This only defines negation if we assume Boolean extensionality. Hence we obtain the interesting fact that Example 25 is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\{\bot, =^o\})$, but is *not* valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\{\bot, =^o\})$.

One can modify Example 25 in a way that requires not only a set instantiation for negation, but also extensionality.

Example 26. $\neg\forall F_{o\to o}\exists X.(FX) \stackrel{*}{=} X$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ if $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ regardless of the signature Σ. Also, the example is not valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta\mathfrak{fb}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$.

Example 27 characterizes when an instantiation can essentially define disjunction and Example 28 characterizes when an instantiation can essentially define the universal quantifier at type α. Clearly one can modify these examples for any other logical constant.

Example 27. $\exists D_{o\to o\to o}\forall P_o\forall Q_o.DPQ \Leftrightarrow (P \vee Q)$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if $\vee \in \Sigma$. The example is also valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if $\{\neg, \wedge\} \subseteq \Sigma$.

Example 28. $\exists Q_{(\alpha\to o)\to o)}\forall P_{\alpha\to o}.QP \Leftrightarrow \forall X_\alpha.PX$ is valid in $\mathfrak{M}^{-\mathfrak{q}}_{\beta}(\Sigma)$ if $\Pi^\alpha \in \Sigma$.

Recall that Example 8 already provided an example in which one *might* require a set instantiation involving primitive equality (depending on how the calculus relates Leibniz equality to primitive equality).

A few interesting set instantiations involve no logical constants, but do make use of projections (see [18]). Sometimes such projections can be obtained from higher-order unification, as in Example 29.

Example 29. $\exists N_{o\to o}\forall P_o.NP \Leftrightarrow P$ is valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\emptyset)$.

However, one cannot expect higher-order unification to *always* provide projection terms when they are needed. Example 30 was studied extensively in [2] (see THM104) in order to demonstrate this fact. In this example, we make use of the abbreviation $\{.\}$ which was defined in Example 22. If the definition of $\{.\}$ makes use of primitive equality, one must assume $=^\iota \in \Sigma$ to express the proposition. If $\{.\}$ is defined using Leibniz equality, then one must assume $\neg, \Pi^{\iota\to o} \in \Sigma$ to express the proposition.

Example 30. $\forall X_\iota \forall Z_\iota.\{X\}\doteq\{Z\} \Rightarrow X\doteq Z$ is valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$ so long as Σ is sufficient to express the proposition.

The examples above are straightforward examples designed to ensure completeness of theorem provers with respect to set comprehension. A more natural theorem which requires set instantiations is Cantor's Theorem. Two forms of Cantor's Theorem were studied with respect to set comprehension in [14]. Example 31 is the surjective form of Cantor's Theorem discussed in [4].

Example 31. (Surjective Cantor Theorem) $\neg\exists G_{\alpha\to\alpha\to o}\forall F_{\alpha\to o}\exists J_\alpha.GJ =^{\alpha\to o} F$ is valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\neg \in \Sigma$. The example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \wedge, \vee\} \cup \{\Pi^\alpha, \Sigma^\alpha | \alpha \in \mathcal{T}\}$ (see Theorem 6.7.8 in [14]).

An alternative formulation of Cantor's Theorem (see [5,14]) is the injective form shown in Example 32. Almost any higher-order theorem prover complete for the corresponding model class should be capable of proving the previous examples in this subsection. Example 32 is far more challenging. At the present time, no theorem prover has found a proof of Example 32 automatically.

Example 32. (Injective Cantor Theorem) $\neg\exists H_{(\iota\to o)\to\iota}\forall P_{\iota\to o}\forall Q_{\iota\to o}.HP =^\iota HQ \Rightarrow P =^{\iota\to o} Q$ is valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\{\neg, \wedge, =^\iota, \Pi^{\iota\to o}\} \subseteq \Sigma$ (see Lemma 6.7.2 in [14]). The example is not valid in $\mathfrak{M}_{\beta\mathfrak{fb}}^{-\mathfrak{q}}(\Sigma)$ if $\Sigma \subseteq \{\top, \bot, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \Pi^\iota, \Sigma^\iota, =^{\iota\to o}\}$. (This fact follows from the results in Section 6.7 of [14].)

One of the difficulties of proving Example 32 is that certain set instantiations seem to be needed beneath other set instantiations (see [5]). The next family of examples illustrates that nontrivial set instantiations can occur within set instantiations with an arbitrary number of iterations.

Example 33. Assume Σ contains $\neg$ and Π^α for every type α. Fix a constant c_ι. We will define a theorem $\mathbf{D}_o^n$ for each natural number n. By induction on n, define simple types τ^n and abbreviations $\mathbf{A}^n_{\tau^n\to o}$ as follows.

(a) Let τ^0 be the type ι and τ^{n+1} be $\tau^n \to o$ for each natural number n.
(b) Let $\mathbf{A}^0_{\iota\to o}$ be $\lambda Z_\iota.(Z\doteq c_\iota) \wedge \top$ and $\mathbf{A}^{n+1}$ be $\lambda Z_{\tau^{n+1}}.(Z\doteq\mathbf{A}^n) \wedge \exists T_{\tau^n}.ZT$ for each natural number n.

Finally, for each n, let $\mathbf{D}_o^n$ be $\exists S_{\tau^n} A^n S$. Each $\mathbf{D}^n$ is a valid in $\mathfrak{M}_\beta^{-\mathfrak{q}}(\Sigma)$. The constant c_ι is the obvious witness for $\mathbf{D}^0$. For each n, $\mathbf{A}^n$ is the witness for $\mathbf{D}^{n+1}$. Note that a subgoal of showing $\mathbf{A}^n$ is the witness for $\mathbf{D}^{n+1}$ involves showing $\mathbf{A}^n$ is nonempty (which was $\mathbf{D}^n$). Hence this proof of $\mathbf{D}^{n+1}$ involves all the previous instantiations $\mathbf{A}^0, \ldots, \mathbf{A}^n$.

6 More Complex Examples

Here we present technically or proof theoretically challenging examples. First we consider a class of hard problems simply involving β-reduction.

Example 34. Let α^0 be ι and α^{n+1} be $(\alpha^n \to \alpha^n)$ for each n. Note that the Church numeral $\overline{2}^{\alpha^n}$ has type α^{n+2}. For any n we can form the term $(\overline{2}^{\alpha^n}\overline{2}^{\alpha^{n-1}}\cdots\overline{2}^{\alpha^0})$ of type $(\iota \to \iota) \to \iota \to \iota$. The size of the β-normal form of this term is approximately of size $2^{(2^{\cdots}2)}$ containing $n+1$ '2s'. (This is a well-known example, mentioned in [27].) For $n \geq 4$ it becomes infeasible to β-normalize such a term (since $2^{2^{2^2}}$ is 2^{65536}, a number much larger than google). One can express relatively simple theorems using this term such as

$$(\overline{2}^{\alpha^n}\overline{2}^{\alpha^{n-1}}\cdots\overline{2}^{\alpha^0})(\lambda X_\iota X) \stackrel{*}{=} (\lambda X_\iota X).$$

If one avoids eager β-normalization and allows lemmas, then there is a reasonably short proof using higher-order logic. We first define the set C_2^α of Church numerals (over α) greater than or equal to 2:

$$\lambda N_{(\alpha\to\alpha)\to\alpha\to\alpha}\forall P.(P\overline{2}^\alpha \wedge (\forall M.PM \Rightarrow P(\overline{s}M))) \Rightarrow PN.$$

(Technically, $(\overline{0}\ \overline{2})$ is β-equal to $(\lambda F_{\iota\to\iota}F)$, which is not equal to $\overline{1}$. We work with the set of Church numerals greater than or equal to 2 to avoid this problem.) One can prove two results with little trouble (where the lengths of the proofs do not depend on the type α):

(a) $\forall N_{((\alpha\to\alpha)\to\alpha\to\alpha)\to(\alpha\to\alpha)\to\alpha\to\alpha}.C_2^{\alpha\to\alpha}N \Rightarrow C_2^\alpha(N\overline{2}^\alpha)$
(b) $\forall N_{(\alpha\to\alpha)\to\alpha\to\alpha}.C_2^\alpha N \Rightarrow .(N(\lambda X_\alpha X)) = (\lambda X_\alpha X)$

Using (a) at several types and (b) at type ι, we can prove, e.g.,

$$(\overline{2}^{\alpha^4}\overline{2}^{\alpha^3}\overline{2}^{\alpha^2}\overline{2}^{\alpha^1}\overline{2}^{\alpha^0})(\lambda X_\iota X) \stackrel{*}{=} (\lambda X_\iota X)$$

in higher-order logic without β-normalizing.

In [13, Chapter 25, p. 376–382] Boolos presents a related example of a first-order problem which has only a very long (practically infeasible) derivation in first-order logic, but which has a short derivation in a second-order logic, by making use of comprehension axioms.

Example 35. (Boolos' Curious Inference)

$$\begin{aligned}&(\forall n.f(n,1) = s(1) \wedge \forall x.f(1,s(x)) = s(s(f(1,x)))\\ &\quad\wedge \forall n.\forall x.f(s(n),s(x)) = f(n,f(s(n),x))\\ &\quad\wedge D(1) \wedge \forall x.(D(x) \Rightarrow D(s(x))))\\ &\Rightarrow D(f(s(s(s(1)))),s(s(s(s(1))))))\end{aligned}$$

If there were an appropriate (first-order) induction principle available, then there should be a short proof of this example. Note that the example specifies f to be the Ackermann function which grows extremely fast and hence $f(s(s(s(1)))),s(s(s(1)))))$

is a very big number. Actually, there is long first-order proof which is relatively easy to describe. Boolos argues that any first-order proof must be of size at least $2^{(2^{\cdots 2})}$ containing 64K '2s' in all (far more enormous than the number 2^{64K} in Example 34). There is no chance of formally representing such a proof with all computation power ever. Boolos presents a short alternative proof in second-order logic that makes use of higher-order lemmas obtained from comprehension axioms. Formulating the appropriate lemmas (as with the lemmas in Example 34) requires human ingenuity that goes beyond the capabilities of what can be supported with primitive substitution and lemma speculation techniques in current theorem proving approaches.

As discussed in [3], there is a family of theorems $\mathbf{A}^1, \mathbf{A}^2, \ldots$ which are all of the same low order such that $\mathbf{A}^n$ is not provable unless one uses set instantiations involving n^{th}-order quantifiers. To obtain concrete examples from the argument, one must use Gödel numbering. A family of simpler examples displaying this phenomenon would likely be enlightening.

7 Conclusion

We have presented a first set of higher-order test examples that may support the development of higher-order proof systems. This set of examples has been structured according to technical aspects and the semantic properties of extensionality and set comprehension. Future work is to add examples and include them in either the TPTP library or an appropriate higher-order variant. Many more examples are particularly needed to illustrate properties of different forms of equality.

References

1. P. B. Andrews. General models and extensionality. *J. of Symbolic Logic*, 37(2):395–397, 1972.
2. P. B. Andrews. On Connections and Higher Order Logic. *J. of Automated Reasoning*, 5:257–291, 1989.
3. P. B. Andrews. Classical type theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 15, pages 965–1007. Elsevier Science, 2001.
4. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.
5. P. B. Andrews, M. Bishop, and C. E. Brown. TPS: A theorem proving system for type theory. In D. McAllester, editor, *Proc. of CADE-17*, number 1831 in LNAI, pages 164–169, Pittsburgh, USA, 2000. Springer.
6. Peter B. Andrews. Resolution in type theory. *J. of Symbolic Logic*, 36(3):414–432, 1971.
7. C. Benzmüller. *Equality and Extensionality in Automated Higher-Order Theorem Proving*. PhD thesis, Saarland University, 1999.
8. C. Benzmüller, C. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *J. of Symbolic Logic*, 69(4):1027–1088, 2004.
9. C. Benzmüller, C. E. Brown, and M. Kohlhase. Semantic techniques for higher-order cut-elimination. SEKI Technical Report SR-2004-07, Saarland University, Saarbrücken, Germany, 2004. Available at: `http://www.ags.uni-sb.de/~chris/papers/R37.pdf`.

10. C. Benzmüller and M. Kohlhase. LEO – a higher order theorem prover. In C. Kirchner and H. Kirchner, editors, *Proc. of CADE-15*, number 1421 in LNAI, pages 139–144, Lindau, Germany, 1998. Springer.
11. C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a higher-order and a first-order theorem prover cooperate? In F. Baader and A. Voronkov, editors, *Proc. of LPAR 2004*, volume 3452 of *LNAI*, pages 415–431. Springer, 2005.
12. M. P. Bonacina and J. Hsiang. Incompleteness of the RUE/NRF inference systems. Newsletter of the Association for Automated Reasoning, No. 20, pages 9–12, 1992.
13. G. Boolos. *Logic, Logic, Logic*. Harvard University Press, 1998.
14. C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 2004.
15. A. Church. A formulation of the simple theory of types. *J. of Symbolic Logic*, 5:56–68, 1940.
16. V. J. Digricoli. Resolution by unification and equality. In W. H. Joyner, editor, *Proc. of CADE-4*, Austin, Texas, USA, 1979.
17. Leon Henkin. Completeness in the theory of types. *J. of Symbolic Logic*, 15(2):81–91, 1950.
18. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
19. M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Saarland University, 1994.
20. M. Kohlhase. Higher-order tableaux. In *Proc. of TABLEAUX 95*, number 918 in LNAI, pages 294–309. Springer, 1995.
21. J.D. McCharen, R.A. Overbeek, and L.A. Wos. Problems and Experiments for and with Automated Theorem-Proving Programs. *IEEE Transactions on Computers*, C-25(8):773–782, 1976.
22. D. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon Univ., 1983.
23. F.J. Pelletier. Seventy-five Problems for Testing Automatic Theorem Provers. *J. of Automated Reasoning*, 2(2):191–216, 1986.
24. F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.
25. C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming*. Progress in Theoretical Computer Science. Birkhäuser, 1998.
26. W. Snyder and J. Gallier. Higher-Order Unification Revisited: Complete Sets of Transformations. *J. of Symbolic Computation*, 8:101–140, 1989.
27. R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.
28. G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *J. of Automated Reasoning*, 21(2):177–203, 1998.
29. G.A. Wilson and J. Minker. Resolution, Refinements, and Search Strategies: A Comparative Study. *IEEE Transactions on Computers*, C-25(8):782–801, 1976.
30. C.-P. Wirth. Descente infinie + Deduction. *Logic J. of the IGPL*, 12(1):1–96, 2004. `www.ags.uni-sb.de/~cp/p/d/welcome.html`.

THE JOURNAL OF SYMBOLIC LOGIC
Volume 69, Number 4, Dec. 2004

HIGHER-ORDER SEMANTICS AND EXTENSIONALITY

CHRISTOPH BENZMÜLLER, CHAD E. BROWN, AND MICHAEL KOHLHASE

Abstract. In this paper we re-examine the semantics of classical higher-order logic with the purpose of clarifying the role of extensionality. To reach this goal, we distinguish nine classes of higher-order models with respect to various combinations of Boolean extensionality and three forms of functional extensionality. Furthermore, we develop a methodology of abstract consistency methods (by providing the necessary model existence theorems) needed to analyze completeness of (machine-oriented) higher-order calculi with respect to these model classes.

§1. Motivation. In classical first-order predicate logic, it is rather simple 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 easily be verified using, for instance, the abstract consistency method (cf. the introductory textbooks [6, 22]). This well understood meta-theory has supported the development of calculi adapted to special applications—such as automated theorem proving (cf. [16, 47] for an overview).

In higher-order logics, the situation is rather different: the intuitive set-theoretic standard semantics cannot give a sensible notion of completeness, since it does not admit complete (recursively axiomatizable) calculi [24, 6]. There is a more general notion of semantics [26], the so-called Henkin models, that allows complete (recursively axiomatizable) calculi and therefore sets the standard for deductive power of calculi.

Peter Andrews' *Unifying Principle for Type Theory* [1] 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 $\mathfrak{T}_\beta$. A calculus $\mathscr{C}$ is called complete relative to a calculus $\mathfrak{T}_\beta$ iff (if and only if) $\mathscr{C}$ proves all theorems of $\mathfrak{T}_\beta$. Since $\mathfrak{T}_\beta$ is not 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. The differences between these notions of completeness can largely be analyzed in terms of availability of various extensionality principles, which can be expressed axiomatically in higher-order logic.

As a consequence of the limitations of Andrew's *Unifying Principle*, calculi for higher-order automated theorem proving [1, 32, 33, 34, 42, 36, 37] and the corresponding theorem proving systems such as TPS [7, 8], or earlier versions of the LEO [14] system are not complete with respect to Henkin models. Moreover, they

Received February 23, 1998; final version March 29, 2004.

0022-4812/04/6904-0004/$7.20

are not even sound with respect to $\mathfrak{T}_\beta$, since they (for the most part) employ η-conversion, which is not admissible in $\mathfrak{T}_\beta$. In other words, their deductive power lies somewhere between $\mathfrak{T}_\beta$ and Henkin models. Characterizing exactly where reveals important theoretical properties of these calculi that have direct consequences for the adequacy in various application domains (see the discussion in section 8.1). Unlike calculi without computational concerns, calculi for mechanized reasoning systems cannot be made complete by simply adding extensionality axioms, since the search spaces induced by their introduction grow prohibitively. Being able to compare and characterize the methods and computational devices used instead is a prerequisite for further development in this area.

In this situation, the aim of this article 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 need to establish:

(1) classes of models that adequately characterize the deductive power of existing theorem-proving calculi (providing semantics with respect to which they are sound and complete), and
(2) a methodology of abstract consistency methods (by providing for these model classes the necessary model existence theorems, which extend Andrews' Unifying Principle), so that the completeness analysis for higher-order calculi will become almost as simple as in first-order logic.

We fully achieve the first goal in this article, and take a large step towards the second. In the model existence theorems presented in this article, we have to assume a new condition called *saturation*, which limits their utility in completeness proofs for machine-oriented calculi. Fortunately, the saturation condition can be lifted by extensions of the methods presented in this article (see the discussion in the conclusion 8.2 and [12]).

Due to the inherent complexity of higher-order semantics we first give an informal exposition of the issues covered and the techniques applied. In Section 4, we will investigate the properties of the model classes introduced in Section 3 in more detail and corroborate them with example models in Section 5. We prove model existence theorems for the model classes in Section 6. Finally, in Section 7 we will apply the model existence theorems from Section 6 to the task of proving completeness of higher-order natural deduction calculi. Section 8 concludes the article with a discussion of related work, possible applications, and the saturation assumption we introduced for the model existence theorems.

The work reported in this article is based on [15] and significantly extends the material presented there.

§2. Informal exposition. Before we turn to the exposition of the semantics in Section 2.3, let us specify what we mean by "higher-order logic": any simply typed logical system that allows quantification over function and predicate variables. Technically, we will follow tradition and employ a logical system $\mathscr{HOL}$ based on the simply typed λ-calculus as introduced in [18]; this does not restrict the generality of the methods reported in this article, since the ideas can be carried over. A related logical system is discussed in detail in [6].

2.1. Simply typed λ-calculus. To formulate higher-order logic we start with a collection of types $\mathscr{T}$. We assume there are some basic types in $\mathscr{T}$ and that whenever $\alpha, \beta \in \mathscr{T}$, then the function type $(\alpha \to \beta)$ is in $\mathscr{T}$. Furthermore, we assume the types are generated freely, so that $(\alpha_1 \to \beta_1) \equiv (\alpha_2 \to \beta_2)$ implies $\alpha_1 \equiv \alpha_2$ and $\beta_1 \equiv \beta_2$.

$\mathscr{HOL}$-formulae (or *term*s) are built up from a set $\mathscr{V}$ of (typed) variables and a *signature* Σ (a set of typed constants) as *application*s and λ-*abstraction*s. We assume the set $\mathscr{V}_\alpha$ of variables of type α is countably infinite for each type α. The set $\mathrm{wff}_\alpha(\Sigma)$ of *well-formed formulae* consists of those formulae which have type α. The type of formula $\boldsymbol{A}_\alpha$ will be annotated as an index, if it is not clear from the context. We will denote variables with upper-case letters $(X_\alpha, Y, Z, X^1_\beta, X^2_\gamma, \dots)$, constants with lower-case letters $(c_\alpha, f_{\alpha\to\beta}, \dots)$ and well-formed formulae with upper-case bold letters $(\boldsymbol{A}_\alpha, \boldsymbol{B}, \boldsymbol{C}^1, \dots)$. Finally, we abbreviate multiple applications and abstractions in a kind of vector notation, so that $\boldsymbol{A}\overline{\boldsymbol{U}^k}$ denotes k-fold application (associating to the left), $\lambda\overline{X^k}.\boldsymbol{A}$ denotes k-fold λ-abstraction (associating to the right) and we use the square dot '.' as an abbreviation for a pair of brackets, where '.' stands for the left one with its partner as far to the right as is consistent with the bracketing already present in the formula. We may avoid full bracketing of formulas in the remainder if the bracketing structure is clear from the context.

We will use the terms like *free* and *bound* variables or *closed* formulae in their standard meaning and use $\mathrm{free}(\boldsymbol{A})$ for the set of free variables of a formula $\boldsymbol{A}$. In particular, alphabetic change of names of bound variables is built into $\mathscr{HOL}$: we consider alphabetic variants to be identical (viewing the actual representation as a representative of an alphabetic equivalence class) and use a notion of substitution that avoids variable capture by systematically renaming bound variables.[1] We denote a substitution that instantiates a free variable X with a formula $\boldsymbol{A}$ with $[\boldsymbol{A}/X]$ and write $\sigma, [\boldsymbol{A}/X]$ for the substitution that is identical with σ but instantiates X with $\boldsymbol{A}$. For any term $\boldsymbol{A}$ we denote by $\boldsymbol{A}[\boldsymbol{B}]_p$ the term resulting by replacing the subterm at position p in $\boldsymbol{A}$ by $\boldsymbol{B}$.

A structural equality relation of $\mathscr{HOL}$ terms is induced by $\beta\eta$-*reduction*

$$(\lambda X.\boldsymbol{A})\boldsymbol{B} \to_\beta [\boldsymbol{B}/X]\boldsymbol{A} \qquad\qquad (\lambda X.\boldsymbol{C}X) \to_\eta \boldsymbol{C}$$

where X is not free in $\boldsymbol{C}$. It is well-known that the reduction relations β, η, and $\beta\eta$ are terminating and confluent on $\mathrm{wff}(\Sigma)$, so that there are unique normal forms (cf. [9] for an introduction). We will denote the β-normal form of a term $\boldsymbol{A}$ by $\boldsymbol{A}\downarrow_\beta$, and the $\beta\eta$-normal form of $\boldsymbol{A}$ by $\boldsymbol{A}\downarrow_{\beta\eta}$. If we allow both reduction and expansion steps, we obtain notions of β-*conversion*, η-*conversion*, and $\beta\eta$-*conversion*. We say $\boldsymbol{A}$ and $\boldsymbol{B}$ are β-*equal* [η-*equal*, $\beta\eta$-*equal*] (written $\boldsymbol{A}\equiv_\beta\boldsymbol{B}$ [$\boldsymbol{A}\equiv_\eta\boldsymbol{B}$, $\boldsymbol{A}\equiv_{\beta\eta}\boldsymbol{B}$]) when $\boldsymbol{A}$ is β-convertible [η-convertible, $\beta\eta$-convertible] to $\boldsymbol{B}$.

2.2. Higher-order logic ($\mathscr{HOL}$). In $\mathscr{HOL}$, the set of base types is $\{o, \iota\}$ for truth values and individuals. We will call a formula of type o a *proposition*, and a *sentence* if it is closed. We will assume that the signature Σ contains logical constants for *negation* $(\neg_{o\to o})$, *disjunction* $(\vee_{o\to o\to o})$, and *universal quantification* $(\Pi^\alpha_{(\alpha\to o)\to o})$ for each type α. Optionally, Σ may contain *primitive equality* $(=^\alpha_{\alpha\to\alpha\to o})$ for each type

[1] We could also have used de Bruijn's indices [19] as a concrete implementation of this approach at the syntax level.

α. All other constants are called *parameter*s, since the argumentation in this article is parametric in their choice.

We write disjunctions and equations, i.e., terms of the form $((\vee \boldsymbol{A})\boldsymbol{B})$ or $((= \boldsymbol{A})\boldsymbol{B})$, in infix notation as $\boldsymbol{A} \vee \boldsymbol{B}$ and $\boldsymbol{A} = \boldsymbol{B}$. As we only assume the logical constants $\neg$, $\vee$, and Π^α (and possibly $=^\alpha$) as primitive, we will use formulae of the form $\boldsymbol{A} \wedge \boldsymbol{B}$, $\boldsymbol{A} \Rightarrow \boldsymbol{B}$, and $\boldsymbol{A} \Leftrightarrow \boldsymbol{B}$ as shorthand for the formulae $\neg((\neg \boldsymbol{A}) \vee (\neg \boldsymbol{B}))$, and $(\neg \boldsymbol{A}) \vee \boldsymbol{B}$, and $(\boldsymbol{A} \Rightarrow \boldsymbol{B}) \wedge (\boldsymbol{B} \Rightarrow \boldsymbol{A})$, respectively. For each $\boldsymbol{A} \in \mathrm{wff}_o(\Sigma)$, the standard notations $\forall X_\alpha . \boldsymbol{A}$ and $\exists X_\alpha . \boldsymbol{A}$ for quantification are regarded as shorthand for $\Pi^\alpha(\lambda X_\alpha . \boldsymbol{A})$ and $\neg(\Pi^\alpha(\lambda X_\alpha . \neg \boldsymbol{A}))$. Finally, we extend the vector notation for λ-binders to k-fold quantification: we will use $\forall \overline{X^k} . \boldsymbol{A}$ and $\exists \overline{X^k} . \boldsymbol{A}$ in the obvious way.

We often need to distinguish between atomic and non-atomic formulae in $\mathrm{wff}_o(\Sigma)$. A non-atomic formula is any formula whose β-normal form is either of the form $\neg \boldsymbol{A}$, $\boldsymbol{A} \vee \boldsymbol{B}$, or $\Pi^\alpha \boldsymbol{C}$ (where $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_o(\Sigma)$ and $\boldsymbol{C} \in \mathrm{wff}_{\alpha \to o}(\Sigma)$). An atomic formula is any other formula in $\mathrm{wff}_o(\Sigma)$—including primitive equations $\boldsymbol{A} =^\alpha \boldsymbol{B}$ in case of the presence of primitive equality.

It is matter of folklore that equality can directly be expressed in $\mathscr{HOL}$. A prominent example is the *Leibniz formula* for equality

$$\mathbf{Q}^\alpha := (\lambda X_\alpha \, Y_\alpha . \forall P_{\alpha \to o} . PX \Rightarrow PY).$$

With this definition, the formula $(\boldsymbol{Q}^\alpha \boldsymbol{A}\boldsymbol{B})$ (expressing equality of two formulae $\boldsymbol{A}$ and $\boldsymbol{B}$ of type α) β-reduces to $\forall P_{\alpha \to o} . (P\boldsymbol{A}) \Rightarrow (P\boldsymbol{B})$, which can be read as: formulae $\boldsymbol{A}$ and $\boldsymbol{B}$ are not equal iff there exists a discerning property P.[2] In other words, $\boldsymbol{A}$ and $\boldsymbol{B}$ are equal, if they are indiscernible. We will use the notation $\boldsymbol{A} \doteq^\alpha \boldsymbol{B}$ as shorthand for the β-reduct $\forall P_{\alpha \to o} . (P\boldsymbol{A}) \Rightarrow (P\boldsymbol{B})$ of $(\boldsymbol{Q}^\alpha \boldsymbol{A}\boldsymbol{B})$ (where $P \notin \mathrm{free}(\boldsymbol{A}) \cup \mathrm{free}(\boldsymbol{B})$).[3] There are alternative ways to define equality in terms of the logical connectives ([6, p. 203]) and the techniques for equality introduced in this article carry over to them (cf. Remark 4.4).

In this article we use several different notions of equality. In order to prevent misunderstandings we explain these different notions together with their syntactical representation here:

If we *define* a concept we use $:=$ (e.g., let $\mathscr{D} := \{\mathtt{T}, \mathtt{F}\}$). $\equiv$ represents identity. We refer to a representative of the identity relation on $\mathscr{D}_\alpha$ as an *object* of the *semantical domain* $\mathscr{D}_{\alpha \to \alpha \to o}$ with q^α. Note that we possibly have one, several, or no q^α in $\mathscr{D}_{\alpha \to \alpha \to o}$ for each domain $\mathscr{D}_\alpha$. The remaining two notions are related to syntax. $=^\alpha$ may occur as a *constant symbol* of type $\alpha \to \alpha \to o$ in a signature Σ. Finally, $\doteq^\alpha$ and $\mathbf{Q}^\alpha$ are used for *Leibniz equality* as described above.

2.3. Notions of models for $\mathscr{HOL}$. A model of $\mathscr{HOL}$ is a collection of non-empty domains $\mathscr{D}_\alpha$ for all types α together with a way of interpreting formulae. The model classes discussed in this article will vary in the domains and specifics of the evaluation of formulae. The relationships between these classes of models are depicted as a cube in Figure 1. We will discuss the model classes from bottom to top, from the most specific notion of standard models ($\mathfrak{ST}$) to the most general notion of υ-complexes, motivating the respective generalizations as we go along. In Section 3, where we develop the theory formally based on the intuitions discussed

[2] Note that this is symmetric by considering complements and hence it is sufficient to use $\Rightarrow$ instead of $\Leftrightarrow$.

[3] Note that $\boldsymbol{A} \doteq^\alpha \boldsymbol{B}$ is β-normal iff $\boldsymbol{A}$ and $\boldsymbol{B}$ are β-normal. The same holds for $\beta\eta$-equality.

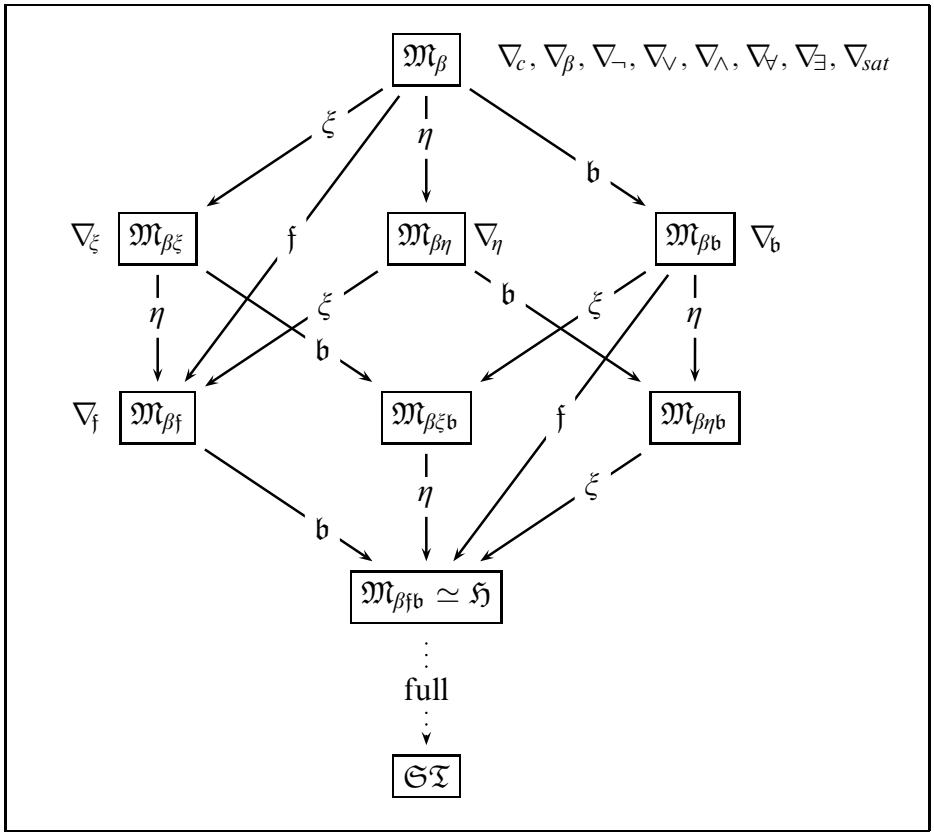


FIGURE 1. The landscape of higher-order semantics.

here, we will proceed the other way around, specializing the notion of a Σ-model more and more.

The symbols in the boxes in Figure 1 denote model classes, the symbols labeling the arrows indicate the properties inducing the corresponding specialization, and the ∇-symbols next to the boxes indicate the clauses in the definition of abstract consistency classes (cf. Definition 6.5) that are needed to establish a model existence theorem for this particular class of models (cf. Theorem 6.34).

2.3.1. *Standard and Henkin models* $[\mathfrak{ST}, \mathfrak{H}, \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}]$. A *standard model* ($\mathfrak{ST}$, cf. Definition 3.51) for $\mathscr{HOL}$ provides a fixed set $\mathscr{D}_\iota$ of individuals and a set $\mathscr{D}_o := \{\mathtt{T}, \mathtt{F}\}$ of truth values. All the domains for the function types are defined inductively: $\mathscr{D}_{\alpha\to\beta}$ is the set of functions $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$. The evaluation function $\mathscr{E}_\varphi$ with respect to an assignment φ of variables is obtained by the standard homomorphic construction that evaluates a λ-abstraction with a function.

One can reconstruct the key idea behind *Henkin models* ($\mathfrak{H}$ isomorphic to $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$, cf. Definitions 3.50, and Theorem 3.68) by the following observation. If the set $\mathscr{D}_\iota$ is infinite, the set $\mathscr{D}_{\iota\to o}$ of sets of individuals must be uncountably infinite. On the other hand, any reasonable semantics of a language with a countable signature that admits

sound and complete calculi must have countable models. Leon Henkin generalized the class of admissible domains for functional types [26]. Instead of requiring $\mathscr{D}_{\alpha\to\beta}$ (and thus in particular, $\mathscr{D}_{\iota\to o}$) to be the full set of functions (predicates), it is sufficient to require that $\mathscr{D}_{\alpha\to\beta}$ has enough members that any well-formed formula can be evaluated (in other words, the domains of function types are rich enough to satisfy comprehension). Note that with this generalized notion of a model, there are fewer formulae that are valid in all models (intuitively, for any given formula there are more possibilities for counter-models). The generalization to Henkin models restricts the set of valid formulae sufficiently so that all of them can be proven by a Hilbert-style calculus [26].

Of course our picture in Figure 1 is not complete here; we can axiomatically require the existence of particular (classes of) functions, e.g., by assuming the description or choice operators. We will not pursue this here; for a detailed discussion of the semantic issues raised by the presence of these logical constants see [3]. Note that even though we can consider model classes with richer and richer function spaces, we can never reach standard models where function spaces are full while maintaining complete (recursively axiomatizable) calculi.

2.3.2. *Models without boolean extensionality* [$\mathfrak{M}_\beta, \mathfrak{M}_{\beta\xi}, \mathfrak{M}_{\beta\eta}, \mathfrak{M}_{\beta\mathfrak{f}}$]. The next generalization of model classes comes from the fact that we want to have logics where the axiom of Boolean extensionality can fail. For instance, in the semantics of natural language we have so-called verbs and adjectives of "propositional attitude" like *believe* or *obvious*. We may not want to commit ourselves to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog" since John might not be aware that "woodchuck" is just another word for "groundhog". The axiom of Boolean extensionality does just that; it states that whenever two propositions are equivalent, they must be equal, and can be substituted for each other. Similarly, the formulae obvious($\boldsymbol{O}$) and obvious($\boldsymbol{F}$) where $\boldsymbol{O} := 2 + 2 = 4$ and $\boldsymbol{F} := \forall n > 2.x^n + y^n = z^n \Rightarrow x = y = z = 0$ should not be equivalent, even if their arguments are. (Both $\boldsymbol{O}$ and $\boldsymbol{F}$ are true over the natural numbers, but Fermat's last theorem $\boldsymbol{F}$ is non-obvious to most people). These phenomena have been studied under the heading of "hyper-intensional semantics" in theoretical semantics; see [39] for a survey.

To account for this behavior, we have to generalize the class of Henkin models further so that there are counter-models to the examples above. Obviously, this involves weakening the assumption that $\mathscr{D}_o \equiv \{\mathtt{T}, \mathtt{F}\}$ since this entails that the values of $\boldsymbol{O}$ and $\boldsymbol{F}$ are identical. We call the assumption that $\mathscr{D}_o$ has two elements property $\mathfrak{b}$. In our Σ-*models* without property $\mathfrak{b}$ ($\mathfrak{M}_\beta$, $\mathfrak{M}_{\beta\xi}$, $\mathfrak{M}_{\beta\eta}$, $\mathfrak{M}_{\beta\mathfrak{f}}$, cf. Definitions 3.41 and 3.49) we only insist that there is a division of the truth values into "good" and "bad" ones, which we express by insisting on the existence of a valuation v of $\mathscr{D}_o$, i.e., a function $v\colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ that is coordinated with the interpretations of the logical constants $\neg$, $\vee$, and Π^α (for each type α). Thus we have a notion of validity: we call a sentence $\boldsymbol{A}$ valid in such a model if $v(\mathtt{a}) \equiv \mathtt{T}$, where $\mathtt{a} \in \mathscr{D}_o$ is the value of the sentence $\boldsymbol{A}$. For example, there is a Σ-model (see Examples 5.4 and 5.5) where woodchuck(phil), groundhog(phil) and believe(john, woodchuck(phil)) are all valid, but believe(john, groundhog(phil)) is not. In this model, the value of woodchuck(phil) is different from the value of groundhog(phil) in $\mathscr{D}_o$.

2.3.3. *Models without functional extensionality* [$\mathfrak{M}_\beta$, $\mathfrak{M}_{\beta\eta}$, $\mathfrak{M}_{\beta\xi}$, $\mathfrak{M}_{\beta\mathfrak{b}}$, $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, $\mathfrak{M}_{\beta\xi\mathfrak{b}}$]. In mathematics (and as a consequence in most higher-order model theories), we assume functional extensionality, which states that two functions are equal, if they return identical values on all arguments. In many applications we want to use a logic that allows a finer-grained modeling of properties of functions. For instance, if we want to model programs as (higher-order) functions, we might be interested in intensional[4] properties like run-time complexity. Consider for instance the two functions $\boldsymbol{I} := \lambda X.X$ and $\boldsymbol{L} := \lambda X.\mathrm{rev}(\mathrm{rev}(X))$, where rev is the self-inverse function that reverses the order of elements in a list. While the identity function has constant complexity, the function rev is linear in the length of its argument. As a consequence, even though $\boldsymbol{L}$ behaves like $\boldsymbol{I}$ on all inputs, they have different time complexity. A logic with a functionally extensional model theory (which is encoded as property $\mathfrak{f}$, cf. Definitions 3.5, 3.41 and 3.46) would conflate $\boldsymbol{I}$ and $\boldsymbol{L}$ semantically and thus hide this difference rendering the logic unsuitable for complexity analysis.

To arrive at a model theory which does not require functional extensionality (which we will a call non-functional model theory in the remainder) we need to generalize the notion of domains at function types and evaluation functions. This is because the usual construction already uses sets of (extensional) functions for the domains of function type and the property of functionality to construct values for λ-terms.

We build on the notion of applicative structures (cf. Definition 3.1) to define Σ-evaluations (cf. Definition 3.18), where the evaluation function is assumed to respect application and β-conversion. In such models, a function is not uniquely determined by its behavior on all possible arguments. Such models can be constructed, for example, by labeling 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 (cf. Example 5.6). Property $\mathfrak{b}$ may or may not hold for non-functional Σ-Models.

We can factor functional extensionality (property $\mathfrak{f}$) into two independent properties, property η and property ξ. A model satisfies property η if it respects η-conversion. A model satisfies property ξ if we can conclude the values of $\lambda X.\boldsymbol{M}$ and $\lambda X.\boldsymbol{N}$ are identical whenever the values of $\boldsymbol{M}$ and $\boldsymbol{N}$ are identical for any assignment of the variable X. We will show that a model satisfies property $\mathfrak{f}$ iff it satisfies both property η and property ξ (cf. Lemma 3.24).

2.3.4. *Andrews' models and υ-complexes* [$\mathfrak{M}_\beta, \mathfrak{M}_{\beta\eta}$]. Peter Andrews has pioneered the construction of non-functional models with his υ-complexes in [1] based on Kurt Schütte's semi-valuation method [50]. These constructions, where both functional and Boolean extensionality fail, are Σ-models as defined in Definition 3.41. (Typically they will not even satisfy the property that Leibniz equality corresponds to identity in the model, but they will have a quotient by Theorem 3.62 which does satisfy this property.)

2.4. Characterizing the deductive power of calculi. These model classes discussed in the previous section characterize the deductive power of many higher-order

[4] Just as in the linguistic application, the word "intensional" is used as a synonym for "non-extensional" even though totally different properties are intended.

theorem provers on a semantic level. For example, TPS [8] can be used in modes in which the deductive power is characterized by $\mathfrak{M}_{\beta\eta}$ (or even $\mathfrak{M}_{\beta}$ if η-conversion is disallowed). Note that in particular TPS is not complete with respect to Henkin models. It is not even complete for $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, although it can be used in modes with some 'extensionality treatment' built into the proof procedure.

The incompleteness of TPS for Henkin models[5] can be seen from the fact that it fails to refute formulae such as $c\boldsymbol{A}_o \wedge \neg c(\neg\neg\boldsymbol{A})$, where c is a constant of type $o \to o$, or to prove formulae like $p(\lambda X_\alpha.\boldsymbol{B}X \wedge \boldsymbol{A}X) \Rightarrow p(\lambda X_\alpha.\boldsymbol{A}X \wedge \boldsymbol{B}X)$, where p is a constant of type $(\alpha \to o) \to o$. The problem in the former example is that the higher-order unification algorithm employed by TPS cannot determine that $\boldsymbol{A}$ and $\neg\neg\boldsymbol{A}$ denote identical semantic objects (by Boolean extensionality as already mentioned before), and thus returns failure instead of success. In the second example both functional and Boolean extensionality are needed in order to prove the theorem.

[21] discusses a presentation of higher-order logic in a first-order logic based on an approach called *theorem proving modulo*. It is easy to check that this approach is also incomplete for model classes with property $\mathfrak{b}$. For instance the approach cannot prove the formula

$$\forall P_{o\to o} X_o Y_o.\ (PX \wedge PY) \Rightarrow P(X \wedge Y)$$

which is valid in Henkin models and which requires $\mathfrak{b}$. As a result, the *theorem proving modulo* approach of representing higher-order logic in a first-order logic [21] can only be used for logics without Boolean extensionality in its current form.

2.4.1. *Model existence theorems.* For all the notions of model classes (except, of course, for standard models, where such a theorem cannot hold for recursively axiomatizable logical systems) we present model existence theorems tying the differentiating conditions of the models to suitable conditions in the abstract consistency classes (cf. Section 6.3).

A model existence theorem for a logical system $\mathcal{S}$ (i.e., a logical language $\mathscr{L}_{\mathcal{S}}$ together with a consequence relation $\models_{\mathcal{S}} \subseteq \mathscr{L}_{\mathcal{S}} \times \mathscr{L}_{\mathcal{S}}$) is a theorem of the form:

> *If a set of sentences Φ of $\mathcal{S}$ is a member of an abstract consistency class Γ, then there exists a $\mathcal{S}$-model for Φ.*

For the proof we can use the classical construction in all cases: abstract consistent sets are extended to Hintikka sets (cf. Section 6.2), which induce a valuation on a term structure (cf Definition 3.35). We then take a quotient by the congruence induced by Leibniz equality in the term model.

2.4.2. *Completeness of calculi.* Given a model existence theorem as described above we can show the completeness of a particular calculus $\mathscr{C}$ (i.e., the derivability relation $\vdash_{\mathcal{S}} \subseteq \mathscr{L}_{\mathcal{S}} \times \mathscr{L}_{\mathcal{S}}$) by proving that the class Γ of sets of sentences Φ that are $\mathscr{C}$-consistent (i.e., cannot be refuted in $\mathscr{C}$) is an abstract consistency class. Then the model existence theorem tells us that $\mathscr{C}$-consistent sets of sentences are satisfiable in $\mathcal{S}$. Now we assume that a sentence $\boldsymbol{A}$ is valid in $\mathcal{S}$, so $\neg\boldsymbol{A}$ does not have a $\mathcal{S}$-model and is therefore $\mathscr{C}$-inconsistent. Hence, $\neg\boldsymbol{A}$ is refutable in $\mathscr{C}$. This shows

[5] 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.

refutation completeness of $\mathscr{C}$. For many calculi $\mathscr{C}$, this also shows $\boldsymbol{A}$ is provable, thus establishing completeness of $\mathscr{C}$.

Note that with this argumentation the completeness proof for $\mathscr{C}$ condenses to verifying that Γ is an abstract consistency class, a task that does not refer to $\mathcal{S}$-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. In this respect a model existence theorem is similar to a Herbrand Theorem, but it is easier to generalize to other logic systems like higher-order logic. The technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [29, 52, 53].

§3. Semantics for higher-order logic. In this section we will introduce the semantical constructions and discuss their relationships. We will start out by defining applicative structures and Σ-evaluations to give an algebraic semantics for the simply typed λ-calculus. To obtain a model for higher-order logic, we use a Σ-valuation to determine whether propositions are true or false.

3.1. Applicative structures.

DEFINITION 3.1 ((Typed) Applicative structure). A collection $\mathscr{D} := \mathscr{D}_{\mathcal{T}} := \{ \mathscr{D}_\alpha \mid \alpha \in \mathcal{T} \}$ of non-empty sets $\mathscr{D}_\alpha$, indexed by the set $\mathcal{T}$ of types, is called a *typed collection* (of sets). Let $\mathscr{D}_{\mathcal{T}}$ and $\mathscr{E}_{\mathcal{T}}$ be typed collections, then a collection $\mathsf{f} := \{ \mathsf{f}^\alpha : \mathscr{D}_\alpha \longrightarrow \mathscr{E}_\alpha \mid \alpha \in \mathcal{T} \}$ of functions is called a *typed function* $\mathsf{f} : \mathscr{D}_{\mathcal{T}} \longrightarrow \mathscr{E}_{\mathcal{T}}$. We will write $\mathscr{F}(A;B)$ for the set of functions from A to B and $\mathscr{F}_{\mathcal{T}}(\mathscr{D}_{\mathcal{T}};\mathscr{E}_{\mathcal{T}})$ for the set of typed functions. In the following we will also use the notion of a typed function extended to the n-ary case in the obvious way.

We call the pair $(\mathscr{D}, @)$ a (typed) *applicative structure* if $\mathscr{D} \equiv \mathscr{D}_{\mathcal{T}}$ is a typed collection of sets and

$$@ := \{ @^{\alpha\beta} : \mathscr{D}_{\alpha\to\beta} \times \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta \mid \alpha, \beta \in \mathcal{T} \}.$$

Each (non-empty) set $\mathscr{D}_\alpha$ is called the *domain* of type α and the family of functions $@$ is called the *application operator*. We write simply $\mathsf{f}@\mathsf{a}$ for $\mathsf{f}@^{\alpha\beta}\mathsf{a}$ when $\mathsf{f} \in \mathscr{D}_{\alpha\to\beta}$ and $\mathsf{a} \in \mathscr{D}_\alpha$ are clear in context.

REMARK 3.2. Often an applicative structure is defined to also include an interpretation of the constants in a given signature (for example, in [44]). We prefer this signature-independent definition (as in [30]) for our purposes.

REMARK 3.3 (Currying). The application operator $@$ in an applicative 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 $\mathsf{f}@(\mathsf{a}^1, \ldots, \mathsf{a}^n) := (\ldots(\mathsf{f}@\mathsf{a}^1)\ldots@\mathsf{a}^n)$ ("Currying").

DEFINITION 3.4 (Frame). An applicative structure $(\mathscr{D}, @)$ is called a *frame*, if $\mathscr{D}_{\alpha\to\beta} \subseteq \mathscr{F}(\mathscr{D}_\alpha;\mathscr{D}_\beta)$ and $@^{\alpha\beta}$ is application for functions for all types α and β.

DEFINITION 3.5 (Functional/full/standard applicative structures). Let $\mathscr{A} := (\mathscr{D}, @)$ be an applicative structure. We say that $\mathscr{A}$ is *functional* if for all types α and β and objects $\mathsf{f}, \mathsf{g} \in \mathscr{D}_{\alpha\to\beta}$, we have $\mathsf{f} \equiv \mathsf{g}$ whenever $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for every

$\mathsf{a} \in \mathscr{D}_\alpha$.[6] We say $\mathscr{A}$ is *full* if for all types α and β and every function $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ there is an object $\mathsf{f} \in \mathscr{D}_{\alpha\to\beta}$ such that $\mathsf{f@a} \equiv f(\mathsf{a})$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. Finally, we say $\mathscr{A}$ is *standard* if it is a frame and $\mathscr{D}_{\alpha\to\beta} \equiv \mathscr{F}(\mathscr{D}_\alpha; \mathscr{D}_\beta)$ for all types α and β. Note that these definitions impose restrictions on the domains for function types only.

REMARK 3.6. It is easy to show that every frame is functional. Furthermore, an applicative structure is standard iff it is a full frame.

EXAMPLE 3.7 (Applicative singleton structure). We choose a single element a and define $\mathscr{D}_\alpha := \{\mathsf{a}\}$ for all types α. The pair $(\mathscr{D}_\mathscr{T}, @^\mathsf{a})$, where $\mathsf{a}@^\mathsf{a}\mathsf{a} = \mathsf{a}$ is a (trivial) example of a functional applicative structure. It is called the *singleton applicative structure*.

EXAMPLE 3.8 (Applicative term structures). If we define $\boldsymbol{A}@\boldsymbol{B} := (\boldsymbol{AB})$ for $\boldsymbol{A} \in \mathrm{wff}_{\alpha\to\beta}(\Sigma)$ and $\boldsymbol{B} \in \mathrm{wff}_\alpha(\Sigma)$, then $@\colon \mathrm{wff}_{\alpha\to\beta}(\Sigma) \times \mathrm{wff}_\alpha(\Sigma) \longrightarrow \mathrm{wff}_\beta(\Sigma)$ is a total function. Thus $(\mathrm{wff}(\Sigma), @)$ is an applicative structure. The intuition behind this example is that we can think of the formula $\boldsymbol{A} \in \mathrm{wff}_{\alpha\to\beta}(\Sigma)$ as a function $\boldsymbol{A}\colon \mathrm{wff}_\alpha(\Sigma) \longrightarrow \mathrm{wff}_\beta(\Sigma)$ that maps $\boldsymbol{B}$ to $(\boldsymbol{AB})$.

Analogously, we can define the applicative structure $(\mathrm{cwff}(\Sigma), @)$ of closed formulae (when we ensure Σ contains enough constants so that $\mathrm{cwff}_\alpha(\Sigma)$ is non-empty for all types α).

DEFINITION 3.9 (Homomorphism). Let $\mathscr{A}^1 := (\mathscr{D}^1, @^1)$ and $\mathscr{A}^2 := (\mathscr{D}^2, @^2)$ be applicative structures. A *homomorphism* from $\mathscr{A}^1$ to $\mathscr{A}^2$ is a typed function $\kappa\colon \mathscr{D}^1 \longrightarrow \mathscr{D}^2$ such that for all types $\alpha, \beta \in \mathscr{T}$, all $\mathsf{f} \in \mathscr{D}^1_{\alpha\to\beta}$, and $\mathsf{a} \in \mathscr{D}^1_\alpha$ we have $\kappa(\mathsf{f})@^2\kappa(\mathsf{a}) \equiv \kappa(\mathsf{f}@^1\mathsf{a})$. We write $\kappa\colon \mathscr{A}^1 \longrightarrow \mathscr{A}^2$. The two applicative structures $\mathscr{A}^1$ and $\mathscr{A}^2$ are called *isomorphic* if there are homomorphisms $i\colon \mathscr{A}^1 \longrightarrow \mathscr{A}^2$ and $j\colon \mathscr{A}^2 \longrightarrow \mathscr{A}^1$ which are mutually inverse at each type.

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

DEFINITION 3.10 (Applicative structure congruences). Let $\mathscr{A} := (\mathscr{D}, @)$ be an applicative structure. A typed equivalence relation $\sim$ is called a *congruence* on $\mathscr{A}$ iff for all $\mathsf{f}, \mathsf{f}' \in \mathscr{D}_{\alpha\to\beta}$ and $\mathsf{a}, \mathsf{a}' \in \mathscr{D}_\alpha$ (for any types α and β), $\mathsf{f} \sim \mathsf{f}'$ and $\mathsf{a} \sim \mathsf{a}'$ imply $\mathsf{f@a} \sim \mathsf{f}'@\mathsf{a}'$.

The *equivalence class* $[\![\mathsf{a}]\!]_\sim$ of $\mathsf{a} \in \mathscr{D}_\alpha$ *modulo* $\sim$ is the set of all $\mathsf{a}' \in \mathscr{D}_\alpha$, such that $\mathsf{a} \sim \mathsf{a}'$. A congruence $\sim$ is called *functional* iff for all types α and β and $\mathsf{f}, \mathsf{g} \in \mathscr{D}_{\alpha\to\beta}$, we have $\mathsf{f} \sim \mathsf{g}$ whenever $\mathsf{f@a} \sim \mathsf{g@a}$ for every $\mathsf{a} \in \mathscr{D}_\alpha$.

LEMMA 3.11. *The β-equality and $\beta\eta$-equality relations $\equiv_\beta$ and $\equiv_{\beta\eta}$ are congruences on the applicative structures* $\mathrm{wff}(\Sigma)$ *and* cwff.

PROOF. The congruence properties are a direct consequence of the fact that $\beta\eta$-reduction rules are defined to act on subterm positions. ⊣

[6] This is called "extensional" in [44]. We use the term "functional" to distinguish it from other forms of extensionality.

DEFINITION 3.12 (Quotient applicative structure). Let $\mathscr{A} := (\mathscr{D}, @)$ be an applicative structure, $\sim$ a congruence on $\mathscr{A}$, and $\mathscr{D}_\alpha^\sim := \{[\![\mathsf{a}]\!]_\sim \mid \mathsf{a} \in \mathscr{D}_\alpha\}$. Furthermore, let $@^\sim$ be defined by $[\![\mathsf{f}]\!]_\sim @^\sim [\![\mathsf{a}]\!]_\sim := [\![\mathsf{f}@\mathsf{a}]\!]_\sim$. (To see that this definition only depends on equivalence classes of $\sim$, consider $\mathsf{f}' \in [\![\mathsf{f}]\!]_\sim$ and $\mathsf{a}' \in [\![\mathsf{a}]\!]_\sim$. Then $\mathsf{f} \sim \mathsf{f}'$ and $\mathsf{a} \sim \mathsf{a}'$ imply $\mathsf{f}@\mathsf{a} \sim \mathsf{f}'@\mathsf{a}'$. Thus, $[\![\mathsf{f}@\mathsf{a}]\!]_\sim \equiv [\![\mathsf{f}'@\mathsf{a}']\!]_\sim$. So, $@^\sim$ is well-defined.) $\mathscr{A}/_\sim := (\mathscr{D}^\sim, @^\sim)$ is also an applicative structure. We call $\mathscr{A}/_\sim$ the *quotient structure* of $\mathscr{A}$ for the relation $\sim$ and the typed function $\pi_\sim \colon \mathscr{A} \longrightarrow \mathscr{A}/_\sim$ that maps a to $[\![\mathsf{a}]\!]_\sim$ its *canonical projection*.

THEOREM 3.13. *Let $\mathscr{A}$ be an applicative structure and let $\sim$ be a congruence on $\mathscr{A}$, then the canonical projection $\pi_\sim$ is a surjective homomorphism. Furthermore, $\mathscr{A}/_\sim$ is functional iff $\sim$ is functional.*

PROOF. Let $\mathscr{A} := (\mathscr{D}, @)$ be an applicative structure. To convince ourselves that $\pi_\sim$ is indeed a surjective homomorphism, we note that $\pi_\sim$ is surjective by the definition of $\mathscr{D}^\sim$. To see that $\pi_\sim$ is a homomorphism let $\mathsf{f} \in \mathscr{D}_{\alpha\to\beta}$, and $\mathsf{a} \in \mathscr{D}_\beta$, then $\pi_\sim(\mathsf{f})@^\sim \pi_\sim(\mathsf{a}) \equiv [\![\mathsf{f}]\!]_\sim @^\sim [\![\mathsf{a}]\!]_\sim \equiv [\![\mathsf{f}@\mathsf{a}]\!]_\sim \equiv \pi_\sim(\mathsf{f}@\mathsf{a})$.

The quotient construction collapses $\sim$ to identity, so functionality of $\sim$ is equivalent to functionality of $\mathscr{A}/_\sim$. Formally, suppose $[\![\mathsf{f}]\!]_\sim$ and $[\![\mathsf{g}]\!]_\sim$ are elements of $\mathscr{D}^\sim_{\alpha\to\beta}$ such that $[\![\mathsf{f}]\!]_\sim @^\sim [\![\mathsf{a}]\!]_\sim \equiv [\![\mathsf{g}]\!]_\sim @^\sim [\![\mathsf{a}]\!]_\sim$ for every $[\![\mathsf{a}]\!]_\sim$ in $\mathscr{D}^\sim_\alpha$. This is equivalent to $[\![\mathsf{f}@\mathsf{a}]\!]_\sim \equiv [\![\mathsf{g}@\mathsf{a}]\!]_\sim$ for every $\mathsf{a} \in \mathscr{D}_\alpha$ and hence $\mathsf{f}@\mathsf{a} \sim \mathsf{g}@\mathsf{a}$ for all $\mathsf{a} \in \mathscr{D}_\alpha$. By functionality of $\sim$, we have $\mathsf{f} \sim \mathsf{g}$. That is, $[\![\mathsf{f}]\!]_\sim \equiv [\![\mathsf{g}]\!]_\sim$. ⊣

LEMMA 3.14. *$\equiv_{\beta\eta}$ is a functional congruence on* $\mathrm{wff}(\Sigma)$. *If Σ_α is infinite for all types $\alpha \in \mathscr{T}$, then $\equiv_{\beta\eta}$ is also functional on* cwff.

PROOF. By Lemma 3.11, $\equiv_{\beta\eta}$ is a congruence relation. To show functionality let $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_{\gamma\to\alpha}(\Sigma)$ such that $\boldsymbol{AC} \equiv_{\beta\eta} \boldsymbol{BC}$ for all $\boldsymbol{C} \in \mathrm{wff}_\gamma(\Sigma)$ be given. In particular, for any variable $X \in \mathscr{V}_\gamma$ that is not free in $\boldsymbol{A}$ or $\boldsymbol{B}$, we have $\boldsymbol{A}X \equiv_{\beta\eta} \boldsymbol{B}X$ and $\lambda X.\boldsymbol{A}X \equiv_{\beta\eta} \lambda X.\boldsymbol{B}X$. By definition we have $\boldsymbol{A} \equiv_\eta \lambda X_\gamma.\boldsymbol{A}X \equiv_{\beta\eta} \lambda X_\gamma.\boldsymbol{B}X \equiv_\eta \boldsymbol{B}$.

To show functionality of $\beta\eta$-equality on closed formulae, suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are closed. With the same variable X as above, let $\boldsymbol{M}$ and $\boldsymbol{N}$ be the $\beta\eta$-normal forms of $\boldsymbol{A}X$ and $\boldsymbol{B}X$, respectively. We cannot conclude that $\boldsymbol{M} \equiv \boldsymbol{N}$ since X is not a closed term. Instead, choose a constant $c_\gamma \in \Sigma_\gamma$ that does not occur in $\boldsymbol{A}$ or $\boldsymbol{B}$. (Such a constant must exist, since we have assumed that Σ_γ is infinite.) An easy induction on the length of the $\beta\eta$-reduction sequence from $\boldsymbol{A}X$ to $\boldsymbol{M}$ shows that c does not occur in $\boldsymbol{M}$ and $\boldsymbol{A}c \equiv [c/X](\boldsymbol{A}X)$ $\beta\eta$-reduces to $[c/X]\boldsymbol{M}$. Similarly, c does not occur in $\boldsymbol{N}$ and $\boldsymbol{B}c$ $\beta\eta$-reduces to $[c/X]\boldsymbol{N}$. Since c is a constant, substituting c for X cannot introduce new redexes. So, simple inductions on the sizes of $\boldsymbol{M}$ and $\boldsymbol{N}$ show $[c/X]\boldsymbol{M}$ and $[c/X]\boldsymbol{N}$ are $\beta\eta$-normal. By assumption, we know $\boldsymbol{A}c \equiv_{\beta\eta} \boldsymbol{B}c$. Since normal forms are unique, we must have $[c/X]\boldsymbol{M} \equiv [c/X]\boldsymbol{N}$. Using the fact that c does not occur in either $\boldsymbol{M}$ or $\boldsymbol{N}$, an induction on the size of $\boldsymbol{M}$ readily shows $\boldsymbol{M} \equiv \boldsymbol{N}$. So, we have $\boldsymbol{A} \equiv_\eta \lambda X_\gamma.\boldsymbol{A}X \equiv_{\beta\eta} \lambda X_\gamma.\boldsymbol{M} \equiv \lambda X_\gamma.\boldsymbol{N} \equiv_{\beta\eta} \lambda X_\gamma.\boldsymbol{B}X \equiv_\eta \boldsymbol{B}$ ⊣

REMARK 3.15. Suppose we have a signature Σ with a single constant c_ι. In this case, c is the only closed $\beta\eta$-normal form of type ι. Since $\lambda X.X \not\equiv_{\beta\eta} \lambda X.c$ even though $(\lambda X.X)c \equiv_{\beta\eta} c \equiv_{\beta\eta} (\lambda X.c)c$ we have a counterexample to functionality of $\equiv_{\beta\eta}$ on cwff. The problem here is that we do not have another constant d_ι to distinguish the two functions. In $\mathrm{wff}(\Sigma)$ we could always use a variable.

REMARK 3.16 (Assumptions on Σ). From now on, we assume Σ_α to be infinite for each type α. Furthermore, we assume there is a particular cardinal $\aleph_s$ such that Σ_α has cardinality $\aleph_s$ for every type α. Since $\mathscr{V}$ is countable, this implies $\mathrm{wff}_\alpha(\Sigma)$ and cwff_α have cardinality $\aleph_s$ for each type α. Also, whether or not primitive equality is included in the signature, there can only be finitely many logical constants in Σ_α for each particular type α. Thus, the cardinality of the set of parameters in Σ_α is also $\aleph_s$. In the countable case, $\aleph_s$ is $\aleph_0$.

3.2. Σ-evaluations. Σ-evaluations are applicative structures with a notion of evaluation for well-formed formulae in $\mathrm{wff}(\Sigma)$.

DEFINITION 3.17 (Variable assignment). Let $\mathscr{A} := (\mathscr{D}, @)$ be an applicative structure. A typed function $\varphi\colon \mathscr{V} \longrightarrow \mathscr{D}$ is called a *variable assignment* into $\mathscr{A}$. Given a variable assignment φ, variable X_α, and value $\mathsf{a} \in \mathscr{D}_\alpha$, we use $\varphi, [\mathsf{a}/X]$ to denote the variable assignment with $(\varphi, [\mathsf{a}/X])(X) \equiv \mathsf{a}$ and $(\varphi, [\mathsf{a}/X])(Y) \equiv \varphi(Y)$ for variables Y other than X.

DEFINITION 3.18 (Σ-evaluation). Let $\mathscr{E}\colon \mathscr{F}_{\mathscr{T}}(\mathscr{V}; \mathscr{D}) \longrightarrow \mathscr{F}_{\mathscr{T}}(\mathrm{wff}(\Sigma), \mathscr{D})$ be a total function, where $\mathscr{F}_{\mathscr{T}}(\mathscr{V}; \mathscr{D})$ is the set of variable assignments and $\mathscr{F}_{\mathscr{T}}(\mathrm{wff}(\Sigma), \mathscr{D})$ is the set of typed functions mapping terms into objects in $\mathscr{D}$. We will write the argument of $\mathscr{E}$ as a subscript. So, for each assignment φ, we have a typed function $\mathscr{E}_\varphi\colon \mathrm{wff}(\Sigma) \longrightarrow \mathscr{D}$. $\mathscr{E}$ is called an *evaluation function* for $\mathscr{A}$ if for any assignments φ and ψ into $\mathscr{A}$, we have

(1) $\mathscr{E}_\varphi\big|_{\mathscr{V}} \equiv \varphi$.
(2) $\mathscr{E}_\varphi(\boldsymbol{FA}) \equiv \mathscr{E}_\varphi(\boldsymbol{F})@\mathscr{E}_\varphi(\boldsymbol{A})$ for any $\boldsymbol{F} \in \mathrm{wff}_{\alpha\to\beta}(\Sigma)$ and $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and types α and β.
(3) $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\psi(\boldsymbol{A})$ for any type α and $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, whenever φ and ψ coincide on $\mathrm{free}(\boldsymbol{A})$.
(4) $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}\downarrow_\beta)$ for all $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$.

We call $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ a Σ-*evaluation* if $(\mathscr{D}, @)$ is an applicative structure and $\mathscr{E}$ is an evaluation function for $(\mathscr{D}, @)$. We call $\mathscr{E}_\varphi(\boldsymbol{A}_\alpha) \in \mathscr{D}_\alpha$ the *denotation* of $\boldsymbol{A}_\alpha$ in $\mathscr{J}$ for φ. (Note that since $\mathscr{E}$ is a function, the denotation in $\mathscr{J}$ is unique. However, for a given applicative structure $\mathscr{A}$, there may be many possible evaluation functions.)

If $\boldsymbol{A}$ is a closed formula, then $\mathscr{E}_\varphi(\boldsymbol{A})$ is independent of φ, since $\mathrm{free}(\boldsymbol{A}) = \emptyset$. In these cases we sometimes drop the reference to φ from $\mathscr{E}_\varphi(\boldsymbol{A})$ and simply write $\mathscr{E}(\boldsymbol{A})$.

We call a Σ-evaluation $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ *functional* [*full*, *standard*] if the applicative structure $(\mathscr{D}, @)$ is *functional* [*full*, *standard*]. We say $\mathscr{J}$ is a Σ-evaluation over a frame if $(\mathscr{D}, @)$ is a frame.

Σ-evaluations generalize Σ-evaluations over frames, which are the basis for Henkin models, to the non-functional case. The existence of an evaluation function that meets the conditions above seems to be the weakest situation where one would like to speak of a model. We cannot in general assume the evaluation function is uniquely determined by its values on constants as this requires functionality. For example, two evaluation functions $\mathscr{E}$ and $\mathscr{E}'$ on the same applicative structure may agree on all constants, but give a different value to the term $(\lambda X_\iota . X)$. Such an example is constructed and discussed later in Remark 5.7.

REMARK 3.19 (Σ-evaluations respect β-equality). Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation and $\boldsymbol{A} =_\beta \boldsymbol{B}$. For all assignments φ into $(\mathscr{D}, @)$, we have $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}\downarrow_\beta) \equiv \mathscr{E}_\varphi(\boldsymbol{B}\downarrow_\beta) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$.

We can easily show Σ-evaluations satisfy a *Substitution-Value Lemma*.

LEMMA 3.20 (Substitution-value lemma). *Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation and φ be an assignment into $\mathscr{J}$. For any types α and β, variables X_β, and formulae $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and $\boldsymbol{B} \in \mathrm{wff}_\beta(\Sigma)$, we have $\mathscr{E}_{\varphi,[\mathscr{E}_\varphi(\boldsymbol{B})/X]}(\boldsymbol{A}) \equiv \mathscr{E}_\varphi([\boldsymbol{B}/X]\boldsymbol{A})$.*

PROOF. Using the fact that $\mathscr{E}$ respects β-equality (cf. Remark 3.19) and the other properties of $\mathscr{E}$ (cf. Definition 3.18), we can compute

$$\begin{aligned}
\mathscr{E}_{\varphi,[\mathscr{E}_\varphi(\boldsymbol{B})/X]}(\boldsymbol{A}) &\equiv \mathscr{E}_{\varphi,[\mathscr{E}_\varphi(\boldsymbol{B})/X]}((\lambda X_{\cdot}\boldsymbol{A})X) \\
&\equiv \mathscr{E}_{\varphi,[\mathscr{E}_\varphi(\boldsymbol{B})/X]}(\lambda X_{\cdot}\boldsymbol{A})@\mathscr{E}_{\varphi,[\mathscr{E}_\varphi(\boldsymbol{B})/X]}(X) \\
&\equiv \mathscr{E}_\varphi(\lambda X_{\cdot}\boldsymbol{A})@\mathscr{E}_\varphi(\boldsymbol{B}) \\
&\equiv \mathscr{E}_\varphi((\lambda X_{\cdot}\boldsymbol{A})\boldsymbol{B}) \\
&\equiv \mathscr{E}_\varphi([\boldsymbol{B}/X]\boldsymbol{A}). \qquad \dashv
\end{aligned}$$

We will consider two weaker notions of functionality. These forms are often discussed in the literature (cf. [28]).

DEFINITION 3.21 (Weakly functional evaluations). Let $\mathscr{J} \equiv (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation. We say $\mathscr{J}$ is *η-functional* if $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}\downarrow_{\beta\eta})$ for any type α, formula $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, and assignment φ. We say $\mathscr{J}$ is *ξ-functional* if for all $\alpha, \beta \in \mathscr{T}$, $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_\beta(\Sigma)$, assignments φ, and variables X_α, $\mathscr{E}_\varphi(\lambda X_\alpha{\cdot}\boldsymbol{M}_\beta) \equiv \mathscr{E}_\varphi(\lambda X_\alpha{\cdot}\boldsymbol{N}_\beta)$ whenever $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$ for every $\mathsf{a} \in \mathscr{D}_\alpha$.

We will now establish that functionality is equivalent to η-functionality and ξ-functionality combined. We prepare for this by first proving two lemmas about functional Σ-evaluations.

LEMMA 3.22. *Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a functional Σ-evaluation. For any assignment φ into $\mathscr{J}$ and $\boldsymbol{F} \in \mathrm{wff}_{\alpha\to\beta}(\Sigma)$ where $X_\alpha \notin \mathrm{free}(\boldsymbol{F})$, we have*

$$\mathscr{E}_\varphi(\lambda X_\alpha{\cdot}\boldsymbol{F}X) \equiv \mathscr{E}_\varphi(\boldsymbol{F}).$$

PROOF. Let $\mathsf{a} \in \mathscr{D}_\alpha$ be given. Since $X_\alpha \notin \mathrm{free}(\boldsymbol{F})$, we have $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{F}) \equiv \mathscr{E}_\varphi(\boldsymbol{F})$. Since $\mathscr{E}$ respects β-equality (cf. Remark 3.19), we can compute

$$\mathscr{E}_\varphi(\lambda X_{\cdot}\boldsymbol{F}X)@\mathsf{a} \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}((\lambda X_{\cdot}\boldsymbol{F}X)X) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{F}X) \equiv \mathscr{E}_\varphi(\boldsymbol{F})@\mathsf{a}.$$

Generalizing over a, we conclude $\mathscr{E}_\varphi(\lambda X_{\cdot}\boldsymbol{F}X) \equiv \mathscr{E}_\varphi(\boldsymbol{F})$ by functionality. $\dashv$

LEMMA 3.23. *Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a functional Σ-evaluation. If a formula $\boldsymbol{A}$ η-reduces to $\boldsymbol{B}$ in one step, then for any assignment φ into $\mathscr{J}$, $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$.*

PROOF. We prove this by induction on the structure of the term $\boldsymbol{A}$. For the base case when $\boldsymbol{A}$ is the η-redex which is reduced, we apply Lemma 3.22. When $\boldsymbol{A} \equiv (\boldsymbol{F}\boldsymbol{C})$, then the η-reduction either occurs in $\boldsymbol{F}$ or $\boldsymbol{C}$. So, $\boldsymbol{B} \equiv (\boldsymbol{G}\boldsymbol{D})$ where $\boldsymbol{F}$ η-reduces to $\boldsymbol{G}$ in one step (or $\boldsymbol{G} \equiv \boldsymbol{F}$) and $\boldsymbol{D} \equiv \boldsymbol{C}$ (or $\boldsymbol{C}$ η-reduces to $\boldsymbol{D}$ in one step). So, by induction we have $\mathscr{E}_\varphi(\boldsymbol{F}) \equiv \mathscr{E}_\varphi(\boldsymbol{G})$ and $\mathscr{E}_\varphi(\boldsymbol{C}) \equiv \mathscr{E}_\varphi(\boldsymbol{D})$. It follows that $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$.

When $\boldsymbol{A}$ is a λ-abstraction, we must use functionality. Suppose for some type α, $\boldsymbol{A} \equiv (\lambda X_\alpha{\cdot}\boldsymbol{C})$ (and this is not the η-redex reduced to obtain $\boldsymbol{B}$). Then $\boldsymbol{B} \equiv (\lambda X_\alpha \boldsymbol{D})$

where $\boldsymbol{C}$ η-reduces in one step to $\boldsymbol{D}$. By the induction hypothesis, for any $\mathsf{a} \in \mathscr{D}_\alpha$, $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{C}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{D})$. Since $\mathscr{E}$ is an evaluation function, we have

$$\begin{aligned}\mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{C})@\mathsf{a} &\equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}((\lambda X_\alpha.\boldsymbol{C})X) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{C}) \\ &\equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{D}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}((\lambda X_\alpha.\boldsymbol{D})X) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{D})@\mathsf{a}.\end{aligned}$$

By functionality, $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{C}) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{D}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$. ⊣

LEMMA 3.24 (Functionality). *Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation. Then $\mathscr{J}$ is functional iff it is both η-functional and ξ-functional.*

PROOF. The fact that functionality implies η-functionality now follows from a simple induction on the number of $\beta\eta$-reduction steps using Lemma 3.23 and Remark 3.19.

To show functionality implies ξ-functionality, let $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_\beta(\Sigma)$, an assignment φ and a variable X_α be given. Suppose $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. We need to show $\mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{M}) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{N})$. This follows from functionality since

$$\begin{aligned}\mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{M})@\mathsf{a} &\equiv \mathscr{E}_{\varphi,[\mathsf{a},X]}((\lambda X_\alpha.\boldsymbol{M})X) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \\ &\equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N}) \equiv \mathscr{E}_{\varphi,[\mathsf{a},X]}((\lambda X_\alpha.\boldsymbol{N})X) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{N})@\mathsf{a}\end{aligned}$$

for every $\mathsf{a} \in \mathscr{D}_\alpha$.

To show functionality from η-functionality and ξ-functionality, let $\mathsf{f}, \mathsf{g} \in \mathscr{D}_{\alpha\to\beta}$ such that $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for all $\mathsf{a} \in \mathscr{D}_\alpha$ be given. We need to show that $\mathsf{f} \equiv \mathsf{g}$. Let $F_{\alpha\to\beta}$, $G_{\alpha\to\beta}$ and X_α be variables and φ be any assignment such that $\varphi(F) \equiv \mathsf{f}$ and $\varphi(G) \equiv \mathsf{g}$. Then for any $\mathsf{a} \in \mathscr{D}_\alpha$ we have $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(FX) \equiv \mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a} \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(GX)$, and thus $\mathscr{E}_\varphi(\lambda X_\alpha.FX) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.GX)$ by ξ-functionality. Hence,

$$\mathsf{f} \equiv \mathscr{E}_\varphi(F) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.FX) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.GX) \equiv \mathscr{E}_\varphi(G) \equiv \mathsf{g}$$

by η-functionality. ⊣

LEMMA 3.25 (ξ-functionality and replacement). *Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a ξ-functional Σ-evaluation and $\boldsymbol{B}, \boldsymbol{C} \in \mathrm{wff}_\beta(\Sigma)$. Suppose $\mathscr{E}_\varphi(\boldsymbol{B}) \equiv \mathscr{E}_\varphi(\boldsymbol{C})$ for every assignment φ into $\mathscr{J}$. Then for all formulae $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, positions p, and assignments φ into $\mathscr{J}$, $\mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{B}]_p) \equiv \mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{C}]_p)$.*

PROOF. We show the assertion by an induction on the structure of $\boldsymbol{A}$. If p is the top position, we have

$$\mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{B}]_p) \equiv \mathscr{E}_\varphi(\boldsymbol{B}) \equiv \mathscr{E}_\varphi(\boldsymbol{C}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{C}]_p).$$

In particular, if $\boldsymbol{A}$ is a constant or a variable, then p must be the top position and we are done. Otherwise, assume p is not the top position. If $\boldsymbol{A}$ is an application $\boldsymbol{FD}$, we have to consider two cases: $\boldsymbol{A}[\boldsymbol{B}]_p = \boldsymbol{F}[\boldsymbol{B}]_q\boldsymbol{D}$ and $\boldsymbol{A}[\boldsymbol{B}]_p = \boldsymbol{F}(\boldsymbol{D}[\boldsymbol{B}]_r)$ for some positions q and r. Since the second case is analogous we only show the first case. By the inductive hypothesis we have

$$\begin{aligned}\mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{B}]_p) &\equiv \mathscr{E}_\varphi(\boldsymbol{F}[\boldsymbol{B}]_q\boldsymbol{D}) \equiv \mathscr{E}_\varphi(\boldsymbol{F}[\boldsymbol{B}]_q)@\mathscr{E}_\varphi(\boldsymbol{D}) \\ &\equiv \mathscr{E}_\varphi(\boldsymbol{F}[\boldsymbol{C}]_q)@\mathscr{E}_\varphi(\boldsymbol{D}) \equiv \mathscr{E}_\varphi(\boldsymbol{F}[\boldsymbol{C}]_q\boldsymbol{D}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{C}]_p).\end{aligned}$$

If $\boldsymbol{A}[\boldsymbol{B}]_p = \lambda X_\gamma.\boldsymbol{D}[\boldsymbol{B}]_q$, then we get the assertion from ξ-functionality. By the inductive hypothesis, we know $\mathscr{E}_\psi(\boldsymbol{D}[\boldsymbol{B}]_q) \equiv \mathscr{E}_\psi(\boldsymbol{D}[\boldsymbol{C}]_p)$ for every assignment ψ. In particular, for any assignment φ and $\mathsf{c} \in \mathscr{D}_\gamma$, we have $\mathscr{E}_{\varphi,[\mathsf{c}/X]}(\boldsymbol{D}[\boldsymbol{B}]_q) \equiv \mathscr{E}_{\varphi,[\mathsf{c}/X]}(\boldsymbol{D}[\boldsymbol{C}]_p)$.

By ξ-functionality, we have

$$\mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{B}]_p) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{D}[\boldsymbol{B}]_q) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{D}[\boldsymbol{C}]_q) \equiv \mathscr{E}_\varphi(\boldsymbol{A}[\boldsymbol{C}]_p).$$

Thus we have completed all the cases and proven the assertion. ⊣

EXAMPLE 3.26 (Singleton evaluation). The singleton applicative structure (cf. Example 3.7) is a Σ-evaluation if for any assignment φ and formula $\boldsymbol{A}$ we take $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathsf{a}$, where a is the (unique) member of $\mathscr{D}_\alpha$. Note that in this Σ-evaluation $\mathscr{E}(\lambda X_\alpha.X) \equiv \mathscr{E}_\varphi(\lambda X_\alpha.Y)$ for any assignment φ.

For a detailed discussion on the closure conditions needed for the domains for function types to be rich enough for evaluation functions to exist, we refer the reader to [2, 4].

Note that the applicative term structure $\mathrm{wff}(\Sigma)$ from Example 3.8 cannot be made into a Σ-evaluation by providing an evaluation function. To see this, suppose $\mathscr{E}$ is an evaluation function for $\mathrm{wff}(\Sigma)$ and $\boldsymbol{F} := \mathscr{E}(\lambda X_\alpha.X) \in \mathrm{wff}_{\alpha\to\alpha}(\Sigma)$. Since $\mathscr{E}$ is assumed to be an evaluation function, we must have

$$\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi((\lambda X_\alpha.X)\boldsymbol{A}) \equiv \boldsymbol{F}@\boldsymbol{A} \equiv \boldsymbol{F}\boldsymbol{A}$$

for every $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$. In particular, for any constant $a_\alpha \in \Sigma_\alpha$, we must have $\boldsymbol{F}a \equiv \mathscr{E}_\varphi(a) \equiv \mathscr{E}((\lambda X_\alpha.X)a) \equiv \mathscr{E}(\lambda X_\alpha.X)@\mathscr{E}(a) \equiv \boldsymbol{F}(\boldsymbol{F}a)$. But clearly $\boldsymbol{F}a \not\equiv \boldsymbol{F}(\boldsymbol{F}a)$ no matter what $\boldsymbol{F} \in \mathrm{wff}_{\alpha\to\alpha}(\Sigma)$ we choose. In particular, the "obvious" choice of $\mathscr{E}(\lambda X_\alpha.X) \equiv (\lambda X_\alpha.X)$ does not work. This example suggests that we need to consider β-convertible terms equal before we can obtain a term evaluation (cf. Definition 3.35).

DEFINITION 3.27 (Σ-evaluation congruences). A *congruence* on a Σ-evaluation $\mathscr{J} \equiv (\mathscr{D}, @, \mathscr{E})$ is a congruence on the underlying applicative structure $(\mathscr{D}, @)$. Given any two variable assignments φ and ψ into $(\mathscr{D}, @)$, we will use the notation $\varphi \sim \psi$ to indicate that $\varphi(X) \sim \psi(X)$ for every variable X.

A typed equivalence relation was defined to be a congruence if it respects application. In order to form a quotient of a Σ-evaluation, we must be able to define an evaluation function $\mathscr{E}^\sim$ on the quotient structure. But $\mathscr{E}^\sim$ interprets all terms, including λ-abstractions. It is not obvious that one can find a well-defined $\mathscr{E}^\sim$ that is really an evaluation function. In fact, the property one needs in order to show $\mathscr{E}^\sim$ will be a well-defined evaluation function is $\mathscr{E}_\varphi(\boldsymbol{A}) \sim \mathscr{E}_\psi(\boldsymbol{A})$ for all $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and assignments φ and ψ with $\varphi \sim \psi$. One can show this by an easy induction on the term $\boldsymbol{A}$ if the congruence $\sim$ is functional. However, without the assumption that $\sim$ is functional, this direct proof will fail when $\boldsymbol{A}$ is a λ-abstraction. This is a general problem with trying to prove properties of evaluations since many objects in $\mathscr{D}_{\alpha\to\beta}$ may represent the same function from $\mathscr{D}_\alpha$ to $\mathscr{D}_\beta$. Fortunately, there is a way to use combinators to reduce such inductions to terms which only have very special λ-abstractions.

DEFINITION 3.28 (***SK***-combinatory formulae). For all types α, β, and γ, we define two families of closed formulae we call *combinators*:

$$\boldsymbol{K}_{\alpha\to\beta\to\alpha} := \lambda X_\alpha Y_\beta.X$$
$$\boldsymbol{S}_{(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma} := \lambda U_{\alpha\to\beta\to\gamma} V_{\alpha\to\beta} W_\alpha.(UW(VW)).$$

We define the set of ***SK***-*combinatory formulae* to be the least subset of the set $\bigcup_{\alpha\in\mathscr{T}} \mathrm{wff}_\alpha(\Sigma)$ containing every $\boldsymbol{K}$ and $\boldsymbol{S}$, every constant $c \in \Sigma$ and every variable, that is closed under application.

As shown in [3], every formula can be β-expanded to an ***SK***-combinatory formula.

LEMMA 3.29. *For every type α and $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, there is an* ***SK****-combinatory formula $\boldsymbol{A}' \in \mathrm{wff}_\alpha(\Sigma)$ such that $\boldsymbol{A}'$ β-reduces to $\boldsymbol{A}$.*

PROOF. See Proposition 1 in [3]. The main difference to this setup is the signature, and this plays no role in the proof. ⊣

Now, we can show $\mathscr{E}_\varphi(\boldsymbol{A}) \sim \mathscr{E}_\psi(\boldsymbol{A})$ for ***SK***-combinatory $\boldsymbol{A}$ whenever $\varphi \sim \psi$.

LEMMA 3.30. *Let $\mathscr{J} \equiv (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation, $\sim$ a congruence on $\mathscr{J}$, and φ and ψ assignments into $\mathscr{J}$ with $\varphi \sim \psi$. For every* ***SK****-combinatory formula $\boldsymbol{A}$, we have $\mathscr{E}_\varphi(\boldsymbol{A}) \sim \mathscr{E}_\psi(\boldsymbol{A})$.*

PROOF. The proof is by induction on the ***SK***-combinatory formula $\boldsymbol{A}$. If $\boldsymbol{A}$ is a variable X, we have $\mathscr{E}_\varphi(X) \equiv \varphi(X) \sim \psi(X) \equiv \mathscr{E}_\psi(X)$. If $\boldsymbol{A}$ is closed (e.g., a constant in Σ or a combinator), then $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\psi(\boldsymbol{A})$, so certainly $\mathscr{E}_\varphi(\boldsymbol{A}) \sim \mathscr{E}_\psi(\boldsymbol{A})$. Finally, if $\boldsymbol{A}$ is an application of two ***SK***-combinatory formulae $\boldsymbol{F}$ and $\boldsymbol{B}$, then by the inductive hypothesis we have $\mathscr{E}_\varphi(\boldsymbol{F}) \sim \mathscr{E}_\psi(\boldsymbol{F})$ and $\mathscr{E}_\varphi(\boldsymbol{B}) \sim \mathscr{E}_\psi(\boldsymbol{B})$. Since $\sim$ respects application, $\mathscr{E}_\varphi(\boldsymbol{FB}) \equiv \mathscr{E}_\varphi(\boldsymbol{F})@\mathscr{E}_\varphi(\boldsymbol{B}) \sim \mathscr{E}_\psi(\boldsymbol{F})@\mathscr{E}_\psi(\boldsymbol{B}) \equiv \mathscr{E}_\psi(\boldsymbol{FB})$. ⊣

We can use this result to show the same property holds for all formulae.

LEMMA 3.31. *Let $\mathscr{J} \equiv (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation, φ and ψ assignments into $\mathscr{J}$ with $\varphi \sim \psi$, and $\sim$ a congruence on $\mathscr{J}$. For every formula $\boldsymbol{A}$, we have $\mathscr{E}_\varphi(\boldsymbol{A}) \sim \mathscr{E}_\psi(\boldsymbol{A})$.*

PROOF. Let $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ for some type α. By Lemma 3.29 there is an ***SK***-combinatory formula $\boldsymbol{A}'$ that β-reduces to $\boldsymbol{A}$. By Remark 3.19 and Lemma 3.30, we have $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{A}') \sim \mathscr{E}_\psi(\boldsymbol{A}') \equiv \mathscr{E}_\psi(\boldsymbol{A})$. ⊣

REMARK 3.32 (Correspondence with logical relations). Lemma 3.31 is essentially an instance of the "Basic Lemma" for logical relations (Lemma 8.2.5 in [44]). In fact, $\sim$ is functional, iff $\sim$ is a logical relation over the applicative structure. If $\sim$ is not functional, it still satisfies this "Basic Lemma" property, which makes it a pre-logical relation in the sense of [31].

DEFINITION 3.33 (Quotient Σ-evaluation). Let $\mathscr{J} \equiv (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation, $\sim$ a congruence on $\mathscr{J}$ and let $(\mathscr{D}^\sim, @^\sim)$ be the quotient applicative structure of $(\mathscr{D}, @)$ with respect to $\sim$.

For each $\mathsf{A} \in \mathscr{D}_\alpha^\sim$, we choose a representative $\mathsf{A}^* \in \mathsf{A}$. So, $[\![\mathsf{A}^*]\!]_\sim \equiv \mathsf{A}$. Note that $[\![\mathsf{a}]\!]_\sim^* \sim \mathsf{a}$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. For any assignment φ into $\mathscr{J}/_\sim$, let φ^* be the assignment into $\mathscr{J}$ given by $\varphi^*(X) := \varphi(X)^*$. Note that $\varphi \equiv \pi_\sim \circ \varphi^*$. So we can define $\mathscr{E}_\varphi^\sim$ as $\pi_\sim \circ \mathscr{E}_{\varphi^*}$, and call $\mathscr{J}/_\sim := (\mathscr{D}^\sim, @^\sim, \mathscr{E}^\sim)$ the *quotient* Σ-*evaluation* of $\mathscr{J}$ modulo $\sim$. (By Lemma 3.31, the definition of $\mathscr{E}^\sim$ does not depend on the choice of representatives.)

This definition is justified by the following theorem.

THEOREM 3.34 (Quotient Σ-evaluation theorem). *If $\mathscr{J}$ is a Σ-evaluation and $\sim$ is a congruence on $\mathscr{J}$, then $\mathscr{J}/_\sim$ is a Σ-evaluation.*

PROOF. We prove that $\mathscr{E}^\sim$ is an evaluation function by verifying the conditions in Definition 3.18. For any assignment φ into the quotient applicative structure, let

φ^* be the assignment with $\varphi \equiv \pi_\sim \circ \varphi^*$ as in Definition 3.33. First, we compute $\mathscr{E}^\sim_\varphi\big|_{\mathscr{V}} \equiv (\pi_\sim \circ \mathscr{E}_{\varphi^*})\big|_{\mathscr{V}} \equiv \pi_\sim \circ \mathscr{E}_{\varphi^*}\big|_{\mathscr{V}} \equiv \pi_\sim \circ \varphi^* \equiv \varphi$. Since $\pi_\sim$ is a homomorphism we have

$$\begin{aligned}\mathscr{E}^\sim_\varphi(\boldsymbol{FA}) &\equiv \pi_\sim(\mathscr{E}_{\varphi^*}(\boldsymbol{FA}))\\ &\equiv \pi_\sim(\mathscr{E}_{\varphi^*}(\boldsymbol{F})@\mathscr{E}_{\varphi^*}(\boldsymbol{A}))\\ &\equiv \pi_\sim(\mathscr{E}_{\varphi^*}(\boldsymbol{F}))@^\sim\pi_\sim(\mathscr{E}_{\varphi^*}(\boldsymbol{A}))\\ &\equiv \mathscr{E}^\sim_\varphi(\boldsymbol{F})@^\sim\mathscr{E}^\sim_\varphi(\boldsymbol{A}).\end{aligned}$$

If φ and ψ coincide on $\mathrm{free}(\boldsymbol{A})$, then $\mathscr{E}^\sim_\varphi(\boldsymbol{A}) \equiv [\![\mathscr{E}_{\varphi^*}(\boldsymbol{A})]\!]_\sim \equiv [\![\mathscr{E}_{\psi^*}(\boldsymbol{A})]\!]_\sim \equiv \mathscr{E}^\sim_\psi(\boldsymbol{A})$ since this entails that φ^* and ψ^* coincide on $\mathrm{free}(\boldsymbol{A})$ too (as we have chosen particular representatives for each equivalence class). Finally, $\mathscr{E}^\sim_\varphi(\boldsymbol{A}) \equiv [\![\mathscr{E}_{\varphi^*}(\boldsymbol{A})]\!]_\sim \equiv [\![\mathscr{E}_{\varphi^*}(\boldsymbol{A}\big\downarrow_\beta)]\!]_\sim \equiv \mathscr{E}^\sim_\varphi(\boldsymbol{A}\big\downarrow_\beta)$. ⊣

DEFINITION 3.35 (Term evaluations for Σ). Let $\mathrm{cwff}(\Sigma)\big\downarrow_\beta$ be the collection of closed well-formed formulae in β-normal form and $\boldsymbol{A}@^\beta\boldsymbol{B}$ be $(\boldsymbol{AB})\big\downarrow_\beta$. For the definition of an evaluation function let φ be an assignment into $\mathrm{cwff}(\Sigma)\big\downarrow_\beta$. Note that $\sigma := \varphi\big|_{\mathrm{free}(\boldsymbol{A})}$ is a substitution, since $\mathrm{free}(\boldsymbol{A})$ is finite. Thus we can choose $\mathscr{E}^\beta_\varphi(\boldsymbol{A}) := \sigma(\boldsymbol{A})\big\downarrow_\beta$. We call $\mathscr{TE}(\Sigma)^\beta := (\mathrm{cwff}\big\downarrow_\beta, @^\beta, \mathscr{E}^\beta)$ the *β-term evaluation* for Σ.

Analogously, we can define $\mathscr{TE}(\Sigma)^{\beta\eta} := (\mathrm{cwff}\downarrow_{\beta\eta}, @^{\beta\eta}, \mathscr{E}^{\beta\eta})$ the *$\beta\eta$-term evaluation* for Σ.

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

LEMMA 3.36. *$\mathscr{TE}(\Sigma)^\beta$ is a Σ-evaluation and $\mathscr{TE}(\Sigma)^{\beta\eta}$ is a functional Σ-evaluation.*

PROOF. The fact that $(\mathrm{cwff}(\Sigma)\big\downarrow_\beta, @^\beta)$ is an applicative structure is immediate: For each type α, $\mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$ is non-empty (by the assumption in Remark 3.16) and

$$@^\beta : \mathrm{cwff}_{\alpha\to\beta}(\Sigma)\big\downarrow_\beta \times \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta \longrightarrow \mathrm{cwff}_\beta(\Sigma)\big\downarrow_\beta.$$

We next check that $\mathscr{E}^\beta$ is an evaluation function.

(1) $\mathscr{E}^\beta_\varphi(X) \equiv \varphi\big|_{\mathrm{free}(X)}(X) \equiv \varphi(X)$.

(2) $\mathscr{E}^\beta_\varphi$ respects application since $\sigma(\boldsymbol{FA})\big\downarrow_\beta \equiv (\sigma(\boldsymbol{F})\big\downarrow_\beta\sigma(\boldsymbol{A})\big\downarrow_\beta)\big\downarrow_\beta$ where $\sigma \equiv \varphi\big|_{\mathrm{free}(\boldsymbol{FA})}$.

(3) $\mathscr{E}^\beta_\varphi(\boldsymbol{A}) \equiv (\varphi\big|_{\mathrm{free}(\boldsymbol{A})}(\boldsymbol{A}))\big\downarrow_\beta \equiv (\varphi'\big|_{\mathrm{free}(\boldsymbol{A})}(\boldsymbol{A}))\big\downarrow_\beta \equiv \mathscr{E}^\beta_{\varphi'}(\boldsymbol{A})$ whenever φ and φ' coincide on $\mathrm{free}(\boldsymbol{A})$.

(4) $\mathscr{E}^\beta_\varphi(\boldsymbol{A}) \equiv \sigma(\boldsymbol{A})\big\downarrow_\beta \equiv \sigma(\boldsymbol{A}\big\downarrow_\beta)\big\downarrow_\beta \equiv \mathscr{E}^\beta_\varphi(\boldsymbol{A}\big\downarrow_\beta)$ where $\sigma \equiv \varphi\big|_{\mathrm{free}(\boldsymbol{A})}$.

A similar argument shows that $\mathscr{TE}(\Sigma)^{\beta\eta}$ is a Σ-evaluation. Also, one can show $\mathscr{TE}(\Sigma)^{\beta\eta}$ is functional using an argument similar to Lemma 3.14 since Σ is infinite at all types by Remark 3.16. (Alternatively, one can simply apply Lemma 3.14 and Theorem 3.13 to note that the applicative structure $\mathrm{cwff}(\Sigma)/_{\equiv_{\beta\eta}}$ is functional. The applicative structure $\mathrm{cwff}(\Sigma)/_{\equiv_{\beta\eta}}$ is isomorphic to the applicative structure

$(\mathrm{cwff}(\Sigma)\downarrow_{\beta\eta}, @^{\beta\eta})$. One can easily show that functionality is preserved under isomorphism.) ⊣

REMARK 3.37. Note that $\mathscr{TE}(\Sigma)^{\beta}$ is not a functional Σ-evaluation since, for instance, for any constant $h_{\gamma\to\delta} \in \Sigma$

$$(\lambda X_\gamma . h_{\gamma\to\delta} X)@^\beta \boldsymbol{C}_\gamma \equiv h@^\beta \boldsymbol{C}$$

for all $\boldsymbol{C}$ in $\mathscr{TE}_\gamma(\Sigma)^\beta$ but $\lambda X . hX \not\equiv h$.

REMARK 3.38. One can show that an evaluation function $\mathscr{E}$ for an applicative structure $(\mathscr{D}, @)$ is uniquely determined by its values $\mathscr{E}(c)$ on the constants $c \in \Sigma$ and its values $\mathscr{E}(\boldsymbol{S})$ and $\mathscr{E}(\boldsymbol{K})$ on the combinators $\boldsymbol{S}$ and $\boldsymbol{K}$. When the applicative structure is functional, even the values of each $\mathscr{E}(\boldsymbol{S})$ and $\mathscr{E}(\boldsymbol{K})$ are determined, so that $\mathscr{E}$ is uniquely determined by its values $\mathscr{E}(c)$ for $c \in \Sigma$.

DEFINITION 3.39 (Homomorphism on Σ-evaluations). Let $\mathscr{J}^1 := (\mathscr{D}^1, @^1, \mathscr{E}^1)$ and $\mathscr{J}^2 := (\mathscr{D}^2, @^2, \mathscr{E}^2)$ be Σ-evaluations. A Σ-*homomorphism* is a typed function $\kappa\colon \mathscr{D}^1 \longrightarrow \mathscr{D}^2$ such that κ is a homomorphism from the applicative structure $(\mathscr{D}^1, @^1)$ to the applicative structure $(\mathscr{D}^2, @^2)$ and $\kappa\left(\mathscr{E}^1_\varphi(\boldsymbol{A})\right) \equiv \mathscr{E}^2_{\kappa\circ\varphi}(\boldsymbol{A})$ for every $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and assignment φ for $\mathscr{J}^1$.

3.3. Σ-models. The semantic notions so far are independent of the set of base types. Now, we specialize these to obtain a notion of models by requiring specialized behavior on the type o of truth values. For this we use the notion of a Σ-valuation which gives a truth-value interpretation to the domain $\mathscr{D}_o$ of a Σ-evaluation consistent with the intuitive interpretations of the logical constants. Since models are semantic entities that are constructed primarily to make a statement about the truth or falsity of a formula, the requirement that there exists a Σ-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 Σ-evaluations that have Σ-valuations.

DEFINITION 3.40. Fix two values $\mathtt{T} \not\equiv \mathtt{F}$. Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation and $\upsilon\colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a (total) function. We define several properties that characterize logical operators with respect to υ in the table shown in Figure 2.

prop.	where	holds when			for all
$\mathfrak{L}_\neg(\mathsf{n})$	$\mathsf{n} \in \mathscr{D}_{o\to o}$	$\upsilon(\mathsf{n}@\mathsf{a}) \equiv \mathtt{T}$	iff	$\upsilon(\mathsf{a}) \equiv \mathtt{F}$	$\mathsf{a} \in \mathscr{D}_o$
$\mathfrak{L}_\vee(\mathsf{d})$	$\mathsf{d} \in \mathscr{D}_{o\to o\to o}$	$\upsilon(\mathsf{d}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$	iff	$\upsilon(\mathsf{a}) \equiv \mathtt{T}$ or $\upsilon(\mathsf{b}) \equiv \mathtt{T}$	$\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$
$\mathfrak{L}_\wedge(\mathsf{c})$	$\mathsf{c} \in \mathscr{D}_{o\to o\to o}$	$\upsilon(\mathsf{c}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$	iff	$\upsilon(\mathsf{a}) \equiv \mathtt{T}$ and $\upsilon(\mathsf{b}) \equiv \mathtt{T}$	$\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$
$\mathfrak{L}_\Rightarrow(\mathsf{i})$	$\mathsf{i} \in \mathscr{D}_{o\to o\to o}$	$\upsilon(\mathsf{i}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$	iff	$\upsilon(\mathsf{a}) \equiv \mathtt{F}$ or $\upsilon(\mathsf{b}) \equiv \mathtt{T}$	$\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$
$\mathfrak{L}_\Leftrightarrow(\mathsf{e})$	$\mathsf{e} \in \mathscr{D}_{o\to o\to o}$	$\upsilon(\mathsf{e}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$	iff	$\upsilon(\mathsf{a}) \equiv \upsilon(\mathsf{b})$	$\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$
$\mathfrak{L}^\alpha_\forall(\pi)$	$\pi \in \mathscr{D}_{(\alpha\to o)\to o}$	$\upsilon(\pi@\mathsf{f}) \equiv \mathtt{T}$	iff	$\forall \mathsf{a} \in \mathscr{D}_\alpha\ \upsilon(\mathsf{f}@\mathsf{a}) \equiv \mathtt{T}$	$\mathsf{f} \in \mathscr{D}_{\alpha\to o}$
$\mathfrak{L}^\alpha_\exists(\sigma)$	$\sigma \in \mathscr{D}_{(\alpha\to o)\to o}$	$\upsilon(\sigma@\mathsf{f}) \equiv \mathtt{T}$	iff	$\exists \mathsf{a} \in \mathscr{D}_\alpha\ \upsilon(\mathsf{f}@\mathsf{a}) \equiv \mathtt{T}$	$\mathsf{f} \in \mathscr{D}_{\alpha\to o}$
$\mathfrak{L}^\alpha_=(\mathsf{q})$	$\mathsf{q} \in \mathscr{D}_{\alpha\to\alpha\to o}$	$\upsilon(\mathsf{q}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$	iff	$\mathsf{a} \equiv \mathsf{b}$	$\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$

FIGURE 2. Logical properties in Σ-models.

DEFINITION 3.41 (Σ-model). Let $\mathscr{J} := (\mathscr{D}, @, \mathscr{E})$ be a Σ-evaluation. A function $\upsilon\colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is called a Σ-*valuation* for $\mathscr{J}$ if $\mathfrak{L}_\neg(\mathscr{E}(\neg))$ and $\mathfrak{L}_\vee(\mathscr{E}(\vee))$ hold,

and for every type α $\mathfrak{L}^{\alpha}_{\forall}(\mathscr{E}(\Pi^{\alpha}))$ holds. In this case, $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ is called a Σ-*model*.

For the case of (the optional) primitive equality, i.e., when $=^{\alpha} \in \Sigma_{\alpha\to\alpha\to o}$ for all types α, we say $\mathscr{M}$ is a Σ-*model with primitive equality* if $\mathfrak{L}^{\alpha}_{=}(\mathscr{E}(=^{\alpha}))$ holds for every type α.

We say that φ is an assignment into $\mathscr{M}$ if it is an assignment into the underlying applicative structure $(\mathscr{D}, @)$. Furthermore, φ *satisfies* a formula $\boldsymbol{A} \in \mathrm{wff}_o(\Sigma)$ in $\mathscr{M}$ (we write $\mathscr{M} \models_{\varphi} \boldsymbol{A}$) if $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{A})) \equiv \mathtt{T}$. We say that $\boldsymbol{A}$ is *valid* in $\mathscr{M}$ (and write $\mathscr{M} \models \boldsymbol{A}$) if $\mathscr{M} \models_{\varphi} \boldsymbol{A}$ for all assignments φ. When $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)$, we drop the reference to the assignment and use the notation $\mathscr{M} \models \boldsymbol{A}$. Finally, we say that $\mathscr{M}$ is a Σ-*model* for a set $\Phi \subseteq \mathrm{cwff}_o(\Sigma)$ (we write $\mathscr{M} \models \Phi$) if $\mathscr{M} \models \boldsymbol{A}$ for all $\boldsymbol{A} \in \Phi$.

A Σ-model $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ is called *functional* [*full*, *standard*] if the applicative structure $(\mathscr{D}, @)$ is *functional* [*full*, *standard*]. Similarly, $\mathscr{M}$ is called η-*functional* [ξ-*functional*] if the evaluation $(\mathscr{D}, @, \mathscr{E})$ is η-*functional* [ξ-*functional*]. We say $\mathscr{M}$ is a Σ-model over a frame if $(\mathscr{D}, @)$ is a frame.

REMARK 3.42 (Adding primitive equality). In the definition of Σ-model above, the addition of property $\mathfrak{L}^{\alpha}_{=}(\mathscr{E}(=^{\alpha}))$ addressing the case of primitive equality above has a purely practical motivation: calculi with a primitive treatment of equality, see for instance [10, 11], may provide a more effective approach to equational reasoning in higher-order logic than the exclusive use of Leibniz equality. Therefore we enrich our theory to automatically also address the situation where (always built-in) Leibniz equality and (optional) primitive equality are simultaneously present in the language. The generalization to primitive equality is less trivial than the generalization to other (optional) primitive logical connectives such as $\wedge$ or $\Rightarrow$. This is the main reason why we built primitive equality directly into our theory while we omit other logical primitives (cf. also Remarks 3.47 and 6.9).

LEMMA 3.43 (Truth and falsity in Σ-models). *Let* $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ *be a* Σ-*model and* φ *an assignment. Let* $\boldsymbol{T}_o := \forall P_o.P \vee \neg P$ *and* $\boldsymbol{F}_o := \neg \boldsymbol{T}_o$. *Then* $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{T}_o)) \equiv \mathtt{T}$ *and* $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{F}_o)) \equiv \mathtt{F}$.

PROOF. Let P be a variable of type o. We have $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{T}_o)) \equiv \mathtt{T}$, iff $\upsilon(\mathscr{E}_{\varphi}(P \vee \neg P)) \equiv \mathtt{T}$ for every assignment φ. The properties of υ show that this statement is equivalent to $\upsilon(\varphi(P)) \equiv \mathtt{T}$ or $\upsilon(\varphi(P)) \equiv \mathtt{F}$, which is always true since υ maps into $\{\mathtt{T}, \mathtt{F}\}$. Note further that $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{F}_o)) \equiv \mathtt{F}$ since $\upsilon(\mathscr{E}_{\varphi}(\boldsymbol{T}_o)) \equiv \mathtt{T}$. ⊣

REMARK 3.44. Let $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model. By Lemma 3.43, $\mathscr{D}_o$ must have at least the two elements $\mathscr{E}_{\varphi}(\boldsymbol{T}_o)$ and $\mathscr{E}_{\varphi}(\boldsymbol{F}_o)$, and υ must be surjective.

REMARK 3.45. In contrast to the case of Henkin models, Definition 3.41 only constrains the functional behavior of the values of the logical constants with respect to υ. This does not fully specify these values since

- $\mathscr{M}$ need not be functional,
- and there can be more than two truth values.

We will now introduce semantical properties called $\mathfrak{q}$, η, $\mathfrak{f}$, and $\mathfrak{b}$, which we will use to characterize different classes of Σ-models.

DEFINITION 3.46 (Properties $\mathfrak{q}$, η, ξ, $\mathfrak{f}$ and $\mathfrak{b}$). Given a Σ-model $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$, we say that $\mathscr{M}$ has *property*

$\mathfrak{q}$: iff for all $\alpha \in \mathscr{T}$ there is some $\mathsf{q}^\alpha \in \mathscr{D}_{\alpha\to\alpha\to o}$ such that $\mathfrak{L}^\alpha_=(\mathsf{q}^\alpha)$ holds.
η: iff $\mathscr{M}$ is η-functional.
ξ: iff $\mathscr{M}$ is ξ-functional.
$\mathfrak{f}$: iff $\mathscr{M}$ is functional. (This is generally associated with functional extensionality.)
$\mathfrak{b}$: iff $\mathscr{D}_o$ has at most two elements. By Lemma 3.44 we can assume without loss of generality that $\mathscr{D}_o \equiv \{\mathtt{T},\mathtt{F}\}$, υ is the identity function, $\mathscr{E}_\varphi(\boldsymbol{T}_o) \equiv \mathtt{T}$ and $\mathscr{E}_\varphi(\boldsymbol{F}_o) \equiv \mathtt{F}$. (This is generally associated with Boolean extensionality.)

REMARK 3.47 (Choice of logical constants). The work presented in this article is based on the choice of the primitive logical constants $\neg$, $\vee$, and Π^α. We have also introduced shorthand for formulas constructed using $\wedge$, $\Rightarrow$, $\Leftrightarrow$, and existential quantification. One can (easily; cf. Lemma 3.48) verify that in any Σ-model $\mathscr{M} \equiv (\mathscr{D},@,\mathscr{E},\upsilon)$, each of the properties $\mathfrak{L}_\wedge(\mathscr{E}(\lambda X_o\, Y_o.X \wedge Y))$, $\mathfrak{L}_\Rightarrow(\mathscr{E}(\lambda X_o\, Y_o.X \Rightarrow Y))$, $\mathfrak{L}_\Leftrightarrow(\mathscr{E}(\lambda X_o\, Y_o.X \Leftrightarrow Y))$ and $\mathfrak{L}^\alpha_\exists(\mathscr{E}(\lambda P_{\alpha\to o}.\exists X_\alpha.PX))$ (for each type α) hold with respect to υ. In this sense, our choice of logical constants and shorthand for other logical constants is sufficient. However, Leibniz equality $\mathbf{Q}^\alpha$ will only satisfy $\mathfrak{L}^\alpha_=(\mathscr{E}(\mathbf{Q}^\alpha))$ for each type α iff the model satisfies property $\mathfrak{q}$ (cf. Remark 3.52 and Theorem 3.63).

On the other hand, in the absence of extensionality, one can gain some (limited) expressive power by including extra logical constants such as $\wedge$ in the signature. This is the case since there may be several objects in $\mathsf{c} \in \mathscr{D}_{o\to o\to o}$ such that $\mathfrak{L}_\wedge(\mathsf{c})$ holds. So, one could have a Σ-model $\mathscr{M} \equiv (\mathscr{D},@,\mathscr{E},\upsilon)$ (where $\wedge$ is also in Σ) such that $\mathfrak{L}_\wedge(\mathscr{E}(\wedge))$ holds, but $\mathscr{E}(\wedge) \not\equiv \mathscr{E}(\lambda X_o\, Y_o.\neg(\neg X \vee \neg Y))$. We will not investigate this possibility here.

Our choice of logical constants differs from Andrews' choice [6] who considers primitive equality as the only logical primitive from which all other logical operators are defined using the definitions in Figure 3. For the sake of clarity, we write $\mathbf{q}^\alpha$ for $=^\alpha$ when $=^\alpha$ is not being written in infix notation. For Henkin models, the definitions in Figure 3 are appropriate. However, without extensionality, the situation is quite different. Suppose $\mathscr{J} \equiv (\mathscr{D},@,\mathscr{E})$ is a Σ-evaluation where $=^\alpha \in \Sigma$ for every type α. Let $\upsilon : \mathscr{D}_o \longrightarrow \{\mathtt{T},\mathtt{F}\}$ be a function such that $\mathfrak{L}^\alpha_=(\mathscr{E}(=^\alpha))$ holds for each type α. The fact that $\upsilon(\mathscr{E}(\boldsymbol{T}_o)) \equiv \mathtt{T}$ follows directly from $\mathfrak{L}^{o\to o\to o}_=(\mathscr{E}(=^{o\to o\to o}))$ and reflexivity of (meta-level) equality. Unfortunately, this is the last definition which is clearly appropriate without further assumptions. So long as $\mathscr{D}_o$ has more than one element, one can show $\upsilon(\mathscr{E}(\boldsymbol{F}_o)) \equiv \mathtt{F}$. So, let us explicitly assume $\mathscr{D}_o$

$\boldsymbol{T}_o$	$:= \mathbf{q}^o =^{o\to o\to o} \mathbf{q}^o$
$\boldsymbol{F}_o$	$:= (\lambda X_o.\boldsymbol{T}_o) =^{o\to o} (\lambda X_o.X)$
$\neg_{o\to o}$	$:= \mathbf{q}^o \boldsymbol{F}_o$
Π^α	$:= \mathbf{q}^{\alpha\to o}(\lambda X_\alpha.\boldsymbol{T}_o)$
$\wedge_{o\to o\to o}$	$:= \lambda X_o\, Y_o.(\lambda G_{o\to o\to o}.G\,\boldsymbol{T}_o\,\boldsymbol{T}_o) =^{(o\to o\to o)\to o} (\lambda G_{o\to o\to o}.GXY)$
$\Rightarrow_{o\to o\to o}$	$:= \lambda X_o\, Y_o.(X =^o (X \wedge Y))$
$\vee_{o\to o\to o}$	$:= \lambda X_o\, Y_o.\neg(\neg X \wedge \neg Y)$
Σ^α	$:= \lambda P_{\alpha\to o}.(\neg\Pi^\alpha \lambda X_\alpha.\neg(PX))$

FIGURE 3. A definition of logical constants from equality in Henkin models.

has more than one element, which is anyway met by Σ-models (cf. Remark 3.44). Next, we investigate whether $\mathfrak{L}_\neg(\mathscr{E}(\neg))$ holds. Let $\mathsf{a} \in \mathscr{D}_o$ be given. By $\mathfrak{L}^o_=(\mathscr{E}(=^o))$, we know $\upsilon(\mathscr{E}(=^o)@\mathscr{E}(\boldsymbol{F}_o)@\mathsf{a}) \equiv \mathtt{T}$ is equivalent to $\mathscr{E}(\boldsymbol{F}_o) \equiv \mathsf{a}$. So, if $\upsilon(\mathscr{E}(=^o)@\mathscr{E}(\boldsymbol{F}_o)@\mathsf{a}) \equiv \mathtt{T}$, then $\upsilon(\mathsf{a}) \equiv \upsilon(\mathscr{E}(\boldsymbol{F}_o)) \equiv \mathtt{F}$. For the converse, suppose $\upsilon(\mathsf{a}) \equiv \mathtt{F}$. This, in general, does not imply $\mathscr{E}(\boldsymbol{F}_o) \equiv \mathsf{a}$. However, if we assume a is the *unique* member of $\mathscr{D}_o$ such that $\upsilon(\mathsf{a}) \equiv \mathtt{F}$, then we can conclude $\mathscr{E}(\boldsymbol{F}_o) \equiv \mathsf{a}$. In particular, if $\mathscr{D}_o$ has only two elements, then υ must be injective and we can conclude $\mathscr{E}(\boldsymbol{F}_o) \equiv \mathsf{a}$. So, Boolean extensionality is required to ensure that $\mathfrak{L}_\neg(\mathscr{E}(\neg))$ holds for this definition of $\neg$.

We now investigate whether $\mathfrak{L}^\alpha_\forall(\mathscr{E}(\Pi^\alpha))$ holds for Π^α defined as in Figure 3. Let $\mathsf{f} \in \mathscr{D}_{\alpha\to o}$ be given. Suppose $\upsilon(\mathscr{E}(=^{\alpha\to o})@\mathscr{E}(\lambda X_\alpha.\boldsymbol{T}_o)@\mathsf{f}) \equiv \mathtt{T}$. Then, by $\mathfrak{L}^{\alpha\to o}_=(\mathscr{E}(=^{\alpha\to o}))$, we know $\mathscr{E}(\lambda X_\alpha.\boldsymbol{T}_o) \equiv \mathsf{f}$. This does guarantee $\mathscr{E}(\boldsymbol{T}_o) \equiv \mathsf{f}@\mathsf{a}$ and hence $\upsilon(\mathsf{f}@\mathsf{a}) \equiv \mathtt{T}$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. However, showing the converse requires that $\mathscr{M}$ is functional (i.e., strong functional extensionality is given). Suppose $\upsilon(\mathscr{E}(=^\alpha)@\mathscr{E}(\lambda X_\alpha.\boldsymbol{T}_o)@\mathsf{f}) \equiv \mathtt{F}$. We can conclude $\mathscr{E}(\lambda X_\alpha.\boldsymbol{T}_o) \not\equiv \mathsf{f}$, but this is of little value. If $\mathscr{J}$ is not functional, then these may be different representatives in $\mathscr{D}_{\alpha\to o}$ of the same function. If $\mathscr{J}$ is functional, there must be some $\mathsf{a} \in \mathscr{D}_\alpha$ such that $\mathscr{E}(\boldsymbol{T}_o) \not\equiv \mathsf{f}@\mathsf{a}$. However, this still does not imply $\upsilon(\mathsf{f}@\mathsf{a}) \equiv \mathtt{F}$. If $\mathscr{D}_o$ has only two elements, then the facts that $\mathscr{E}(\boldsymbol{T}_o) \not\equiv \mathsf{f}@\mathsf{a}$ and $\mathscr{E}(\boldsymbol{T}_o) \not\equiv \mathscr{E}(\boldsymbol{F}_o)$ imply $\mathscr{E}(\boldsymbol{F}_o) \equiv \mathsf{f}@\mathsf{a}$, hence $\upsilon(\mathsf{f}@\mathsf{a}) \equiv \mathtt{F}$.

Similar observations apply to the other definitions in Figure 3. These definitions do show that at least $\boldsymbol{T}_o$ and $\boldsymbol{F}_o$ are definable from primitive equality (so long as $\mathscr{D}_o$ has at least two elements). Furthermore, if $\mathscr{D}_o$ has exactly two elements $\neg$ is definable from primitive equality. We conjecture that this is as much as one can define in terms of primitive equality without extensionality assumptions. That is, we conjecture that without assuming $\mathscr{D}_o$ has two elements, there may be no object $\mathsf{n} \in \mathscr{D}_{o\to o}$ such that $\mathfrak{L}_\neg(\mathsf{n})$ holds. Furthermore, we conjecture that without assuming functionality and that $\mathscr{D}_o$ has two elements, there may be no object $\mathsf{d} \in \mathscr{D}_{o\to o\to o}$ such that $\mathfrak{L}_\vee(\mathsf{d})$ holds, and there may be no object $\pi \in \mathscr{D}_{(\alpha\to o)\to o}$ such that $\mathfrak{L}^\alpha_\forall(\pi)$ holds.

The next lemma formally verifies that $\mathfrak{L}_\Leftrightarrow(\mathscr{E}(\lambda X_o\, Y_o.X \Leftrightarrow Y))$ holds with respect to the valuation of a Σ-model, as indicated in the remark above.

LEMMA 3.48 (Equivalence). *Let* $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ *be a* Σ*-model,* φ *an assignment into* $\mathscr{M}$*, and* $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_o(\Sigma)$. $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \Leftrightarrow \boldsymbol{B})) \equiv \mathtt{T}$ *iff* $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B}))$.

PROOF. Suppose $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \Leftrightarrow \boldsymbol{B})) \equiv \mathtt{T}$. This implies $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{A} \vee \boldsymbol{B})) \equiv \mathtt{T}$ and $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{B} \vee \boldsymbol{A})) \equiv \mathtt{T}$. If $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \mathtt{T}$, then $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{A} \vee \boldsymbol{B})) \equiv \mathtt{T}$ implies $\upsilon(\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{T}$, so $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \mathtt{T} \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B}))$. If $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \mathtt{F}$, then $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{B} \vee \boldsymbol{A})) \equiv \mathtt{T}$ implies $\upsilon(\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{F}$, so $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \mathtt{F} \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B}))$. Since these are the only two possible values for $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A}))$, we have $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B}))$.

Suppose $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B}))$. Either $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{T}$ or $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{F}$. An easy consideration of both cases verifies $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{A} \vee \boldsymbol{B})) \equiv \mathtt{T}$ and $\upsilon(\mathscr{E}_\varphi(\neg\boldsymbol{B} \vee \boldsymbol{A})) \equiv \mathtt{T}$. Hence, $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \Leftrightarrow \boldsymbol{B})) \equiv \mathtt{T}$. ⊣

We next define classes of Σ-models in which certain properties hold. These classes are denoted by $\mathfrak{M}_*$ where $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. The subscript β is always included to emphasize that β-equal terms are interpreted to be identical elements in all models (cf. Remark 3.19). The subscripts η, ξ, $\mathfrak{f}$ and $\mathfrak{b}$ indicate when the corresponding properties must hold (cf. Definition 3.46). Note that we are not including property $\mathfrak{q}$ as an explicit subscript. The only Σ-models we need to consider

which do not satisfy property $\mathfrak{q}$ are term models. It will turn out (cf. Theorem 3.62) that we can obtain a model satisfying property $\mathfrak{q}$ from a model that does not by taking a quotient. However, this may not preserve properties ξ or $\mathfrak{f}$. Consequently, we omit $\mathfrak{q}$ as a subscript and define the sets $\mathfrak{M}_*$ (for $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$) so that every model in $\mathfrak{M}_*$ satisfies property $\mathfrak{q}$. (This choice will be discussed further in Remark 3.52.)

DEFINITION 3.49 (Higher-order model classes). We will denote the class of Σ-models that satisfy property $\mathfrak{q}$ by $\mathfrak{M}_\beta$, and we will use subclasses of $\mathfrak{M}_\beta$ depending on the validity of the properties η, ξ, $\mathfrak{f}$, and $\mathfrak{b}$. We obtain the specialized classes of Σ-models $\mathfrak{M}_{\beta\eta}$, $\mathfrak{M}_{\beta\xi}$, $\mathfrak{M}_{\beta\mathfrak{f}}$, $\mathfrak{M}_{\beta\mathfrak{b}}$, $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, $\mathfrak{M}_{\beta\xi\mathfrak{b}}$, and $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ by requiring that the properties specified in the index are valid.

If primitive equality is in the signature, i.e., if $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$, then we require the models to be Σ-models with primitive equality. Note that in this case property $\mathfrak{q}$ is automatically ensured.

We can group these eight classes in two dimensions as in Figure 4 based on the "amount of extensionality" required.

		functional			
		none	weak (η)	weak (ξ)	strong ($\mathfrak{f}$)
Boolean	none	$\mathfrak{M}_\beta$	$\mathfrak{M}_{\beta\eta}$	$\mathfrak{M}_{\beta\xi}$	$\mathfrak{M}_{\beta\mathfrak{f}}$
	$\mathfrak{b}$	$\mathfrak{M}_{\beta\mathfrak{b}}$	$\mathfrak{M}_{\beta\eta\mathfrak{b}}$	$\mathfrak{M}_{\beta\xi\mathfrak{b}}$	$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$

FIGURE 4. Extensional model classes.

DEFINITION 3.50 (Σ-Henkin models). A Σ-*Henkin model* is a model $\mathscr{M}$ over a frame with $\mathscr{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$. We denote the class of all Σ-Henkin models by $\mathfrak{H}$. (Such models are called *general models* in [2] and [6]. We avoid this terminology here since we consider models which are more general than these.)

DEFINITION 3.51 (Σ-standard models). A Σ-*standard model* is a Σ-Henkin model that is also full (i.e., a model $\mathscr{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ over a standard frame). The class of all Σ-standard models is denoted by $\mathfrak{ST}$.

REMARK 3.52 (Property $\mathfrak{q}$). The purpose of property $\mathfrak{q}$ is to ensure that for all types α there is an object q^α in $\mathscr{D}_{\alpha\to\alpha\to o}$ representing meta equality for the domain $\mathscr{D}_\alpha$. This ensures the existence of objects representing unit sets $\{\mathsf{a}\}$ for each $\mathsf{a} \in \mathscr{D}_\alpha$ in the domains $\mathscr{D}_{\alpha\to o}$, which in turn makes Leibniz equality the intended equality relation. This is because membership in these unit sets can be used as an appropriately strong criterion to distinguish between different elements of $\mathscr{D}_\alpha$. This aspect is discussed in detail by Peter Andrews in [2]. He notes that Leon Henkin unintentionally introduced in [26] a class of models which need not satisfy property $\mathfrak{q}$ instead of the class of Henkin models in the sense above. As Andrews shows, a consequence is that such a model may fail to satisfy the principle of strong functional extensionality (cf. Definition 4.5) given by the formula

$$\forall F_{\iota\to\iota}.\forall G_{\iota\to\iota}.(\forall X_\iota.FX \doteq^\iota GX) \Rightarrow F \doteq^{\iota\to\iota} G$$

even though the model (as a model over a frame) is functional. Andrews fixed this problem by introducing property $\mathfrak{q}$. Here, we have followed this by requiring property $\mathfrak{q}$ in all our model classes $\mathfrak{M}_*$.

Now let us extend the notion of a quotient evaluation to Σ-models.

DEFINITION 3.53 (Σ-model congruences). A *congruence* on a Σ-model $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ is a congruence on the underlying Σ-evaluation $(\mathscr{D}, @, \mathscr{E})$ such that $\upsilon(\mathsf{a}) \equiv \upsilon(\mathsf{b})$ for all $\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$ with $\mathsf{a} \sim \mathsf{b}$.

DEFINITION 3.54 (Quotient Σ-model). Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model, $\sim$ be a congruence on $\mathscr{M}$, and $(\mathscr{D}^{\sim}, @^{\sim}, \mathscr{E}^{\sim})$ be the quotient Σ-evaluation of $(\mathscr{D}, @, \mathscr{E})$ with respect to $\sim$ (cf. Definition 3.33). Using the notation for representatives $\mathsf{A}^* \in \mathsf{A}$ for $\mathsf{A} \in \mathscr{D}^{\sim}_{\alpha}$ as in Definition 3.33, we define $\upsilon^{\sim} : \mathscr{D}^{\sim}_o \longrightarrow \{\mathsf{T}, \mathsf{F}\}$ by $\upsilon^{\sim}(\mathsf{A}) := \upsilon(\mathsf{A}^*)$ for every $\mathsf{A} \in \mathscr{D}^{\sim}_o$. (Since $\upsilon(\mathsf{a}) \equiv \upsilon(\mathsf{b})$ whenever $\mathsf{a} \sim \mathsf{b}$ in $\mathscr{D}_o$, this definition of $\upsilon^{\sim}$ does not depend on the choice of representatives and $\upsilon^{\sim}(\llbracket \mathsf{a} \rrbracket_{\sim}) \equiv \upsilon(\mathsf{a})$ for every $\mathsf{a} \in \mathscr{D}_o$.) We call $\mathscr{M}/_{\sim} := (\mathscr{D}^{\sim}, @^{\sim}, \mathscr{E}^{\sim}, \upsilon^{\sim})$ the *quotient* Σ-model of $\mathscr{M}$ with respect to $\sim$.

THEOREM 3.55 (Quotient Σ-model theorem). *Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model and $\sim$ be a congruence on $\mathscr{M}$. The quotient $\mathscr{M}/_{\sim}$ is a Σ-model.*

Furthermore, if for every type α, $=^{\alpha} \in \Sigma_{\alpha}$ and we have $\upsilon(\mathscr{E}(=^{\alpha})@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$ iff $\mathsf{a} \sim \mathsf{b}$ for every $\mathsf{a}, \mathsf{b} \in \mathscr{D}_{\alpha}$, then $\mathscr{M}/_{\sim}$ is a Σ-model with primitive equality.

PROOF. We check the conditions of Definition 3.41, again using the A^* notation for representatives. To check condition $\mathfrak{L}_{\neg}(\mathscr{E}^{\sim}(\neg))$ for $\upsilon^{\sim}$, for all $\mathsf{A} \in \mathscr{D}^{\sim}_o$ we need to show that $\upsilon^{\sim}(\mathscr{E}^{\sim}(\neg)@^{\sim}\mathsf{A}) \equiv \mathsf{T}$ iff $\upsilon^{\sim}(\mathsf{A}) \equiv \mathsf{F}$. Let $\mathsf{A} \in \mathscr{D}^{\sim}_o$ be given. Since $\mathscr{M}$ is a Σ-model we have $\upsilon(\mathscr{E}(\neg)@\mathsf{A}^*) \equiv \mathsf{T}$ iff $\upsilon(\mathsf{A}^*) \equiv \mathsf{F}$. Since $\llbracket \mathsf{A}^* \rrbracket_{\sim} \equiv \mathsf{A}$ and $\llbracket \mathscr{E}(\neg)@\mathsf{A}^* \rrbracket_{\sim} \equiv \mathscr{E}^{\sim}(\neg)@^{\sim}\mathsf{A}$, we have $\upsilon^{\sim}(\mathscr{E}^{\sim}(\neg)@^{\sim}\mathsf{A}) \equiv \mathsf{T}$ iff $\upsilon^{\sim}(\mathsf{A}) \equiv \mathsf{F}$. Checking condition $\mathfrak{L}_{\vee}(\mathscr{E}^{\sim}(\vee))$ for $\upsilon^{\sim}$ is analogous.

To check condition $\mathfrak{L}^{\alpha}_{\forall}(\mathscr{E}^{\sim}(\Pi^{\alpha}))$ for $\upsilon^{\sim}$, suppose we have $\mathsf{G} \in \mathscr{D}^{\sim}_{\alpha \to o}$. For every $\mathsf{A} \in \mathscr{D}^{\sim}_{\alpha}$, $\upsilon^{\sim}(\mathsf{G}@^{\sim}\mathsf{A}) \equiv \upsilon(\mathsf{G}^*@\mathsf{A}^*)$. So, if $\upsilon^{\sim}(\mathsf{G}@^{\sim}\mathsf{A}) \equiv \mathsf{T}$ for every $\mathsf{A} \in \mathscr{D}^{\sim}_{\alpha}$, then $\upsilon(\mathsf{G}^*@\mathsf{a}) \equiv \upsilon(\mathsf{G}^*@\llbracket \mathsf{a} \rrbracket^*_{\sim}) \equiv \mathsf{T}$ for every $\mathsf{a} \in \mathscr{D}_{\alpha}$, and we conclude $\upsilon(\mathscr{E}(\Pi^{\alpha})@\mathsf{G}^*) \equiv \mathsf{T}$. Hence, $\upsilon^{\sim}(\mathscr{E}^{\sim}(\Pi^{\alpha})@^{\sim}\mathsf{G}) \equiv \mathsf{T}$. Conversely, suppose $\upsilon^{\sim}(\mathscr{E}^{\sim}(\Pi^{\alpha})@\mathsf{G}) \equiv \mathsf{T}$. Then $\upsilon(\mathscr{E}(\Pi^{\alpha})@\mathsf{G}^*) \equiv \mathsf{T}$ and hence $\upsilon^{\sim}(\mathsf{G}@\mathsf{A}) \equiv \upsilon(\mathsf{G}^*@\mathsf{A}^*) \equiv \mathsf{T}$ for every $\mathsf{A} \in \mathscr{D}^{\sim}_{\alpha}$.

Suppose primitive equality is in the signature and $\upsilon(\mathscr{E}(=^{\alpha})@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$ iff $\mathsf{a} \sim \mathsf{b}$ for every $\mathsf{a}, \mathsf{b} \in \mathscr{D}_{\alpha}$. To verify $\mathfrak{L}^{\alpha}_{=}(\mathscr{E}^{\sim}(=^{\alpha}))$ holds for $\upsilon^{\sim}$, we simply note that $\upsilon^{\sim}(\mathscr{E}^{\sim}(=^{\alpha})@^{\sim}\mathsf{A}@^{\sim}\mathsf{B}) \equiv \mathsf{T}$, iff $\upsilon(\mathscr{E}(=^{\alpha})@\mathsf{A}^*@\mathsf{B}^*) \equiv \mathsf{T}$, iff $\mathsf{A}^* \sim \mathsf{B}^*$, iff $\mathsf{A} \equiv \mathsf{B}$. ⊣

We can define properties of a congruence analogous to those defined for models in Definition 3.46.

DEFINITION 3.56 (Properties η, ξ, $\mathfrak{f}$ and $\mathfrak{b}$ for congruences). Given a Σ-model $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ and a congruence $\sim$ on $\mathscr{M}$, we say $\sim$ has *property*

η: iff $\mathscr{E}_{\varphi}(\boldsymbol{A}) \sim \mathscr{E}_{\varphi}(\boldsymbol{A}\downarrow_{\beta\eta})$ for any type α, $\boldsymbol{A} \in \mathrm{wff}_{\alpha}(\Sigma)$, and assignment φ.

ξ: iff for all $\alpha, \beta \in \mathscr{T}$, $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_{\beta}(\Sigma)$, assignment φ, and variables X_{α}, $\mathscr{E}_{\varphi}(\lambda X_{\alpha}.\boldsymbol{M}_{\beta}) \sim \mathscr{E}_{\varphi}(\lambda X_{\alpha}.\boldsymbol{N}_{\beta})$ whenever $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \sim \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$ for every $\mathsf{a} \in \mathscr{D}_{\alpha}$.

$\mathfrak{f}$: iff $\sim$ is functional.

$\mathfrak{b}$: iff $\mathscr{D}_o$ has at most two equivalence classes with respect to $\sim$. (By Remark 3.44 there are always at least two.)

REMARK 3.57. It follows trivially from reflexivity of congruences that if a model satisfies property η, then any congruence on the model satisfies property η. Similarly, if a model has only two elements in $\mathscr{D}_o$, then $\mathscr{D}_o$ can have at most two equivalence classes with respect to any congruence $\sim$. So, if a model satisfies property $\mathfrak{b}$, then any congruence on the model satisfies property $\mathfrak{b}$. This is not true for properties ξ or $\mathfrak{f}$. For an example, we refer to the functional model (satisfying property $\mathfrak{f}$, hence property ξ) constructed by Andrews in [2]. Using the results we prove below, one can show Leibniz equality must induce a congruence failing to satisfy properties ξ and $\mathfrak{f}$ on this functional model.

LEMMA 3.58. *Let $\mathscr{M}$ be a Σ-model, $\Phi \subseteq \mathrm{cwff}_o(\Sigma)$, and $\sim$ be a congruence on $\mathscr{M}$. We have $\mathscr{M}/_\sim \models \Phi$ iff $\mathscr{M} \models \Phi$. Furthermore, if $* \in \{\eta, \xi, \mathfrak{f}, \mathfrak{b}\}$ and $\sim$ satisfies property $*$, then $\mathscr{M}/_\sim$ satisfies property $*$.*

PROOF. Let $\boldsymbol{A}_o \in \Phi$. Since $\boldsymbol{A}$ is closed, $\mathscr{M} \models \boldsymbol{A}$, iff $v(\mathscr{E}(\boldsymbol{A})) \equiv \mathsf{T}$, iff $v^\sim(\mathscr{E}^\sim(\boldsymbol{A})) \equiv \mathsf{T}$, iff $\mathscr{M}/_\sim \models \boldsymbol{A}$. So, $\mathscr{M} \models \Phi$ iff $\mathscr{M}/_\sim \models \Phi$.

Suppose $\sim$ satisfies property η. Let $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, and an assignment φ into $\mathscr{M}/_\sim$ be given. Let φ^* be a corresponding assignment into $\mathscr{M}$ (cf. Definition 3.33). Since $\sim$ satisfies property η, we know $\mathscr{E}_{\varphi^*}(\boldsymbol{A}) \sim \mathscr{E}_{\varphi^*}(\boldsymbol{A}\downarrow_{\beta\eta})$. Taking equivalence classes, we have $\mathscr{E}^\sim_\varphi(\boldsymbol{A}) \equiv \mathscr{E}^\sim_\varphi(\boldsymbol{A}\downarrow_{\beta\eta})$.

Suppose $\sim$ satisfies property ξ. Let $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_\beta(\Sigma)$, a variable X_α and an assignment φ into $\mathscr{M}/_\sim$ be given. Again, let φ^* be a corresponding assignment into $\mathscr{M}$. Suppose $\mathscr{E}^\sim_{\varphi,[\mathsf{A}/X]}(\boldsymbol{M}) \equiv \mathscr{E}^\sim_{\varphi,[\mathsf{A}/X]}(\boldsymbol{N})$ for every $\mathsf{A} \in \mathscr{D}^\sim_\alpha$. This means $\mathscr{E}_{\varphi^*,[\mathsf{A}^*/X]}(\boldsymbol{M}) \sim \mathscr{E}_{\varphi^*,[\mathsf{A}^*/X]}(\boldsymbol{N})$ for every $\mathsf{A} \in \mathscr{D}^\sim_\alpha$. For any $\mathsf{a} \in \mathscr{D}_\alpha$, using Lemma 3.31, we know

$$\mathscr{E}_{\varphi^*,[\mathsf{a}/X]}(\boldsymbol{M}) \sim \mathscr{E}_{\varphi^*,[\mathsf{A}^*/X]}(\boldsymbol{M}) \sim \mathscr{E}_{\varphi^*,[\mathsf{A}^*/X]}(\boldsymbol{N}) \sim \mathscr{E}_{\varphi^*,[\mathsf{a}/X]}(\boldsymbol{N})$$

where $\mathsf{A} \in \mathscr{D}^\sim_\alpha$ is the equivalence class of a. Since $\sim$ satisfies property ξ, we know that $\mathscr{E}_{\varphi^*}(\lambda X_\alpha.\boldsymbol{M}) \sim \mathscr{E}_{\varphi^*}(\lambda X_\alpha.\boldsymbol{N})$. Taking equivalence classes, we see that $\mathscr{E}^\sim_\varphi(\lambda X_\alpha.\boldsymbol{M}) \equiv \mathscr{E}^\sim_\varphi(\lambda X_\alpha.\boldsymbol{N})$.

If $\sim$ is functional (satisfies property $\mathfrak{f}$), we know $\mathscr{M}/_\sim$ is functional (satisfies property $\mathfrak{f}$) by Theorem 3.13.

Finally, if $\sim$ satisfies property $\mathfrak{b}$, then clearly $\mathscr{D}^\sim_o$ has only two elements. So, $\mathscr{M}/_\sim$ satisfies property $\mathfrak{b}$. ⊣

DEFINITION 3.59 (Congruence relation $\dot\sim$). Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, v)$ be a Σ-model. Let $\mathsf{q}^\alpha \in \mathscr{D}_{\alpha\to\alpha\to o}$ be $\mathscr{E}(\mathbf{Q}^\alpha)$, i.e., the interpretation of Leibniz equality at type α. We define $\mathsf{a} \dot\sim \mathsf{b}$ in $\mathscr{D}_\alpha$ iff $v(\mathsf{q}^\alpha @ \mathsf{a} @ \mathsf{b}) \equiv \mathsf{T}$.

Before checking $\dot\sim$ is a congruence, we first show that it is at least reflexive.

LEMMA 3.60. *Let $\mathscr{M}$ be a Σ-model. For each type α and $\mathsf{a} \in \mathscr{D}_\alpha$, we have $\mathsf{a} \dot\sim \mathsf{a}$.*

PROOF. We need to check $v(\mathscr{E}(\mathbf{Q}^\alpha)@\mathsf{a}@\mathsf{a}) \equiv \mathsf{T}$. Let X_α be a variable of type α and φ be some assignment with $\varphi(X) \equiv \mathsf{a}$. Let $\mathsf{r} := \mathscr{E}_\varphi(\lambda P_{\alpha\to o}.\neg(PX) \vee PX))$. For any $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$, since $\mathscr{E}$ is an evaluation function, we have

$$v(\mathsf{r}@\mathsf{p}) \equiv v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(\neg(PX) \vee PX)).$$

As $\mathscr{M}$ is a Σ-model, we have $v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(\neg(PX) \vee PX)) \equiv \mathsf{T}$ since either

$$v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(PX)) \equiv \mathsf{T} \quad \text{or} \quad v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(\neg(PX))) \equiv \mathsf{T}.$$

So, again since $\mathscr{M}$ is a Σ-model, $v(\mathscr{E}(\Pi^{\alpha\to o})@\mathsf{r}) \equiv \mathtt{T}$. By the definitions of r and $\doteq^{\alpha}$, we have $v(\mathscr{E}_{\varphi}(X \doteq^{\alpha} X)) \equiv \mathtt{T}$. As $X \doteq^{\alpha} X$ is a β-reduct of $\mathbf{Q}^{\alpha}XX$, we have $v(\mathscr{E}_{\varphi}(\mathbf{Q}^{\alpha}XX)) \equiv \mathtt{T}$ as well. Using $\varphi(X) \equiv \mathsf{a}$, we see that $v(\mathscr{E}(\mathbf{Q}^{\alpha})@\mathsf{a}@\mathsf{a}) \equiv \mathtt{T}$. ⊣

In order to check that $\dot{\sim}$ is a congruence, it is useful to unwind the definitions to better characterize when $\mathsf{a} \dot{\sim} \mathsf{b}$ for $\mathsf{a}, \mathsf{b} \in \mathscr{D}_{\alpha}$.

LEMMA 3.61 (Properties of $\dot{\sim}$). *Let $\mathscr{M}$ be a Σ-model. For each type α and $\mathsf{a}, \mathsf{b} \in \mathscr{D}_{\alpha}$, the following are equivalent*:

(1) $\mathsf{a} \dot{\sim} \mathsf{b}$.
(2) *For all variables X_{α} and Y_{α} and assignments φ such that $\varphi(X) \equiv \mathsf{a}$ and $\varphi(Y) \equiv \mathsf{b}$, we have $v(\mathscr{E}_{\varphi}(X \doteq^{\alpha} Y)) \equiv \mathtt{T}$.*
(3) *For every $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$, $v(\mathsf{p}@\mathsf{a}) \equiv \mathtt{T}$ implies $v(\mathsf{p}@\mathsf{b}) \equiv \mathtt{T}$.*
(4) *For every $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$, $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b})$.*

PROOF. At each type α, let $\mathsf{q}^{\alpha} \in \mathscr{D}_{\alpha\to\alpha\to o}$ be the interpretation $\mathscr{E}(\mathbf{Q}^{\alpha})$ of Leibniz equality. By definition, $\mathsf{a} \dot{\sim} \mathsf{b}$ iff $v(\mathsf{q}^{\alpha}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$.

To show (1) implies (2), suppose $\mathsf{a} \dot{\sim} \mathsf{b}$ and φ is an assignment with $\varphi(X_{\alpha}) \equiv \mathsf{a}$ and $\varphi(Y_{\alpha}) \equiv \mathsf{b}$. Since $v(\mathsf{q}^{\alpha}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$, we have $v(\mathscr{E}_{\varphi}(\mathbf{Q}^{\alpha}XY)) \equiv \mathtt{T}$. Since $\mathscr{E}$ respects β-equality (cf. Remark 3.19), we have $v(\mathscr{E}_{\varphi}(X \doteq^{\alpha} Y)) \equiv \mathtt{T}$.

To show (2) implies (3), suppose $v(\mathscr{E}_{\varphi}(X \doteq^{\alpha} Y)) \equiv \mathtt{T}$ whenever φ is an assignment with $\varphi(X) \equiv \mathsf{a}$ and $\varphi(Y) \equiv \mathsf{b}$. Let X and Y be particular distinct variables of type α and φ be any such assignment with $\varphi(X) \equiv \mathsf{a}$ and $\varphi(Y) \equiv \mathsf{b}$. Let $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$ with $v(\mathsf{p}@\mathsf{a}) \equiv \mathtt{T}$ and a variable $P_{\alpha\to o}$ be given. By assumption, $v(\mathscr{E}_{\varphi}(\forall P_{\alpha\to o}.\neg(PX) \vee (PY))) \equiv \mathtt{T}$. Since $v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(PX)) \equiv v(\mathsf{p}@\mathsf{a}) \equiv \mathtt{T}$, we have $v(\mathsf{p}@\mathsf{b}) \equiv v(\mathscr{E}_{\varphi,[\mathsf{p}/P]}(PY)) \equiv \mathtt{T}$.

To show (3) implies (4), let $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$ be given. If $v(\mathsf{p}@\mathsf{a}) \equiv \mathtt{T}$, then we have $v(\mathsf{p}@\mathsf{b}) \equiv \mathtt{T}$ by assumption. So, $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b})$ in this case. Otherwise, we must have $v(\mathsf{p}@\mathsf{a}) \equiv \mathtt{F}$. Let $\mathsf{q} := \mathscr{E}_{\varphi}(\lambda X_{\alpha}.\neg(P_{\alpha\to o}X))$ where φ is some assignment with $\varphi(P) := \mathsf{p}$. Since $\mathscr{M}$ is a model, $v(\mathsf{q}@\mathsf{a}) \equiv v(\mathscr{E}(\neg)@(\mathsf{p}@\mathsf{a})) \equiv \mathtt{T}$. Applying the assumption to q, we have $v(\mathsf{q}@\mathsf{b}) \equiv \mathtt{T}$ and so $v(\mathscr{E}(\neg)@(\mathsf{p}@\mathsf{b})) \equiv \mathtt{T}$. Thus, $v(\mathsf{p}@\mathsf{b}) \equiv \mathtt{F}$ and $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b})$ in this case as well.

To show (4) implies (1), suppose $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b})$ for every $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$. In particular, this holds for $\mathsf{p} := \mathsf{q}^{\alpha}@\mathsf{a} \in \mathscr{D}_{\alpha\to o}$. Since $v(\mathsf{q}^{\alpha}@\mathsf{a}@\mathsf{a}) \equiv \mathtt{T}$ by Lemma 3.60, we must have $v(\mathsf{q}^{\alpha}@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$. That is, $\mathsf{a} \dot{\sim} \mathsf{b}$. ⊣

THEOREM 3.62 (Properties of $\mathscr{M}/_{\dot{\sim}}$). *Let $\mathscr{M}$ be a Σ-model. Then $\dot{\sim}$ is a congruence relation on the model $\mathscr{M}$ and $\mathscr{M}/_{\dot{\sim}}$ satisfies property q. Furthermore, if for every type α, $=^{\alpha} \in \Sigma_{\alpha}$ and $v(\mathscr{E}(=^{\alpha})@a@b) \equiv \mathtt{T}$ iff $a \dot{\sim} b$ for all $\mathsf{a}, \mathsf{b} \in \mathscr{D}_{\alpha}$, then $\mathscr{M}/_{\dot{\sim}}$ is a Σ-model with primitive equality.*

PROOF. We first verify that $\dot{\sim}$ is an equivalence relation on each $\mathscr{D}_{\alpha}$. Reflexivity was shown in Lemma 3.60. To check symmetry and transitivity we use condition (4) in Lemma 3.61. For symmetry, let $\mathsf{a} \dot{\sim} \mathsf{b}$ in $\mathscr{D}_{\alpha}$ and $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$ be given. So, $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b})$. Generalizing over p, we have $\mathsf{b} \dot{\sim} \mathsf{a}$. For transitivity, let $\mathsf{a} \dot{\sim} \mathsf{b}$ and $\mathsf{b} \dot{\sim} \mathsf{c}$ in $\mathscr{D}_{\alpha}$ and $\mathsf{p} \in \mathscr{D}_{\alpha\to o}$ be given. So, $v(\mathsf{p}@\mathsf{a}) \equiv v(\mathsf{p}@\mathsf{b}) \equiv v(\mathsf{p}@\mathsf{c})$. Generalizing over p, we have $\mathsf{a} \dot{\sim} \mathsf{c}$.

We next verify that $\dot{\sim}$ is a congruence. Suppose $\mathsf{f} \dot{\sim} \mathsf{g}$ in $\mathscr{D}_{\alpha\to\beta}$ and $\mathsf{a} \dot{\sim} \mathsf{b} \in \mathscr{D}_{\alpha}$. To show $\mathsf{f}@\mathsf{a} \dot{\sim} \mathsf{g}@\mathsf{b}$ we use condition (3) in Lemma 3.61. Let $\mathsf{p} \in \mathscr{D}_{\beta\to o}$ with $v(\mathsf{p}@(\mathsf{f}@\mathsf{a})) \equiv \mathtt{T}$ be given. Let φ be an assignment with $\varphi(P_{\beta\to o}) \equiv \mathsf{p}$, $\varphi(X_{\alpha}) \equiv \mathsf{a}$

and $\varphi(G_{\alpha\to\beta}) \equiv \mathsf{g}$ for variables P, X and G. We can use Lemma 3.61(3) with $\mathscr{E}_\varphi(\lambda F_{\alpha\to\beta}.(P(FX)))$ and $\mathsf{f} \dot{\sim} \mathsf{g}$ to verify that $\upsilon(\mathsf{p}@(\mathsf{g}@\mathsf{a})) \equiv \mathsf{T}$. Using Lemma 3.61(3) with $\mathscr{E}_\varphi(\lambda X_\alpha.(P(GX)))$ and $\mathsf{a} \dot{\sim} \mathsf{b}$ verifies $\upsilon(\mathsf{p}@(\mathsf{g}@\mathsf{b})) \equiv \mathsf{T}$. So, $\mathsf{f}@\mathsf{a} \dot{\sim} \mathsf{g}@\mathsf{b}$.

It remains to check that $\upsilon(\mathsf{a}) \equiv \upsilon(\mathsf{b})$ whenever $\mathsf{a} \dot{\sim} \mathsf{b}$ for $\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$. Let $\mathsf{a} \dot{\sim} \mathsf{b}$ in $\mathscr{D}_o$ be given. Applying Lemma 3.61(4) to $\mathscr{E}(\lambda X_o.X) \in \mathscr{D}_{o\to o}$ we have $\upsilon(\mathsf{a}) \equiv \upsilon(\mathscr{E}(\lambda X_o.X)@\mathsf{a}) \equiv \upsilon(\mathscr{E}(\lambda X_o.X)@\mathsf{b}) \equiv \upsilon(\mathsf{b})$ as desired. So, $\dot{\sim}$ is a congruence relation on $\mathscr{M}$.

Now, we show $\mathscr{M}/_{\dot{\sim}}$ satisfies property q. At each type α, let $\mathsf{q}^\alpha \in \mathscr{D}_{\alpha\to\alpha\to o}$ be the interpretation $\mathscr{E}(\mathbf{Q}^\alpha)$ of Leibniz equality. To check property q, we show that $[\![\mathsf{q}^\alpha]\!]_{\dot{\sim}}$ is the appropriate object in $\mathscr{D}^{\dot{\sim}}_{\alpha\to\alpha\to o}$ for each $\alpha \in \mathscr{T}$. Let $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$ be given. Note that $[\![\mathsf{a}]\!]_{\dot{\sim}} \equiv [\![\mathsf{b}]\!]_{\dot{\sim}}$ is equivalent to $\mathsf{a} \dot{\sim} \mathsf{b}$.

Also, $\upsilon^{\dot{\sim}}([\![\mathsf{q}^\alpha]\!]_{\dot{\sim}}@^{\dot{\sim}}[\![\mathsf{a}]\!]_{\dot{\sim}}@^{\dot{\sim}}[\![\mathsf{b}]\!]_{\dot{\sim}}) \equiv \mathsf{T}$ is equivalent to $\upsilon(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$. So, we need to show that $\upsilon(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$ if and only if $\mathsf{a} \dot{\sim} \mathsf{b}$. But this is precisely the definition of $\dot{\sim}$.

The statement for primitive equality follows immediately by Theorem 3.55. ⊣

Now, we know that when one takes a quotient of a model $\mathscr{M}$ by $\dot{\sim}$, one obtains a model satisfying property q. It is worthwhile to note the following relationship between $\dot{\sim}$ and property q.

THEOREM 3.63. *Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model. The following are equivalent*:

(1) *$\mathscr{M}$ satisfies property q.*
(2) *For any congruence $\sim$ on $\mathscr{M}$, type α, and $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$, $\mathsf{a} \sim \mathsf{b}$ implies $\mathsf{a} \equiv \mathsf{b}$.*
(3) *For any type α, and $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$, $\mathsf{a} \dot{\sim} \mathsf{b}$ implies $\mathsf{a} \equiv \mathsf{b}$.*
(4) *For any type α, $\mathfrak{L}^\alpha_=(\mathscr{E}(\mathbf{Q}^\alpha))$ holds for υ.*

PROOF. To show (1) implies (2), suppose $\mathscr{M}$ satisfies q, $\sim$ is a congruence on $\mathscr{M}$, and $\mathsf{a} \sim \mathsf{b}$ for $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$. Let $\mathsf{q}^\alpha \in \mathscr{D}_{\alpha\to\alpha\to o}$ be the object at type α guaranteed to exist by property q. Since $\mathsf{a} \sim \mathsf{b}$, we have $(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{a}) \sim (\mathsf{q}^\alpha@\mathsf{a}@\mathsf{b})$. By property q, we have $\upsilon(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{a}) \equiv \mathsf{T}$ (since $\mathsf{a} \equiv \mathsf{a}$). Since $\sim$ is a congruence on the model, we have $\upsilon(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$. By property q, this means $\mathsf{a} \equiv \mathsf{b}$.

Since $\dot{\sim}$ is a particular congruence on $\mathscr{M}$, we know (2) implies (3).

To show (3) implies (4), we need to show $\mathfrak{L}^\alpha_=(\mathscr{E}(\mathbf{Q}^\alpha))$ holds for each type α. By the definition of $\dot{\sim}$, for every $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$ we have $\upsilon(\mathscr{E}(\mathbf{Q}^\alpha)@\mathsf{a}@\mathsf{b}) \equiv \mathsf{T}$, if and only if $\mathsf{a} \dot{\sim} \mathsf{b}$, iff $\mathsf{a} \equiv \mathsf{b}$. The last equivalence holds by our assumption that $\mathsf{a} \dot{\sim} \mathsf{b}$ implies that $\mathsf{a} \equiv \mathsf{b}$, and by Lemma 3.60.

For each type α, $\mathfrak{L}^\alpha_=(\mathscr{E}(\mathbf{Q}^\alpha))$ implies $\mathscr{E}(\mathbf{Q}^\alpha)$ is the witness required to show property q. So, we know (4) implies (1). ⊣

REMARK 3.64 (Congruences for Σ-models with primitive equality). Theorem 3.63 shows that once we have a model $\mathscr{M}$ which satisfies property q, there are no nontrivial congruences on $\mathscr{M}$. Hence, there are no nontrivial quotients of $\mathscr{M}$. In particular, the only possible congruence for a Σ-model with primitive equality is the trivial congruence given by the identity relation $\equiv$. Consequently, the quotient construction in the case of a Σ-model with primitive equality leads to essentially the same model again. We therefore do not consider quotients of models with primitive equality.

3.4. Σ-models over frames. In this section, we define the notion of an isomorphism between two models and show every functional Σ-model is isomorphic to a

model over a frame. In particular, this shows that the model class $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ is simply the closure of the class $\mathfrak{H}$ of Henkin models under isomorphism of Σ-models.

DEFINITION 3.65 (Σ-model homomorphism/isomorphism). Let $\mathscr{M}^1 \equiv (\mathscr{D}^1, @^1, \mathscr{E}^1, v^1)$ and $\mathscr{M}^2 \equiv (\mathscr{D}^2, @^2, \mathscr{E}^2, v^2)$ be Σ-models. A *homomorphism* from $\mathscr{M}^1$ to $\mathscr{M}^2$ is a typed function $\kappa\colon \mathscr{D}^1 \longrightarrow \mathscr{D}^2$ such that κ is a homomorphism from the evaluation $(\mathscr{D}^1, @^1, \mathscr{E}^1)$ to the evaluation $(\mathscr{D}^2, @^2, \mathscr{E}^2)$ and $v^1(\mathsf{a}) \equiv v^2(\kappa(\mathsf{a}))$ for every $\mathsf{a} \in \mathscr{D}^1_o$.

A homomorphism i from $\mathscr{M}^1$ to $\mathscr{M}^2$ is called an *isomorphism* iff there is a homomorphism j from $\mathscr{M}^2$ to $\mathscr{M}^1$ where $j_\alpha\colon \mathscr{D}^2_\alpha \longrightarrow \mathscr{D}^1_\alpha$ is the inverse of $i_\alpha\colon \mathscr{D}^1_\alpha \longrightarrow \mathscr{D}^2_\alpha$ at each type α. Two models are said to be *isomorphic* if there is such an isomorphism. (It is clear from the definition that this is a symmetric relationship between models.)

REMARK 3.66. The class $\mathfrak{H}$ of Henkin models is not closed under isomorphism of models. Neither is the class $\mathfrak{S}\mathfrak{T}$ of standard models. This is because Henkin and standard models require that the domains $\mathscr{D}_{\alpha\to\beta}$ consist of functions from $\mathscr{F}(\mathscr{D}_\alpha; \mathscr{D}_\beta)$. We may, however, take a given Henkin model and appropriately modify it to obtain an isomorphic model that is not in the class of Henkin models. For example, we may choose $\mathscr{D}'_{\alpha\to\beta} := \{\, (0, f) \mid f \in \mathscr{D}_{\alpha\to\beta} \,\}$ and define @ appropriately (cf. Example 5.6 for a similar construction).

LEMMA 3.67. *Let $\mathscr{M}^1$ and $\mathscr{M}^2$ be isomorphic Σ-models.*

(1) *For any set of sentences Φ, $\mathscr{M}^1 \models \Phi$, iff $\mathscr{M}^2 \models \Phi$.*
(2) *If $\mathscr{M}^1$ is a Σ-model with primitive equality, then $\mathscr{M}^2$ is a Σ-model with primitive equality.*
(3) *If $* \in \{\mathsf{q}, \eta, \xi, \mathfrak{f}, \mathfrak{b}\}$ and $\mathscr{M}^1$ satisfies $*$, then $\mathscr{M}^2$ satisfies $*$.*

In particular, each model class $\mathfrak{M}_*$ is closed under isomorphism of models.

PROOF. Let i be a homomorphism from $\mathscr{M}^1 \equiv (\mathscr{D}^1, @^1, \mathscr{E}^1, v^1)$ to $\mathscr{M}^2 \equiv (\mathscr{D}^2, @^2, \mathscr{E}^2, v^2)$ and j be its inverse.

Let Φ be a set of sentences with $\mathscr{M}^1 \models \Phi$. That is, for every $\boldsymbol{A} \in \Phi$, $v^1(\mathscr{E}^1(\boldsymbol{A})) \equiv \mathsf{T}$. So, for every $\boldsymbol{A} \in \Phi$, $v^2(\mathscr{E}^2(\boldsymbol{A})) \equiv v^1(j(\mathscr{E}^2(\boldsymbol{A}))) \equiv v^1(\mathscr{E}^1(\boldsymbol{A})) \equiv \mathsf{T}$ (since $\boldsymbol{A}$ is closed, we can ignore the variable assignment). This shows $\mathscr{M}^2 \models \Phi$; the other direction is obtained by switching indices.

Suppose $\mathsf{q}^\alpha \in \mathscr{D}^1_{\alpha\to\alpha\to o}$ is such that $\mathfrak{L}^\alpha_=(\mathsf{q}^\alpha)$ holds for v^1. We show that $\mathfrak{L}^\alpha_=(i(\mathsf{q}^\alpha))$ holds for v^2. Given $\mathsf{a}, \mathsf{b} \in \mathscr{D}^2_\alpha$. We have $\mathsf{a} \equiv \mathsf{b}$, iff $j(\mathsf{a}) \equiv j(\mathsf{b})$, iff $v^1(\mathsf{q}^\alpha @^1 j(\mathsf{a}) @^1 j(\mathsf{b})) \equiv \mathsf{T}$, iff $v^2(i(\mathsf{q}^\alpha @^1 j(\mathsf{a}) @^1 j(\mathsf{b}))) \equiv \mathsf{T}$, iff $v^2(i(\mathsf{q}^\alpha) @^2 \mathsf{a} @^2 \mathsf{b})) \equiv \mathsf{T}$.

In particular, suppose $\mathscr{M}^1$ is a Σ-model with primitive equality. Then, we have $\mathfrak{L}^\alpha_=(\mathscr{E}^1(=^\alpha))$ for v^1 at each type α. So, $\mathfrak{L}^\alpha_=(i(\mathscr{E}^1(=^\alpha)))$ holds for v^2 at each type α. Since $i(\mathscr{E}^1(=^\alpha)) \equiv \mathscr{E}^2(=^\alpha)$, we know $\mathscr{M}^2$ is a Σ-model with primitive equality.

Next, suppose $\mathscr{M}^1$ satisfies property q. Let α be a type and q^α be the witness for property q in $\mathscr{M}^1$ at α. That is, $\mathfrak{L}^\alpha_=(\mathsf{q}^\alpha)$ holds for v^1. We have shown $\mathfrak{L}^\alpha_=(i(\mathsf{q}^\alpha))$ holds for v^2. Hence, $\mathscr{M}^2$ satisfies property q.

Suppose $\mathscr{M}^1$ satisfies property η. To show $\mathscr{M}^2$ satisfies η, let $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and an assignment φ into $\mathscr{M}^2$ be given. We compute

$$\begin{aligned}\mathscr{E}^2_\varphi(\boldsymbol{A}) &\equiv (i \circ j)(\mathscr{E}^2_\varphi(\boldsymbol{A})) \equiv i(\mathscr{E}^1_{j\circ\varphi}(\boldsymbol{A})) \\ &\equiv i(\mathscr{E}^1_{j\circ\varphi}(\boldsymbol{A}{\downarrow_{\beta\eta}})) \equiv (i \circ j)(\mathscr{E}^2_\varphi(\boldsymbol{A}{\downarrow_{\beta\eta}})) \equiv \mathscr{E}^2_\varphi(\boldsymbol{A}{\downarrow_{\beta\eta}}).\end{aligned}$$

So, $\mathscr{M}^2$ satisfies property η.

$\mathscr{M}^2$ satisfies ξ, let $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_\beta(\Sigma)$, a variable X_α, and an assignment ψ into $\mathscr{M}^2$ be given. Suppose $\mathscr{E}^2_{\psi,[\mathsf{b}/X]}(\boldsymbol{M}) \equiv \mathscr{E}^2_{\psi,[\mathsf{b}/X]}(\boldsymbol{N})$ for all $\mathsf{b} \in \mathscr{D}^2_\alpha$. For any $\mathsf{a} \in \mathscr{D}^1_\alpha$, we compute

$$\begin{aligned}\mathscr{E}^1_{j\circ\psi,[\mathsf{a}/X]}(\boldsymbol{M}) &\equiv j(\mathscr{E}^2_{i\circ j\circ\psi,[i(\mathsf{a})/X]}(\boldsymbol{M})) \equiv j(\mathscr{E}^2_{\psi,[i(\mathsf{a})/X]}(\boldsymbol{M}))\\ &\equiv j(\mathscr{E}^2_{\psi,[i(\mathsf{a})/X]}(\boldsymbol{N})) \equiv \mathscr{E}^1_{j\circ\psi,[\mathsf{a}/X]}(\boldsymbol{N}).\end{aligned}$$

Since $\mathscr{M}^1$ satisfies property ξ, we know $\mathscr{E}^1_{j\circ\psi}(\lambda X.\boldsymbol{M}) \equiv \mathscr{E}^1_{j\circ\psi}(\lambda X.\boldsymbol{N})$. Finally, we compute

$$\mathscr{E}^2_\psi(\lambda X.\boldsymbol{M}) \equiv i(\mathscr{E}^1_{j\circ\psi}(\lambda X.\boldsymbol{M})) \equiv i(\mathscr{E}^1_{j\circ\psi}(\lambda X.\boldsymbol{N})) \equiv \mathscr{E}^2_\psi(\lambda X.\boldsymbol{N}).$$

So, $\mathscr{M}^2$ satisfies property ξ.

Suppose $\mathscr{M}^1$ satisfies property $\mathfrak{f}$ and we are given $\mathsf{f}, \mathsf{g} \in \mathscr{D}^2_{\alpha\to\beta}$ for types α and β. Suppose further that $\mathsf{f}@^2\mathsf{b} \equiv \mathsf{g}@^2\mathsf{b}$ for every $\mathsf{b} \in \mathscr{D}^2_\alpha$. It is enough to show $j(\mathsf{f}) \equiv j(\mathsf{g})$. This follows from property $\mathfrak{f}$ in $\mathscr{M}^1$ if we can show $j(\mathsf{f})@^1\mathsf{a} \equiv j(\mathsf{g})@^1\mathsf{a}$ for every $\mathsf{a} \in \mathscr{D}^1_\alpha$. So, let $\mathsf{a} \in \mathscr{D}^1_\alpha$ be given. We finish the proof by computing

$$\begin{aligned}j(\mathsf{f})@^1\mathsf{a} &\equiv j(\mathsf{f})@^1(j\circ i)(\mathsf{a}) \equiv j(\mathsf{f}@^2 i(\mathsf{a}))\\ &\equiv j(\mathsf{g}@^2 i(\mathsf{a})) \equiv j(\mathsf{g})@^1(j\circ i)(\mathsf{a}) \equiv j(\mathsf{g})@^1\mathsf{a}.\end{aligned}$$

Finally, if $\mathscr{M}^1$ satisfies property $\mathfrak{b}$, then $\mathscr{D}^1_o$ has two elements. Since $i_o : \mathscr{D}^1_o \longrightarrow \mathscr{D}^2_o$ has inverse j_o, $\mathscr{D}^2_o$ must also have two elements. Thus, $\mathscr{M}^2$ satisfies property $\mathfrak{b}$. ⊣

THEOREM 3.68 (Models over frames). *Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model which satisfies property $\mathfrak{f}$ (i.e., $\mathscr{M}$ is functional). Then there is an isomorphic model $\mathscr{M}^{fr}$ over a frame.*

PROOF. We define the model $\mathscr{M}^{fr} := (\mathscr{D}^{fr}, @^{fr}, \mathscr{E}^{fr}, \upsilon^{fr})$ by defining its components.

We first define the domains $\mathscr{D}^{fr}$ for $\mathscr{M}^{fr}$ by induction on types. We simultaneously define functions $i_\alpha : \mathscr{D}_\alpha \longrightarrow \mathscr{D}^{fr}_\alpha$ and $j_\alpha : \mathscr{D}^{fr}_\alpha \longrightarrow \mathscr{D}_\alpha$ which will witness that the two models are isomorphic. At each step of the definition, we check that i_α and j_α are mutual inverses. For base types $\alpha \in \{\iota, o\}$ let $\mathscr{D}^{fr}_\alpha := \mathscr{D}_\alpha$ and i_α and j_α be the identity functions (clearly mutual inverses).

Given two types α and β, we assume we have $\mathscr{D}^{fr}_\alpha$, mutual inverses $i_\alpha : \mathscr{D}_\alpha \to \mathscr{D}^{fr}_\alpha$ and $j_\alpha : \mathscr{D}^{fr}_\alpha \longrightarrow \mathscr{D}_\alpha$, as well as $\mathscr{D}^{fr}_\beta$ and mutual inverses $i_\beta : \mathscr{D}_\beta \to \mathscr{D}^{fr}_\beta$ and $j_\beta : \mathscr{D}^{fr}_\beta \longrightarrow \mathscr{D}_\beta$. We define

$$\mathscr{D}^{fr}_{\alpha\to\beta} := \left\{ f : \mathscr{D}^{fr}_\alpha \longrightarrow \mathscr{D}^{fr}_\beta \;\middle|\; \exists \mathsf{f} \in \mathscr{D}_{\alpha\to\beta}. \forall a \in \mathscr{D}^{fr}_\alpha . f(a) \equiv i_\beta(\mathsf{f}@j_\alpha(a)) \right\}.$$

Note that $\mathscr{D}^{fr}_{\alpha\to\beta} \subseteq \mathscr{F}(\mathscr{D}^{fr}_\alpha; \mathscr{D}^{fr}_\beta)$. To define the map $i_{\alpha\to\beta} : \mathscr{D}_{\alpha\to\beta} \longrightarrow \mathscr{D}^{fr}_{\alpha\to\beta}$, we let $i_{\alpha\to\beta}(\mathsf{f})$ be the function taking each $a \in \mathscr{D}^{fr}_\alpha$ to $i_\beta(\mathsf{f}@j_\alpha(a))$. This choice for $i_{\alpha\to\beta}(\mathsf{f})$ is clearly in $\mathscr{D}^{fr}_{\alpha\to\beta}$ by definition. To define the inverse map $j_{\alpha\to\beta} : \mathscr{D}^{fr}_{\alpha\to\beta} \longrightarrow \mathscr{D}_{\alpha\to\beta}$, we must use the fact that $\mathscr{M}$ is functional. Given any $f \in \mathscr{D}^{fr}_{\alpha\to\beta}$, by definition there is some $\mathsf{f} \in \mathscr{D}_{\alpha\to\beta}$ such that $f(a) \equiv i_\beta(\mathsf{f}@j_\alpha(a))$ for every $a \in \mathscr{D}^{fr}_\alpha$. (Note that the function f and object f are different in general.) By functionality and the fact that the i and j at types α and β are already inverses, this f is unique, since if

$i_\beta(\mathsf{f}@j_\alpha(a)) \equiv i_\beta(\mathsf{g}@j_\alpha(a))$ for every $a \in \mathscr{D}_\alpha^{fr}$, then $\mathsf{f}@j_\alpha(i_\alpha(\mathsf{a})) \equiv \mathsf{g}@j_\alpha(i_\alpha(\mathsf{a}))$ for every $\mathsf{a} \in \mathscr{D}_\alpha^{fr}$. That is, $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for every $\mathsf{a} \in \mathscr{D}_\alpha^{fr}$. So, for every $f \in \mathscr{D}_{\alpha\to\beta}^{fr}$, we define $j_{\alpha\to\beta}(f)$ to be the *unique* f such that $f(a) \equiv i_\beta(\mathsf{f}@j_\alpha(a))$. It is easy to check that $i_{\alpha\to\beta}$ and $j_{\alpha\to\beta}$ are mutually inverse.

For the applicative structure $(\mathscr{D}^{fr}, @^{fr})$ to be a frame, we are forced to let the application operator $@^{fr}$ to be function application. That is, for every $f \in \mathscr{D}_{\alpha\to\beta}^{fr}$ and $a \in \mathscr{D}_\alpha^{fr}$, $f\,@^{fr}a := f(a)$. We define the evaluation function $\mathscr{E}^{fr}$ simply by $\mathscr{E}_\varphi^{fr}(\boldsymbol{A}) := i(\mathscr{E}_{j\circ\varphi}(\boldsymbol{A}))$ for every $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$ and assignment φ into the applicative structure $(\mathscr{D}^{fr}, @^{fr})$. Since $\mathscr{D}_o^{fr} \equiv \mathscr{D}_o$, we can let $\upsilon^{fr} := \upsilon$.

We only sketch the remainder of the proof. First one can show that i and j preserve application. One can use this fact to verify that $\mathscr{E}^{fr}$ is an evaluation function so that $(\mathscr{D}^{fr}, @^{fr}, \mathscr{E}^{fr})$ is a Σ-evaluation, and that $\upsilon^{fr} \equiv \upsilon$ is a valuation function for this evaluation. This verifies $\mathscr{M}^{fr}$ is a model. Finally, to verify one has an isomorphism, one can easily check the remainder of the conditions for i and j to be homomorphisms between the models. These are isomorphisms since they are mutually inverse on the domains of each type. ⊣

We can conclude that $\mathfrak{M}_{\beta\mathfrak{fb}}$ is simply the closure of the class of $\mathfrak{H}$ of Henkin models under isomorphism. Given any $\mathscr{M} \in \mathfrak{M}_{\beta\mathfrak{fb}}$, by Theorem 3.68, there is an isomorphic model $\mathscr{M}^{fr}$ over a frame. By Lemma 3.67, this model $\mathscr{M}^{fr}$ satisfies $\mathfrak{q}$, $\mathfrak{f}$, and $\mathfrak{b}$ (since $\mathscr{M}$ does). Also, if primitive equality is present in the signature, by the same lemma we know $\mathscr{M}^{fr}$ is a model with primitive equality. That is, $\mathscr{M}^{fr} \in \mathfrak{H}$.

§4. Properties of model classes. In this section we discuss some properties of the model classes introduced in section 3. Our interest is in the properties of Leibniz equality and primitive equality.

DEFINITION 4.1 (Extensionality for Leibniz equality). We call a formula of the form

$$\mathrm{EXT}_{\doteq}^{\alpha\to\beta} \quad := \quad \forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX \doteq^\beta GX) \Rightarrow F \doteq^{\alpha\to\beta} G$$

an *axiom of (strong) functional extensionality for Leibniz equality*, and refer to the set

$$\mathrm{EXT}_{\doteq}^{\rightarrow} := \{\, \mathrm{EXT}_{\doteq}^{\alpha\to\beta} \mid \alpha, \beta \in \mathscr{T} \,\}$$

as the *axioms of (strong) functional extensionality for Leibniz equality*. Note that $\mathrm{EXT}_{\doteq}^{\rightarrow}$ specifies functionality of the relation corresponding to Leibniz equality $\doteq$. We call the formula

$$\mathrm{EXT}_{\doteq}^{o} \quad := \quad \forall A_o.\forall B_o.(A \Leftrightarrow B) \Rightarrow A \doteq^o B$$

the *axiom of Boolean extensionality*. We call the set $\mathrm{EXT}_{\doteq}^{\rightarrow} \cup \{\mathrm{EXT}_{\doteq}^{o}\}$ the *axioms of (strong) extensionality for Leibniz equality*.

In Examples 5.4 to 5.8 below we give concrete models in which $\mathrm{EXT}_{\doteq}^{o}$ and $\mathrm{EXT}_{\doteq}^{\alpha\to\beta}$ fail in various ways. First, we prove relationships between properties $\mathfrak{q}$, $\mathfrak{b}$ and $\mathfrak{f}$ and the statements $\mathrm{EXT}_{\doteq}^{o}$ and $\mathrm{EXT}_{\doteq}^{\rightarrow}$.

LEMMA 4.2 (Leibniz equality in Σ-models). *Let $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ be a Σ-model, φ be an assignment, $\alpha \in \mathscr{T}$, and $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_\alpha(\Sigma)$.*

(1) *If* $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$, *then* $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})) \equiv \mathtt{T}$.
(2) *If* $\mathscr{M}$ *satisfies property* $\mathfrak{q}$ *and* $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})) \equiv \mathtt{T}$, *then* $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$.

PROOF. Let φ be any assignment into $\mathscr{M}$. For the first part, suppose $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$. Given $\mathsf{r} \in \mathscr{D}_{\alpha\to o}$, we have either $\upsilon(\mathsf{r}@\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \upsilon(\mathsf{r}@\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{F}$ or $\upsilon(\mathsf{r}@\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \upsilon(\mathsf{r}@\mathscr{E}_\varphi(\boldsymbol{A})) \equiv \mathtt{T}$. In either case, for any variable $P_{\alpha\to o}$ not in $\mathrm{free}(\boldsymbol{A}) \cup \mathrm{free}(\boldsymbol{B})$, we have $\upsilon(\mathscr{E}_{\varphi,[\mathsf{r}/P]}(\neg(P\boldsymbol{A}) \vee P\boldsymbol{B})) \equiv \mathtt{T}$. So, we have $\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \equiv \mathtt{T}$.

To show the second part, suppose $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})) \equiv \mathtt{T}$. By property $\mathfrak{q}$, there is some $\mathsf{q}^\alpha \in \mathscr{D}_{\alpha\to\alpha\to o}$ such that for $\mathsf{a}, \mathsf{b} \in \mathscr{D}_\alpha$ we have $\upsilon(\mathsf{q}^\alpha@\mathsf{a}@\mathsf{b}) \equiv \mathtt{T}$ iff $\mathsf{a} \equiv \mathsf{b}$. Let $\mathsf{r} \equiv \mathsf{q}^\alpha@\mathscr{E}_\varphi(\boldsymbol{A})$. From $\upsilon(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})) \equiv \mathtt{T}$, we obtain $\mathscr{E}_{\varphi,[\mathsf{r}/P]}(\neg P\boldsymbol{A} \vee P\boldsymbol{B}) \equiv \mathtt{T}$ (where $P_{\alpha\to o} \notin \mathrm{free}(\boldsymbol{A}) \cup \mathrm{free}(\boldsymbol{B})$). Since $\mathscr{E}_{\varphi,[\mathsf{r}/P]}(P\boldsymbol{A}) \equiv \mathsf{q}^\alpha@\mathscr{E}_\varphi(\boldsymbol{A})@\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathtt{T}$, we must have $\upsilon(\mathscr{E}_{\varphi,[\mathsf{r}/P]}(P\boldsymbol{B})) \equiv \mathtt{T}$. That is, $\upsilon(\mathsf{q}^\alpha@\mathscr{E}_\varphi(\boldsymbol{A})@\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{T}$. By the choice of q^α, we have $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$. ⊣

THEOREM 4.3 (Extensionality in Σ-models). *Let* $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ *be a* Σ*-model.*
(1) *If* $\mathscr{M}$ *satisfies property* $\mathfrak{q}$ *but not property* $\mathfrak{f}$, *then* $\mathscr{M} \not\models \mathrm{EXT}^{\to}_{\doteq}$.
(2) *If* $\mathscr{M}$ *satisfies property* $\mathfrak{q}$ *but not property* $\mathfrak{b}$, *then* $\mathscr{M} \not\models \mathrm{EXT}^{o}_{\doteq}$.
(3) *If* $\mathscr{M}$ *satisfies properties* $\mathfrak{q}$ *and* $\mathfrak{f}$, *then* $\mathscr{M} \models \mathrm{EXT}^{\to}_{\doteq}$.
(4) *If* $\mathscr{M}$ *satisfies property* $\mathfrak{b}$, *then* $\mathscr{M} \models \mathrm{EXT}^{o}_{\doteq}$.

Thus we can characterize the different semantical structures with respect to Boolean and functional extensionality by the table in Figure 5.[7]

in	$\mathfrak{M}_\beta$, $\mathfrak{M}_{\beta\eta}$, $\mathfrak{M}_{\beta\xi}$		$\mathfrak{M}_{\beta\mathfrak{f}}$		$\mathfrak{M}_{\beta\mathfrak{b}}$, $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, $\mathfrak{M}_{\beta\xi\mathfrak{b}}$		$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$	
formula	valid?	by	valid?	by	valid?	by	valid?	by
$\mathrm{EXT}^{\to}_{\doteq}$	—	1.	+	3.	—	1.	+	3.
$\mathrm{EXT}^{o}_{\doteq}$	—	2.	—	2.	+	4.[7]	+	4.[7]

FIGURE 5. Extensionality in Σ-models.

PROOF. Suppose $\mathscr{M}$ satisfies property $\mathfrak{q}$ but does not satisfy property $\mathfrak{f}$. Then there must be types α and β and objects $\mathsf{f}, \mathsf{g} \in \mathscr{D}_{\alpha\to\beta}$ such that $\mathsf{f} \not\equiv \mathsf{g}$ but $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. Let $F_{\alpha\to\beta}, G_{\alpha\to\beta} \in \mathscr{V}_{\alpha\to\beta}$ be distinct variables, $X_\alpha \in \mathscr{V}_\alpha$, and φ be any assignment with $\varphi(F) \equiv \mathsf{f}$ and $\varphi(G) \equiv \mathsf{g}$. For any $\mathsf{a} \in \mathscr{D}_\alpha$, $\mathsf{f}@\mathsf{a} \equiv \mathsf{g}@\mathsf{a}$ implies $\upsilon(\mathscr{E}_{\varphi,[\mathsf{a}/X]}(FX \doteq^\beta GX)) \equiv \mathtt{T}$ by Lemma 4.2(1). Using the fact that υ is a valuation, we have $\upsilon(\mathscr{E}_\varphi(\forall X.(FX \doteq^\beta GX))) \equiv \mathtt{T}$. On the other hand, since $\mathsf{f} \not\equiv \mathsf{g}$ and $\mathscr{M}$ satisfies property $\mathfrak{q}$, we have $\upsilon(\mathscr{E}_\varphi(F \doteq^{\alpha\to\beta} G)) \equiv \mathtt{F}$ by contraposition of Lemma 4.2(2). This implies $\mathscr{M} \not\models \mathrm{EXT}^{\alpha\to\beta}_{\doteq}$.

Suppose $\mathscr{M}$ satisfies property $\mathfrak{q}$ but does not satisfy property $\mathfrak{b}$. Then, there must be at least three elements in $\mathscr{D}_o$. Since υ maps into a two element set, there must be two distinct elements $\mathsf{a}, \mathsf{b} \in \mathscr{D}_o$ such that $\upsilon(\mathsf{a}) \equiv \upsilon(\mathsf{b})$. Let $A_o, B_o \in \mathscr{V}_o$ be distinct variables and φ be any assignment into $\mathscr{M}$ with $\varphi(A) \equiv \mathsf{a}$ and $\varphi(B) \equiv \mathsf{b}$. By Lemma 3.48, we know $\upsilon(\mathscr{E}_\varphi(A \Leftrightarrow B)) \equiv \mathtt{T}$. Since $\mathsf{a} \not\equiv \mathsf{b}$ and property $\mathfrak{q}$ holds,

[7] The cases in the figure corresponding to Theorem 4.3(4) are actually special cases. In Theorem 4.3(4), we can infer a model satisfies $\mathrm{EXT}^{o}_{\doteq}$ even if property $\mathfrak{q}$ does not hold. However, the models in $\mathfrak{M}_{\beta\mathfrak{b}}$, $\mathfrak{M}_{\beta\eta\mathfrak{b}}$, $\mathfrak{M}_{\beta\xi\mathfrak{b}}$ and $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ do satisfy property $\mathfrak{q}$ by the definition of these model classes.

by contraposition of Lemma 4.2(2), we know $\upsilon(\mathscr{E}_\varphi(A \doteq^o B)) \equiv \mathtt{F}$. It follows that $\mathscr{M} \not\models \mathrm{EXT}^o_{\doteq}$.

Let φ be any assignment into $\mathscr{M}$. From $\upsilon(\mathscr{E}_\varphi(\forall X_\alpha . FX \doteq GX)) \equiv \mathtt{T}$ we know $\upsilon(\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(FX \doteq GX)) \equiv \mathtt{T}$ holds for all $\mathfrak{a} \in \mathscr{D}_\alpha$. By Lemma 4.2(2) we can conclude that $\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(FX) \equiv \mathscr{E}_{\varphi,[\mathfrak{a}/X]}(GX)$ for all $\mathfrak{a} \in \mathscr{D}_\alpha$ and hence $\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(F)@\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(X) \equiv \mathscr{E}_{\varphi,[\mathfrak{a}/X]}(G)@\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(X)$ for all $\mathfrak{a} \in \mathscr{D}_\alpha$. That is, $\mathscr{E}_{\varphi,[\mathfrak{a}/X]}(F)@\mathfrak{a} \equiv \mathscr{E}_{\varphi,[\mathfrak{a}/X]}(G)@\mathfrak{a}$ for all $\mathfrak{a} \in \mathscr{D}_\alpha$. Since X does not occur free in F or G, by property $\mathfrak{f}$ and Definition 3.18(3) we obtain $\mathscr{E}_\varphi(F) \equiv \mathscr{E}_\varphi(G)$. This finally gives us that $\upsilon(\mathscr{E}_\varphi(F \doteq^{\alpha\to\beta} G)) \equiv \mathtt{T}$ with Lemma 4.2(1). It follows that $\mathscr{M} \models \mathrm{EXT}^{\alpha\to\beta}_{\doteq}$ and $\mathscr{M} \models \mathrm{EXT}^{\to}_{\doteq}$, since α and β were chosen arbitrarily. Note that we certainly need the assumption that $\mathscr{M}$ satisfies property $\mathfrak{q}$ (which is employed within the application of Lemma 4.2(2). As explained in Remark 3.52, there is a functional model in which property $\mathfrak{q}$ fails and $\mathrm{EXT}^{\iota\to\iota}_{\doteq}$ is not valid.

Let $A_o, B_o \in \mathscr{V}_o$ be distinct variables and φ be any assignment into $\mathscr{M}$. Since property $\mathfrak{b}$ holds, we can assume $\mathscr{D}_o \equiv \{\mathtt{T}, \mathtt{F}\}$ and υ is the identity function. Suppose $\upsilon(\mathscr{E}_\varphi(A \Leftrightarrow B)) \equiv \mathtt{T}$. By Lemma 3.48, we have $\mathscr{E}_\varphi(A) \equiv \upsilon(\mathscr{E}_\varphi(A)) \equiv \upsilon(\mathscr{E}_\varphi(B)) \equiv \mathscr{E}_\varphi(B)$. By Lemma 4.2(1), we have $\upsilon(\mathscr{E}_\varphi(A \doteq^o B)) \equiv \mathtt{T}$. It follows that $\mathscr{M} \models \mathrm{EXT}^o_{\doteq}$. ⊣

REMARK 4.4 (Alternative definitions of equality). Leibniz equality is a very prominent way of defining equality in higher-order logic. However, there are alternative definitions such as (cf. [6, p. 203])

$$\ddot{=}^\alpha := \lambda X_\alpha Y_\alpha . \forall Q_{\alpha\to\alpha\to o} . (\forall Z_\alpha . QZZ) \Rightarrow QXY.$$

An important question is whether an alternative definition of equality is equivalent to the Leibniz definition in particular model classes. As Remark 3.47 shows, this has to be carefully investigated for each equality definition and each model class in question. We can show that for all $\boldsymbol{A}_\alpha, \boldsymbol{B}_\alpha \in \mathrm{cwff}_\alpha(\Sigma)$ $\boldsymbol{A} \ddot{=} \boldsymbol{B}$ and $\boldsymbol{A} \doteq \boldsymbol{B}$ are equivalent modulo υ for all $\mathscr{M} \in \mathfrak{M}_\beta$ (and thus for all other model classes). That is, we can show $\upsilon(\mathscr{E}(\boldsymbol{A} \ddot{=}^\alpha \boldsymbol{B})) \equiv \upsilon(\mathscr{E}(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}))$. Note that this is weaker than showing $\mathscr{E}(\boldsymbol{A} \ddot{=}^\alpha \boldsymbol{B}) \equiv \mathscr{E}(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})$. The key idea is to reduce the definition of $\ddot{=}$ to $\doteq$ (and vice versa) by instantiating the universally quantified set variables Q and P appropriately. We may, for instance, show $\boldsymbol{A} \ddot{=}^\alpha \boldsymbol{B}$ implies $\boldsymbol{A} \doteq^\alpha \boldsymbol{B}$ by choosing the instantiation $[\lambda U_\alpha V_\alpha . \forall P_{\alpha\to o} . PU \Rightarrow PV]$ for Q and the converse by choosing the instantiation $[\lambda V_\alpha . \forall Q_{\alpha\to\alpha\to o} . (\forall Z_\alpha . QZZ) \Rightarrow QAV]$ for P. As a consequence the properties of Leibniz equality with respect to extensionality also apply to $\ddot{=}$.

DEFINITION 4.5 (Extensionality for primitive equality). Analogous to the extensionality axioms for Leibniz equality, we can define the *axioms of strong (functional and Boolean) extensionality for primitive equality*:

$$\mathrm{EXT}^{\alpha\to\beta}_{=} := \forall F_{\alpha\to\beta} . \forall G_{\alpha\to\beta} . (\forall X_\alpha . FX =^\beta GX) \Rightarrow F =^{\alpha\to\beta} G$$

$$\mathrm{EXT}^o_{=} := \forall A_o . \forall B_o . (A \Leftrightarrow B) \Rightarrow A =^o B.$$

As before we refer to the set $\mathrm{EXT}^{\to}_{=} := \{\mathrm{EXT}^{\alpha\to\beta}_{=} \mid \alpha, \beta \in \mathscr{T}\}$ as the *axioms of (strong) functional extensionality for primitive equality*.

The following lemma shows that in a Σ-model with primitive equality for each $\alpha \in \mathscr{T}$ the denotations of $=^\alpha$ and $\doteq^\alpha$ are identical modulo υ.

LEMMA 4.6 (Primitive and Leibniz equality). *If* $\mathscr{M} := (\mathscr{D}, @, \mathscr{E}, v) \in \mathfrak{M}_*$ *is a* Σ*-model with primitive equality where* $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$*, then we have* $v(\mathscr{E}_\varphi(\boldsymbol{A} =^\alpha \boldsymbol{B})) \equiv v(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}))$ *for all assignments* φ *into* $\mathscr{M}$*, types* $\alpha \in \mathscr{T}$*, and* $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_\alpha(\Sigma)$.

PROOF. Since property $\mathfrak{q}$ holds for $\mathscr{M} \in \mathfrak{M}_*$, by Lemma 4.2 parts (1) and (2), we have $v(\mathscr{E}_\varphi(\boldsymbol{A} \doteq^\alpha \boldsymbol{B})) \equiv \mathtt{T}$ iff $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$. Since $\mathscr{M}$ is a Σ-model with primitive equality, we know $\mathscr{E}_\varphi(\boldsymbol{A}) \equiv \mathscr{E}_\varphi(\boldsymbol{B})$ is equivalent to $v(\mathscr{E}(=^\alpha)@\mathscr{E}_\varphi(\boldsymbol{A})@\mathscr{E}_\varphi(\boldsymbol{B})) \equiv \mathtt{T}$, and hence to $v(\mathscr{E}_\varphi(\boldsymbol{A} =^\alpha \boldsymbol{B})) \equiv \mathtt{T}$. ⊣

REMARK 4.7. Lemma 4.6 implies that for all models in our model classes $\mathfrak{M}_*$ the extensionality axioms for primitive equality are equivalent to the corresponding extensionality axioms for Leibniz equality. Thus, the analysis for the Leibniz versions applies directly to the versions using primitive equality. Also, Lemma 4.6 reinforces that (provided property $\mathfrak{q}$ holds) we can indeed use Leibniz equality to treat equality as a defined notion (relative to models in $\mathfrak{M}_*$). Thus, we principally do not need to assume the constants $=^\alpha$ to be in our signature. The critical part in this choice is that for ensuring the correct meaning for $\boldsymbol{Q}^\alpha$ we have to require the existence of an object representing the identity relation for each type in each Σ-model (cf. [2] for a discussion in the context of Henkin models). This requirement is automatically met if we consider primitive equality. Hence it seems natural to treat equality as primitive.

REMARK 4.8 (Properties η and ξ). We have shown, in the presence of property $\mathfrak{q}$, a model $\mathscr{M}$ satisfies property $\mathfrak{f}$ iff $\mathscr{M} \models \mathrm{EXT}^{\rightarrow}_{\doteq}$. Similarly, we have shown that property $\mathfrak{b}$ corresponds to a model satisfying $\mathrm{EXT}^{o}_{\doteq}$. A corresponding analysis can be done for properties η and ξ (cf. Definition 3.46). Assume $\mathscr{M}$ satisfies property $\mathfrak{q}$. Then, $\mathscr{M}$ satisfies property η iff $\mathscr{M} \models \boldsymbol{A} \doteq^\alpha (\boldsymbol{A}{\downarrow_{\beta\eta}})$ for every type α and closed formula $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$. Also, $\mathscr{M}$ satisfies property ξ iff

$$\mathscr{M} \models \forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX \doteq^\beta GX) \Rightarrow (\lambda X.FX) \doteq^{\alpha\to\beta} (\lambda X.GX)$$

for all types α and β.

§5. Example models. We now sketch the construction of models in the model classes $\mathfrak{M}_*$ to demonstrate concretely how properties for Boolean, strong and weak functional extensionality can fail. We need this to show that the inclusions (cf. Figure 1) of the model classes defined in Section 3 are proper, and we indeed need all of them.

We start with the simplest example of a Henkin model, which we will call the *singleton model*, since the domain of individuals is a singleton. Note that the underlying evaluation of this model is not the singleton evaluation from Example 3.26 since $\mathscr{D}_o$ has two elements. In this model, all forms of extensionality are valid.

EXAMPLE 5.1 (Singleton model—$\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \in \mathfrak{ST} \subseteq \mathfrak{H} \subseteq \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$). Let $(\mathscr{D}, @)$ be the full frame with $\mathscr{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathscr{D}_\iota := \{*\}$. One can easily define an evaluation function $\mathscr{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions. The identity function $v\colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way (including primitive equality, if present in Σ). So, $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} := (\mathscr{D}, @, \mathscr{E}, v)$ defines a model. This model clearly

satisfies all our properties $\mathfrak{b}$, $\mathfrak{f}$ (hence η and ξ) and $\mathfrak{q}$ (since the frame is full). So, $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \in \mathfrak{ST} \subseteq \mathfrak{H} \subseteq \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$.

REMARK 5.2. In particular, all our model classes are non-empty. By parts (3) and (4) of Theorem 4.3, we have $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \models \mathrm{EXT}^o_{\doteq}$ and $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \models \mathrm{EXT}^{\rightarrow}_{\doteq}$.

We can use the singleton model $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}}$ to construct another model which makes the importance of property $\mathfrak{q}$ clear.

REMARK 5.3. Let $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon)$ as above and $\mathscr{TE}(\Sigma)^\beta \equiv (\mathscr{D}^\beta, @^\beta, \mathscr{E}^\beta)$ be the β-term evaluation as defined in Definition 3.35. Let $\upsilon' \colon \mathscr{D}^\beta_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be the function $\upsilon'(\boldsymbol{A}) := \upsilon(\mathscr{E}(\boldsymbol{A}))$ for every $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$. One can show $\mathscr{M}' := (\mathscr{D}^\beta, @^\beta, \mathscr{E}^\beta, \upsilon')$ is a Σ-model such that $\mathscr{M}' \models \boldsymbol{A}$ iff $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \models \boldsymbol{A}$ for every sentence $\boldsymbol{A}$. In particular, $\mathscr{M}' \models \mathrm{EXT}^o_{\doteq}$ and $\mathscr{M}' \models \mathrm{EXT}^{\rightarrow}_{\doteq}$.

Nevertheless, $\mathscr{M}'$ fails to satisfy properties $\mathfrak{q}$, $\mathfrak{b}$, η and $\mathfrak{f}$. Property $\mathfrak{b}$ does not hold since $\mathscr{D}^\beta_o \equiv \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$ is infinite. Property η does not hold since, for example,

$$\mathscr{E}^\beta(\lambda F_{\iota\to\iota} X_\iota . FX) \equiv \lambda F_{\iota\to\iota} X_\iota . FX \not\equiv \lambda F_{\iota\to\iota} . F \equiv \mathscr{E}^\beta(\lambda F_{\iota\to\iota} . F).$$

Property $\mathfrak{f}$ cannot hold since property η does not hold. (On the other hand, property ξ does hold since the underlying evaluation is a term evaluation.)

We know now by Theorem 4.3, either part (1) or part (2), that property $\mathfrak{q}$ must not hold. A concrete way to see that property $\mathfrak{q}$ fails is to consider two distinct constants $a_\iota, b_\iota \in \Sigma_\iota$. We must have $\mathscr{M}^{\beta\mathfrak{f}\mathfrak{b}} \models a \doteq^\iota b$ (since $\mathscr{D}_\iota$ has only one element), and so $\mathscr{M}' \models a \doteq^\iota b$. On the other hand a and b are distinct elements (as distinct β-normal forms) in $\mathscr{D}^\beta_\iota$.

The model $\mathscr{M}'$ shows that property $\mathfrak{q}$ is needed in the proofs of parts (1) and (2) of Theorem 4.3.

EXAMPLE 5.4 (Failure of $\mathfrak{b}$—$\mathscr{M}^{\beta\mathfrak{f}} \in \mathfrak{M}_{\beta\mathfrak{f}} \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$). Let $(\mathscr{D}, @)$ be the full frame with $\mathscr{D}_o = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathscr{D}_\iota = \{0, 1\}$. We define an evaluation function $\mathscr{E}$ for this frame by defining $\mathscr{E}(\neg)$, $\mathscr{E}(\vee)$, and $\mathscr{E}(\Pi^\alpha)$ to be the functions given in the following table:

$\mathscr{E}(\neg)$	a	b	c
	c	c	a

$\mathscr{E}(\vee)$	a	b	c
a	a	a	a
b	a	a	a
c	a	a	c

$$\mathscr{E}(\Pi^\alpha)@\mathsf{f} = \begin{cases} \mathsf{a}, & \text{if } \mathsf{f}@\mathsf{g} \in \{\mathsf{a}, \mathsf{b}\} \text{ for all } \mathsf{g} \in \mathscr{D}_\alpha, \\ \mathsf{c}, & \text{if } \mathsf{f}@\mathsf{g} = \mathsf{c} \text{ for some } \mathsf{g} \in \mathscr{D}_\alpha. \end{cases}$$

We can choose $\mathscr{E}(w)$ to be arbitrary for parameters $w \in \Sigma$. Since the applicative structure $(\mathscr{D}, @)$ is a frame, hence functional, this uniquely determines $\mathscr{E}$ on all formulae. Also, since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.

Let the map $\upsilon \colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$. It is easy to check that $\mathscr{M}^{\beta\mathfrak{f}} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$ is indeed a Σ-model. Since this is a model over a frame, we automatically know it satisfies property $\mathfrak{f}$. Since the frame is full, we know property $\mathfrak{q}$ holds. (By the same argument, if primitive equality is in the signature, we can ensure $\mathscr{E}(=^\alpha)$ is interpreted appropriately for each type

α.) Clearly property $\mathfrak{b}$ fails, so we have $\mathscr{M}^{\beta\mathfrak{f}} \in \mathfrak{M}_{\beta\mathfrak{f}} \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$. By Theorem 4.3(2), $\mathscr{M}^{\beta\mathfrak{f}} \not\models \mathrm{EXT}^o_{\doteq}$.

In this model one can easily verify, if $\mathsf{d} := \mathscr{E}_\varphi(\boldsymbol{D}_o)$ and $\mathsf{e} := \mathscr{E}_\varphi(\boldsymbol{E}_o)$, then the values $\mathscr{E}_\varphi(\boldsymbol{D} \wedge \boldsymbol{E})$, $\mathscr{E}_\varphi(\boldsymbol{D} \Rightarrow \boldsymbol{E})$, and $\mathscr{E}_\varphi(\boldsymbol{D} \Leftrightarrow \boldsymbol{E})$ are given by the following tables:

$\mathscr{E}(\boldsymbol{D} \wedge \boldsymbol{E})$	e: a	e: b	e: c
d: a	a	a	c
d: b	a	a	c
d: c	c	c	c

$\mathscr{E}(\boldsymbol{D} \Rightarrow \boldsymbol{E})$	e: a	e: b	e: c
d: a	a	a	c
d: b	a	a	c
d: c	a	a	a

$\mathscr{E}(\boldsymbol{D} \Leftrightarrow \boldsymbol{E})$	e: a	e: b	e: c
d: a	a	a	c
d: b	a	a	c
d: c	c	c	a

Note that one can properly model the woodchuck / groundhog example from [39] referred to in the introduction in $\mathscr{M}^{\beta\mathfrak{f}}$.

EXAMPLE 5.5 (Groundhogs and woodchucks). Let $\mathscr{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathrm{woodchuck}_{\iota\to o}$, $\mathrm{groundhog}_{\iota\to o}$, john_ι, and phil_ι are in the signature Σ. Let $\mathscr{E}(\mathrm{phil}) := 0$ and $\mathscr{E}(\mathrm{john}) := 1$. Let $\mathscr{E}(\mathrm{woodchuck})$ be the function $\mathsf{w} \in \mathscr{D}_{\iota\to o}$ with $\mathsf{w}(0) \equiv \mathsf{b}$ and $\mathsf{w}(1) \equiv \mathsf{c}$. Let $\mathscr{E}(\mathrm{groundhog})$ be the function $\mathsf{g} \in \mathscr{D}_{\iota\to o}$ with $\mathsf{g}(0) \equiv \mathsf{a}$ and $\mathsf{g}(1) \equiv \mathsf{c}$. One can show that the sentence $\forall X_\iota.(\mathrm{woodchuck}\ X) \Leftrightarrow (\mathrm{groundhog}\ X)$ is valid. Also, $\mathscr{E}(\mathrm{woodchuck\ phil}) \equiv \mathsf{b}$ and $\mathscr{E}(\mathrm{groundhog\ phil}) \equiv \mathsf{a}$, so the propositions (woodchuck phil) and (groundhog phil) are valid. Next, suppose $\mathrm{believe}_{\iota\to o\to o} \in \Sigma$ and $\mathscr{E}(\mathrm{believe})$ is the (Curried) function $\mathsf{bel} \in \mathscr{D}_{\iota\to o\to o}$ such that $\mathsf{bel}(1)(\mathsf{b}) \equiv \mathsf{b}$ and $\mathsf{bel}(1)(\mathsf{a}) \equiv \mathsf{bel}(1)(\mathsf{c}) \equiv \mathsf{bel}(0)(\mathsf{a}) \equiv \mathsf{bel}(0)(\mathsf{b}) \equiv \mathsf{bel}(0)(\mathsf{c}) \equiv \mathsf{c}$ (Intuitively, John believes propositions with value b, but not those with value a or c). So, believes john(woodchuck phil) is valid, while believes john(groundhog phil) is not.

As we have seen, Boolean extensionality fails when one has more than two values in $\mathscr{D}_o$. We can generalize the construction defining $\mathscr{D}_o := \{\mathsf{F}\} \cup \mathscr{B}$, where $\mathscr{B}$ is any set with $\mathsf{T} \in \mathscr{B}$ and $\mathsf{F} \notin \mathscr{B}$. The model will satisfy Boolean extensionality iff $\mathscr{B} \equiv \{\mathsf{T}\}$. In this way, we can easily construct models for the case with property $\mathfrak{b}$ and the case without property $\mathfrak{b}$ simultaneously. We will use this idea to parameterize the remaining model constructions by $\mathscr{B}$. These semantic constructions are similar to those in multi-valued logics, which have been studied for higher-order logic in [38]. In contrast to these logics where the logical connectives are adapted to talk about multiple truth values, in our setting we are mainly interested in multiple truth values as diverse v-pre-images of T and F.

EXAMPLE 5.6 (Failure of $\mathfrak{f}$ and η—$\mathscr{M}^{\beta\xi\mathfrak{b}} \in \mathfrak{M}_{\beta\xi\mathfrak{b}} \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$). We start by constructing a non-functional applicative structure by attaching distinguishing labels to functions without changing their applicative behavior. Let $\mathscr{B}$ be any set with $\mathsf{T} \in \mathscr{B}$ and $\mathsf{F} \notin \mathscr{B}$. Let $\mathscr{D}_o := \{\mathsf{F}\} \cup \mathscr{B}$ and $\mathscr{D}_\iota := \{*\}$ with $*$ as singleton element. For each function type $\alpha \to \beta$, let

$$\mathscr{D}_{\alpha\to\beta} := \{\, (i, f) \mid i \in \{0, 1\} \text{ and } f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta \,\}.$$

Technically, we should write $\mathscr{D}^{\mathscr{B}}$ for $\mathscr{D}$, but to ease the notation, we wait until the model is defined to make its dependence on $\mathscr{B}$ explicit. We define application by $(i, f)@\mathsf{a} := f(\mathsf{a})$ whenever $(i, f) \in \mathscr{D}_{\alpha\to\beta}$ and $\mathsf{a} \in \mathscr{D}_\alpha$. It is easy to see that $(\mathscr{D}, @)$ is an applicative structure and is not functional. Consider, for example, the

unique function $u : \mathscr{D}_\iota \longrightarrow \mathscr{D}_\iota$. For both $(0,u),(1,u) \in \mathscr{D}_{\iota\to\iota}$ we have $(i,u)@* \equiv *$, although $(0,u) \not\equiv (1,u)$.

We can define an evaluation function by induction on terms. We must begin by interpreting the constants. For the logical constants, let $\mathscr{E}(\neg) := (0,n)$ where $n(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathscr{B}$ and $n(\mathtt{F}) := \mathtt{T}$. Let $\mathscr{E}(\vee) := (0,d)$ where $d(\mathsf{b}) := (0,k^{\mathtt{T}})$ for every $\mathsf{b} \in \mathscr{B}$, $d(\mathtt{F}) := (0,id)$, $k^{\mathtt{T}}$ is the constant $\mathtt{T}$ function and id is the identity function from $\mathscr{D}_o$ to $\mathscr{D}_o$. For each type α, let $d(\Pi^\alpha) := (0,\pi^\alpha)$ where for each $(i,f) \in \mathscr{D}_{\alpha\to o}$, $\pi^\alpha((i,f)) := \mathtt{T}$ if $f(\mathsf{a}) \in \mathscr{B}$ for all $\mathsf{a} \in \mathscr{D}_\alpha$ and $\pi^\alpha(i,f) := \mathtt{F}$ otherwise. For each type α, let $\mathsf{q}^\alpha := (0,q^\alpha) \in \mathscr{D}_{\alpha\to\alpha\to o}$ where $q^\alpha(\mathsf{a}) := (0,s^{\mathsf{a}})$ and $s^{\mathsf{a}}(\mathsf{b}) := \mathtt{T}$ if $\mathsf{a} \equiv \mathsf{b}$ and $s^{\mathsf{a}}(\mathsf{b}) := \mathtt{F}$ otherwise. If primitive equality is present in the signature, let $\mathscr{E}(=^\alpha) := \mathsf{q}^\alpha$. Let $\mathscr{E}(w) \in \mathscr{D}_\alpha$ be arbitrary for parameters $w \in \Sigma_\alpha$.

For variables, we must define $\mathscr{E}_\varphi(X) := \varphi(X)$. Similarly, for application, we must define $\mathscr{E}_\varphi(\boldsymbol{FA}) := \mathscr{E}_\varphi(\boldsymbol{F})@\mathscr{E}_\varphi(\boldsymbol{A})$. For λ-abstractions, we have a choice. To be definite, we choose $\mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{B}_\beta) := (0,f)$ where $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ is the function such that $f(\mathsf{a}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{B})$ for all $\mathsf{a} \in \mathscr{D}_\alpha$.

With some work (which we omit), one can show that this $\mathscr{E}$ is an evaluation function. Furthermore, taking υ to be the function such that $\upsilon(\mathsf{b}) := \mathtt{T}$ for every $\mathsf{b} \in \mathscr{B}$ and $\upsilon(\mathtt{F}) := \mathtt{F}$, one can easily show that this is a valuation. Hence, $\mathscr{M}^{\mathscr{B}} := (\mathscr{D},@,\mathscr{E},\upsilon)$ is a Σ-model.

The objects q^α witness property $\mathfrak{q}$ for $\mathscr{M}^{\mathscr{B}}$ (and also show that this is a model with primitive equality, when primitive equality is in the signature). Note that the objects $(1,q^\alpha)$ also witness property $\mathfrak{q}$. So, in the non-functional case such witnesses are not unique.

We have already noted that property $\mathfrak{f}$ fails, since the applicative structure is not functional. One may question whether properties η or ξ hold. In fact, property η does not, as one may verify by computing, for example, $\mathscr{E}(\lambda F_{\alpha\to\beta}.F)$ and $\mathscr{E}(\lambda F_{\alpha\to\beta}X_\alpha.FX)$ for types α and β. We have $\mathscr{E}(\lambda F_{\alpha\to\beta}.F) \equiv (0,id)$ where id is the identity function from $\mathscr{D}_{\alpha\to\beta}$ to $\mathscr{D}_{\alpha\to\beta}$. However, $\mathscr{E}(\lambda F_{\alpha\to\beta}X_\alpha.FX) \equiv (0,p)$ where p is the function from $\mathscr{D}_{\alpha\to\beta}$ to $\mathscr{D}_{\alpha\to\beta}$ such that $p((i,f)) \equiv (0,f)$ for each $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$. Property ξ does hold.[8] The reason is that if $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$ for every $\mathsf{a} \in \mathscr{D}_\alpha$, then $\mathscr{E}_\varphi(\lambda X_\alpha.\boldsymbol{M}) \equiv (0,f) \equiv \mathscr{E}_\varphi(\lambda X.\boldsymbol{N})$ where $f(\mathsf{a}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$ for every $\mathsf{a} \in \mathscr{D}_\alpha$.

Since $\mathscr{M}^{\mathscr{B}}$ is satisfies property $\mathfrak{q}$ but not property $\mathfrak{f}$, by Theorem 4.3(1) we have $\mathscr{M}^{\mathscr{B}} \not\models \mathrm{EXT}_{\doteq}^{\alpha\to\beta}$ for some types α and β. (One can easily check that, in fact, $\mathscr{M}^{\mathscr{B}} \not\models \mathrm{EXT}_{\doteq}^{\alpha\to\beta}$ for all types α and β by considering the witnesses $(0,f)$ and $(1,f)$ in $\mathscr{D}_{\alpha\to\beta}$ where $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ is any function.)

If $\mathscr{B} \equiv \{\mathtt{T}\}$, then the model $\mathscr{M}^{\beta\xi\mathfrak{b}} := \mathscr{M}^{\{\mathtt{T}\}}$ satisfies property $\mathfrak{b}$. So, we know $\mathscr{M}^{\beta\xi\mathfrak{b}} \in \mathfrak{M}_{\beta\xi\mathfrak{b}} \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$. On the other hand, if b is any value with $\mathsf{b} \notin \{\mathtt{T},\mathtt{F}\}$, and $\mathscr{B} \equiv \{\mathtt{T},\mathsf{b}\}$, then the model $\mathscr{M}^{\beta\xi} := \mathscr{M}^{\{\mathtt{T},\mathsf{b}\}}$ does not satisfy property $\mathfrak{b}$. In this case, we know $\mathscr{M}^{\beta\xi} \in \mathfrak{M}_{\beta\xi} \setminus (\mathfrak{M}_{\beta\mathfrak{f}} \cup \mathfrak{M}_{\beta\xi\mathfrak{b}})$.

[8] This construction is an example of how one constructs models for the simply typed λ-calculus using retractions. Such constructions will always yield models satisfying property ξ, but only yield models satisfying property η when each retraction is an isomorphism, in which case the applicative structure is functional.

REMARK 5.7. Let $\mathscr{M}^{\mathscr{B}}$ be the Σ-model $(\mathscr{D}, @, \mathscr{E}, \upsilon)$ constructed in Example 5.6. We can define an alternative evaluation function $\mathscr{E}'$ by induction on terms. For all $w \in \Sigma$, let $\mathscr{E}'(w) := \mathscr{E}(w)$. For variables, we define $\mathscr{E}'_\varphi(X) := \varphi(X)$. For application, we must define $\mathscr{E}'_\varphi(\boldsymbol{FA}) := \mathscr{E}'_\varphi(\boldsymbol{F})@\mathscr{E}'_\varphi(\boldsymbol{A})$. For λ-abstractions, we choose $\mathscr{E}'_\varphi(\lambda X_\alpha . \boldsymbol{B}_\beta) := (1, f)$ where $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ is the function such that $f(\mathsf{a}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{B})$ for all $\mathsf{a} \in \mathscr{D}_\alpha$. We omit checking $\mathscr{E}'$ is an evaluation function, but the verification is that same is checking $\mathscr{E}$ is an evaluation function. Notice that $\mathscr{E}$ and $\mathscr{E}'$ agree on all constants (by definition). However, they are different evaluation functions. For example,

$$\mathscr{E}(\lambda X_\iota . X) \equiv (0, \mathrm{id}) \not\equiv (1, \mathrm{id}) \equiv \mathscr{E}'(\lambda X_\iota . X)$$

where $\mathrm{id} : \mathscr{D}_\iota \longrightarrow \mathscr{D}_\iota$ is the identity function.This example shows that evaluation functions are not uniquely determined by their values on constants in non-functional models.

In Lemma 3.14, we have shown that $\beta\eta$-equality induces a functional congruence if the Σ_α is infinite for all types α. As a result, with such signatures, the term evaluation $\mathscr{TE}(\Sigma)^{\beta\eta}$ is functional (cf. Lemma 3.36). As noted in Remark 3.15, if Σ is finite, we cannot show that functionality holds. Nevertheless, even if Σ is finite, the evaluation $\mathscr{TE}(\Sigma)^{\beta\eta}$ interprets $\beta\eta$-convertible terms the same. We can use this idea to construct non-functional models which satisfy property η.

EXAMPLE 5.8 (Failure of ξ—Instances of $\mathfrak{M}_\beta$, $\mathfrak{M}_{\beta\eta}$, $\mathfrak{M}_{\beta\mathfrak{b}}$, $\mathfrak{M}_{\beta\eta\mathfrak{b}}$). Again, let $\mathscr{B}$ be any set with $\mathtt{T} \in \mathscr{B}$ and $\mathtt{F} \notin \mathscr{B}$. Choose constants $c_\iota, c_o \in \Sigma$ and let $\Sigma' := \{c_\iota, c_o\}$. By induction on types, we define $\boldsymbol{C}'_\alpha \in \mathrm{cwff}_\alpha(\Sigma')\downarrow_{\beta\eta} \subseteq \mathrm{cwff}_\alpha(\Sigma')\downarrow_\beta$. At base types, let $\boldsymbol{C}'_\iota := c_\iota$ and $\boldsymbol{C}'_o := c_o$. At function types, let $\boldsymbol{C}'_{\alpha\to\beta} := \lambda X_\alpha . \boldsymbol{C}'_\beta$. (Thus each $\boldsymbol{C}'_\alpha$ is of the form $\lambda\overline{X} . c_\beta$ where $\beta \in \{\iota, o\}$.) In particular, $\mathrm{cwff}_\alpha(\Sigma')\downarrow_{\beta\eta}$ and $\mathrm{cwff}_\alpha(\Sigma')\downarrow_\beta$ are non-empty for each type α.

We can now inductively define a map ρ from $\mathrm{wff}_\alpha(\Sigma)$ to $\mathrm{wff}_\alpha(\Sigma')$ which collapses terms to the smaller signature. For variables, let $\rho(X) := X$. For constants $w_\alpha \in \Sigma$ (including logical constants), let $\rho(w_\alpha) := \boldsymbol{C}'_\alpha$. For application and λ-abstraction, we simply use $\rho(\boldsymbol{FA}) := \rho(\boldsymbol{F})\rho(\boldsymbol{A})$ and $\rho(\lambda X . \boldsymbol{A}) := \lambda X . \rho(\boldsymbol{A})$. By induction on the formula $\boldsymbol{A}$, one can show $[\rho(\boldsymbol{B})/X]\rho(\boldsymbol{A}) \equiv \rho([\boldsymbol{B}/X]\boldsymbol{A})$ for any $\boldsymbol{A} \in \mathrm{wff}_\alpha(\Sigma)$, $\boldsymbol{B} \in \mathrm{wff}_\beta(\Sigma)$ and X_β. From this, one can show $\rho(\boldsymbol{A}) \equiv_{\beta\eta} \rho(\boldsymbol{B})$ whenever $\boldsymbol{A} \equiv_{\beta\eta} \boldsymbol{B}$ for every $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{wff}_\alpha(\Sigma)$. Note also that $\rho(\boldsymbol{A}') \equiv \boldsymbol{A}'$ for every $\boldsymbol{A}' \in \mathrm{wff}_\alpha(\Sigma')$.

We can construct a non-functional applicative structure using an indexing technique similar to Example 5.6. In this case, instead of indexing with $i \in \{0, 1\}$, we use terms in $\mathrm{cwff}_\alpha(\Sigma')\downarrow_*$ as indices. (Here $\boldsymbol{A}\downarrow_*$ means the β-normal form if $* \equiv \beta$ and the $\beta\eta$-normal form if $* \equiv \beta\eta$.) In essence, this index records some information about the "implementation" of the function. Note that $\mathrm{cwff}_\iota(\Sigma')\downarrow_* \equiv \{c_\iota\}$ and $\mathrm{cwff}_o(\Sigma')\downarrow_* \equiv \{c_o\}$. Let $\mathscr{D}_\iota := \{(c_\iota, 0)\}$ and $\mathscr{D}_o := \{(c_o, \mathtt{F})\} \cup \{(c_o, \mathsf{b}) \mid \mathsf{b} \in \mathscr{B}\}$. For function types, let $\mathscr{D}_{\alpha\to\beta}$ be the set of pairs $(\boldsymbol{F}'_{\alpha\to\beta}, f)$, where $\boldsymbol{F}' \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma')\downarrow_*$ and $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ is any function such that $f(\boldsymbol{A}', a) \equiv ((\boldsymbol{F}'\boldsymbol{A}')\downarrow_*, b)$ for some value b. Application is defined as in Example 5.6: $(\boldsymbol{F}, f)@\mathsf{a} := f(\mathsf{a})$. The construction of this applicative structure closely follows Andrews' υ-complexes in [1], except we have a very restricted signature Σ' which does not include logical constants.

To show that each domain is non-empty, we construct a particular element $\mathsf{c}^\alpha \in \mathscr{D}_\alpha$ for each type α. (This element will also be used to interpret parameters.) Let $\mathsf{c}^\iota := (c_\iota, 0)$, $\mathsf{c}^o := (c_o, \mathtt{F})$, and $\mathsf{c}^{\alpha\to\beta} := (\boldsymbol{C}'_{\alpha\to\beta}, k)$ where $k\colon \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ is the constant function $k(\mathsf{a}) := \mathsf{c}^\beta$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. The fact that $\mathsf{c}^{\alpha\to\beta} \in \mathscr{D}_{\alpha\to\beta}$ follows from $(\boldsymbol{C}'_{\alpha\to\beta}\boldsymbol{A})\!\downarrow_* \equiv \boldsymbol{C}'_\beta$.

One can see that the applicative structure is non-functional by noting $(\lambda X_\iota . X, f)$ and $(\lambda X_\iota . c_\iota, f)$ are distinct members of $\mathscr{D}_{\iota\to\iota}$, where f is the unique function taking $\mathscr{D}_\iota$ into itself. However, $(\lambda X_\iota . X, f)@\mathsf{c}^\iota \equiv \mathsf{c}^\iota \equiv (\lambda X_\iota . c_\iota, f)@\mathsf{c}^\iota$. In fact, once we define the evaluation function, this same example will show that property ξ will fail.

Let $\upsilon\colon \mathscr{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be $\upsilon((c_o, \mathtt{F})) := \mathtt{F}$ and $\upsilon((c_o, \mathsf{b})) := \mathtt{T}$ for each $\mathsf{b} \in \mathscr{B}$. This will be the valuation function on the model.

We only sketch the definition of the evaluation function $\mathscr{E}$ and the proof that this gives a model $\mathscr{M}^{*,\mathscr{B}} := (\mathscr{D}, @, \mathscr{E}, \upsilon)$. We can define $\mathscr{E}$ by induction on terms. First, we interpret parameters $w_\alpha \in \Sigma$ by $\mathscr{E}(w_\alpha) := \mathsf{c}^\alpha$. For logical constants $a_\alpha \in \Sigma$, we choose the first component of $\mathscr{E}(a_\alpha)$ to be $\boldsymbol{C}'_\alpha$ and the second component to be an appropriate function. We can define the witnesses q^α in a similar way and use these to interpret primitive equality, if it is present in the signature.

We are forced to let $\mathscr{E}_\varphi(X) := \varphi(X)$ and $\mathscr{E}_\varphi(\boldsymbol{FA}) := \mathscr{E}_\varphi(\boldsymbol{F})@\mathscr{E}_\varphi(\boldsymbol{A})$. For the λ-abstraction step, we choose $\mathscr{E}_\varphi(\lambda X_\alpha . \boldsymbol{B}_\beta) := ((\sigma(\rho(\lambda X . \boldsymbol{B}))\!\downarrow_*, f)$, where $f : \mathscr{D}_\alpha \longrightarrow \mathscr{D}_\beta$ satisfies $f(\mathsf{a}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{B})$ for all $\mathsf{a} \in \mathscr{D}_\alpha$ and σ is the substitution defined by letting $\sigma(Y)$ be the first component of $\varphi(Y)$ for each $Y \in \mathrm{free}(\lambda X . \boldsymbol{B})$. In order to show $\mathscr{E}$ is well-defined, one shows the first component of $\mathscr{E}_\varphi(\boldsymbol{A})$ is $(\sigma(\rho(\boldsymbol{A})))\!\downarrow_*$ (where σ is the substitution for $\mathrm{free}(\boldsymbol{A})$ defined from the first components of the values of φ) for every formula $\boldsymbol{A}$.

The fact that $\mathscr{E}$ evaluates variables and application properly is immediate from the definition. The fact that $\mathscr{E}_\varphi(\boldsymbol{A})$ depends only the free variables in $\boldsymbol{A}$ follows by an induction on the definition of $\mathscr{E}$. To show $\mathscr{E}$ respects β-conversion if $* \equiv \beta$ and $\beta\eta$-conversion if $* \equiv \beta\eta$ (so that the model will also satisfy property η), one first shows $\mathscr{E}$ respects a single $\beta[\eta]$-reduction, then does an induction on the position of the redex, and finally does an induction on the number of $\beta[\eta]$-reductions.

Once these details are checked, we know $\mathscr{M}^{*,\mathscr{B}}$ is a model (with primitive equality, if present) satisfying property $\mathfrak{q}$. We already know the model will not satisfy property $\mathfrak{f}$ since the applicative structure is not functional. We can also check that the model will not satisfy property ξ by considering $\mathscr{E}(\lambda X_\iota . X)$ and $\mathscr{E}(\lambda X_\iota . c_\iota)$. We know $\mathscr{E}(\lambda X_\iota . X) \not\equiv \mathscr{E}(\lambda X_\iota . c_\iota)$ since the first components ($(\lambda X_\iota . X)$ and $(\lambda X_\iota . c_\iota)$) are not equal. However, $\mathscr{D}_\iota$ has only one element, $\mathsf{c}^\iota \equiv (c_\iota, 0)$. So, we must have $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(X) \equiv \mathsf{c}^\iota \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(c_\iota)$ for every $\mathsf{a} \in \mathscr{D}_\iota$. This shows property ξ fails.

If $* \equiv \beta\eta$, then we have noted above that $\mathscr{E}$ respects $\beta\eta$-conversion. So, in this case, the model satisfies property η. If $* \equiv \beta$, then we can easily check $\mathscr{E}(\lambda F_{\iota\to\iota} X_\iota . FX) \not\equiv \mathscr{E}(\lambda F_{\iota\to\iota} . F)$ since the first components will differ. So, in this case, the model does not satisfy property η.

As in Example 5.6, if $\mathscr{B} \equiv \{\mathtt{T}\}$, then $\mathscr{M}^{\beta\mathfrak{b}} := \mathscr{M}^{\beta,\{\mathtt{T}\}}$ and $\mathscr{M}^{\beta\eta\mathfrak{b}} := \mathscr{M}^{\beta\eta,\{\mathtt{T}\}}$ satisfy property $\mathfrak{b}$. So, we know $\mathscr{M}^{\beta\mathfrak{b}} \in \mathfrak{M}_{\beta\mathfrak{b}} \setminus (\mathfrak{M}_{\beta\eta\mathfrak{b}} \cup \mathfrak{M}_{\beta\xi\mathfrak{b}})$ and $\mathscr{M}^{\beta\eta\mathfrak{b}} \in \mathfrak{M}_{\beta\eta\mathfrak{b}} \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$. If $\mathscr{B} \equiv \{\mathtt{T}, \mathsf{b}\}$ where b is any value with $\mathsf{b} \notin \{\mathtt{T}, \mathtt{F}\}$, then the models $\mathscr{M}^\beta := \mathscr{M}^{\beta,\{\mathtt{T},\mathsf{b}\}}$ and $\mathscr{M}^{\beta\eta} := \mathscr{M}^{\beta\eta,\{\mathtt{T},\mathsf{b}\}}$ do not satisfy property $\mathfrak{b}$, so $\mathscr{M}^\beta \in \mathfrak{M}_\beta \setminus (\mathfrak{M}_{\beta\eta} \cup \mathfrak{M}_{\beta\xi} \cup \mathfrak{M}_{\beta\mathfrak{b}})$ and $\mathscr{M}^{\beta\eta} \in \mathfrak{M}_{\beta\eta} \setminus (\mathfrak{M}_{\beta\mathfrak{f}} \cup \mathfrak{M}_{\beta\eta\mathfrak{b}})$.

In particular, the models $\mathscr{M}^{\beta\eta}$ and $\mathscr{M}^{\beta\eta\mathfrak{b}}$ show that respecting η-conversion does not guarantee strong functional extensionality.

Thus we have given (sketches of) concrete models that distinguish model classes and shown that the inclusions between the $\mathfrak{M}_*$ model classes in Figure 1 are proper.

§6. Model existence. In this section we present the model existence theorems for the different semantical notions introduced in Section 3. The model existence theorems have the following form, where $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$:

THEOREM (Model existence). *For a given abstract consistency class* $\Gamma_\Sigma \in \mathfrak{Acc}_*$ (*cf. Definition* 6.7) *and a set* $\Phi \in \Gamma_\Sigma$ *there is a* Σ*-model* $\mathscr{M}$ *of* Φ, *such that* $\mathscr{M} \in \mathfrak{M}_*$ (*cf. Definition* 3.49).

The most important tools used in the proofs of the model existence theorems are the so-called Σ-Hintikka sets. These sets allow computations that resemble those in the considered semantical structures (e.g., Henkin models) and allow us to construct appropriate valuations for the term evaluation $\mathscr{TE}(\Sigma)^\beta$ defined in Definition 3.35. The key step in the proof of the model existence theorems is an extension lemma, which guarantees a Σ-Hintikka set $\mathscr{H}$ for any sufficiently Σ-pure set of sentences Φ in Γ_Σ.

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

DEFINITION 6.1 (Compactness). Let $\mathscr{C}$ be a class of sets.

(1) $\mathscr{C}$ is called *closed under subsets* if for any sets S and T, $S \in \mathscr{C}$ whenever $S \subseteq T$ and $T \in \mathscr{C}$.
(2) $\mathscr{C}$ is called *compact* if for every set S we have $S \in \mathscr{C}$ iff every finite subset of S is a member of $\mathscr{C}$.

LEMMA 6.2. *If* $\mathscr{C}$ *is compact, then* $\mathscr{C}$ *is closed under subsets.*

PROOF. Suppose $S \subseteq T$ and $T \in \mathscr{C}$. Every finite subset A of S is a finite subset of T, and since $\mathscr{C}$ is compact we know that $A \in \mathscr{C}$. Thus $S \in \mathscr{C}$. ⊣

We will now introduce a technical side-condition that ensures that we always have enough witness constants.

DEFINITION 6.3 (Sufficiently Σ-pure). Let Σ be a signature and Φ be a set of Σ-sentences. Φ is called *sufficiently* Σ*-pure* if for each type α there is a set $\mathscr{P}_\alpha \subseteq \Sigma_\alpha$ of parameters with equal cardinality to $\mathrm{wff}_\alpha(\Sigma)$, such that the elements of $\mathscr{P}_\alpha$ do not occur in the sentences of Φ.

This can be obtained in practice by enriching the signature with spurious parameters. Another way would be to use specially marked variables (which may never be instantiated) as in [36]. Note that for any set to be sufficiently Σ-pure, Σ_α must be infinite for each type α, since we have assumed that $\mathscr{V}_\alpha \subseteq \mathrm{wff}(\Sigma)$ are infinite. Recall that in Remark 3.16 we assumed every Σ_α has a common (infinite) cardinality $\aleph_s$ for every type α. (One could easily show that no set of Σ-sentences could be sufficiently pure if, for example, Σ_ι is countable while $\Sigma_{\iota\to\iota}$ is uncountable. In such a case $\mathrm{wff}_\alpha(\Sigma)$ is uncountable for every type α so one could not satisfy the sufficient purity condition at type ι.)

NOTATION 6.4. For reasons of legibility we will write $S * a$ for $S \cup \{a\}$, where S is a set. We will use this notation with the convention that $*$ associates to the left.

DEFINITION 6.5 (Properties for abstract consistency classes). Let Γ_Σ be a class of sets of Σ-sentences. We define the following properties of Γ_Σ, where $\Phi \in \Gamma_\Sigma$, α, $\beta \in \mathscr{T}$, $\boldsymbol{A}$, $\boldsymbol{B} \in \mathrm{cwff}_o$, $\boldsymbol{F} \in \mathrm{cwff}_{\alpha\to o}$, and $\boldsymbol{G}$, $\boldsymbol{H}$, $(\lambda X_\alpha.\boldsymbol{M})$, $(\lambda X_\alpha.\boldsymbol{N}) \in \mathrm{cwff}_{\alpha\to\beta}$ are arbitrary.

∇_c: If $\boldsymbol{A}$ is atomic, then $\boldsymbol{A} \notin \Phi$ or $\neg\boldsymbol{A} \notin \Phi$.
$\nabla_\neg$: If $\neg\neg\boldsymbol{A} \in \Phi$, then $\Phi * \boldsymbol{A} \in \Gamma_\Sigma$.
∇_β: If $\boldsymbol{A} \equiv_\beta \boldsymbol{B}$ and $\boldsymbol{A} \in \Phi$, then $\Phi * \boldsymbol{B} \in \Gamma_\Sigma$.
∇_η: If $\boldsymbol{A} \equiv_{\beta\eta} \boldsymbol{B}$ and $\boldsymbol{A} \in \Phi$, then $\Phi * \boldsymbol{B} \in \Gamma_\Sigma$.
$\nabla_\vee$: If $\boldsymbol{A} \vee \boldsymbol{B} \in \Phi$, then $\Phi * \boldsymbol{A} \in \Gamma_\Sigma$ or $\Phi * \boldsymbol{B} \in \Gamma_\Sigma$.
$\nabla_\wedge$: If $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \Phi$, then $\Phi * \neg\boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$.
$\nabla_\forall$: If $\Pi^\alpha \boldsymbol{F} \in \Phi$, then $\Phi * \boldsymbol{F}\boldsymbol{W} \in \Gamma_\Sigma$ for each $\boldsymbol{W} \in \mathrm{cwff}_\alpha$.
$\nabla_\exists$: If $\neg\Pi^\alpha \boldsymbol{F} \in \Phi$, then $\Phi * \neg(\boldsymbol{F}w) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
$\nabla_\flat$: If $\neg(\boldsymbol{A} \doteq^o \boldsymbol{B}) \in \Phi$, then $\Phi * \boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$ or $\Phi * \neg\boldsymbol{A} * \boldsymbol{B} \in \Gamma_\Sigma$.
∇_ξ: If $\neg(\lambda X_\alpha.\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X_\alpha.\boldsymbol{N}) \in \Phi$, then $\Phi * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N}) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
$\nabla_\dagger$: If $\neg(\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}) \in \Phi$, then $\Phi * \neg(\boldsymbol{G}w \doteq^\beta \boldsymbol{H}w) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
∇_{sat}: Either $\Phi * \boldsymbol{A} \in \Gamma_\Sigma$ or $\Phi * \neg\boldsymbol{A} \in \Gamma_\Sigma$.

For the optional case of primitive equality, i.e., when $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ for all types α, we now add a set of further properties. While our first choice will be to combine the $\nabla^r_=$ property with $\nabla^{\doteq}_=$, we will later show that other pair combinations from this set are equivalent.

DEFINITION 6.6 (Properties for abstract consistency classes). Suppose $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ for all types α. Let Γ_Σ be a class of sets of Σ-sentences. We define for $\Phi \in \Gamma_\Sigma$, $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_\alpha$ and $\boldsymbol{F} \in \mathrm{cwff}_o$ where $\boldsymbol{F}$ has a subterm of type α at position p:

$\nabla^r_=$: $\neg(\boldsymbol{A} =^\alpha \boldsymbol{A}) \notin \Phi$.
$\nabla^s_=$: If $\boldsymbol{F}[\boldsymbol{A}]_p \in \Phi$ and $\boldsymbol{A} =^\alpha \boldsymbol{B} \in \Phi$, then $\Phi * \boldsymbol{F}[\boldsymbol{B}]_p \in \Gamma_\Sigma$.[9]
$\nabla^{\doteq}_=$: If $\boldsymbol{A} =^\alpha \boldsymbol{B} \in \Phi$, then $\Phi * \boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \Gamma_\Sigma$.
$\nabla^{=}_{\doteq}$: If $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \Phi$, then $\Phi * \boldsymbol{A} =^\alpha \boldsymbol{B} \in \Gamma_\Sigma$.
$\nabla^{\doteq\,-}_{=\,-}$: If $\neg(\boldsymbol{A} =^\alpha \boldsymbol{B}) \in \Phi$, then $\Phi * \neg(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \Gamma_\Sigma$.
$\nabla^{=\,-}_{\doteq\,-}$: If $\neg(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \Phi$, then $\Phi * \neg(\boldsymbol{A} =^\alpha \boldsymbol{B}) \in \Gamma_\Sigma$.

DEFINITION 6.7 (Abstract consistency classes). Let Σ be a signature and Γ_Σ be a class of sets of Σ-sentences that is closed under subsets. If $\nabla_c, \nabla_\neg, \nabla_\beta, \nabla_\vee, \nabla_\wedge, \nabla_\forall$ and $\nabla_\exists$ are valid for Γ_Σ, then Γ_Σ is called an *abstract consistency class* for Σ-models. Furthermore, when $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ for all types α and the properties $\nabla^r_=$ and $\nabla^{\doteq}_=$ are valid then Γ_Σ is called an *abstract consistency class with primitive equality*. In the following we often simply use the phrase abstract consistency class to refer to an abstract consistency class with or without primitive equality. We will denote

[9] Although this resembles Lemma 3.25 which required property ξ, it is far weaker since $\boldsymbol{A}$ and $\boldsymbol{B}$ must be closed.

the collection of abstract consistency classes (with primitive equality) by $\mathfrak{Acc}_\beta$. Similarly, we introduce the following collections of specialized abstract consistency classes (with primitive equality): $\mathfrak{Acc}_{\beta\eta}$, $\mathfrak{Acc}_{\beta\xi}$, $\mathfrak{Acc}_{\beta\mathfrak{f}}$, $\mathfrak{Acc}_{\beta\mathfrak{b}}$, $\mathfrak{Acc}_{\beta\eta\mathfrak{b}}$, $\mathfrak{Acc}_{\beta\xi\mathfrak{b}}$, $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$, where we indicate by indices which additional properties from $\{\nabla_\eta, \nabla_\xi, \nabla_\mathfrak{f}, \nabla_\mathfrak{b}\}$ are required.

REMARK 6.8. If primitive equality is not in the signature, $\mathfrak{Acc}_\beta$ corresponds to the abstract consistency property discussed by Andrews in [1]. The only (technical) differences correspond to $\alpha\beta$-conversion. In [1], α-conversion is handled in the ∇_β rule using α-standardized forms. Also, we have defined the ∇_β rule to work with β-conversion instead of β-reduction. We prefer this stronger version of ∇_β over the weaker option "If $\boldsymbol{A} \in \Phi$, then $\Phi * \boldsymbol{A}\downarrow_\beta \in \Gamma_\Sigma$" since it helps to avoid the use of ∇_{sat} in several proofs below. (Note that ∇_β follows from the weaker option and ∇_{sat}.) Furthermore, in practical applications, e.g., proving completeness of calculi, the stronger property is typically as easy to validate as the weaker one. An analogous argument applies to ∇_η.

REMARK 6.9. While the work presented in this article is based on the choice of the primitive logical connectives $\neg, \vee$, and Π^α (and possibly primitive equality), a means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [22]. It is clearly possible to achieve such a generalization for our framework as well. This can be done in straightforward manner: $\nabla_\wedge$ becomes an α-property, $\nabla_\vee$ becomes a β-property, $\nabla_\forall$ becomes a γ-property, and $\nabla_\exists$ becomes a δ-property. Thus they will have the following form:

α-case: If $\alpha \in \Phi$, then $\Phi * \alpha_1 * \alpha_2 \in \Gamma_\Sigma$.
β-case: If $\beta \in \Phi$, then $\Phi * \beta_1 \in \Gamma_\Sigma$ or $\Phi * \beta_2 \in \Gamma_\Sigma$.
γ-case: If $\gamma \in \Phi$, then $\Phi * \gamma\boldsymbol{W} \in \Gamma_\Sigma$ for each $\boldsymbol{W} \in \mathrm{cwff}_\alpha$.
δ-case: If $\delta \in \Phi$, then $\Phi * \delta w \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma$ which does not occur in any sentence of Φ.

We often refer to property ∇_c as "atomic consistency". The next lemma shows that we also have the corresponding property for non-atoms.

LEMMA 6.10 (Non-atomic consistency). *Let Γ_Σ be an abstract consistency class and $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)$, then for all $\Phi \in \Gamma_\Sigma$ we have $\boldsymbol{A} \notin \Phi$ or $\neg\boldsymbol{A} \notin \Phi$.*

PROOF following a similar argument in [1], Lemma 3.3.3. If for some $\Phi \in \Gamma_\Sigma$ and $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)$ we have $\boldsymbol{A} \in \Phi$ and $\neg\boldsymbol{A} \in \Phi$, then $\{\boldsymbol{A}, \neg\boldsymbol{A}\} \in \Gamma_\Sigma$ since Γ_Σ is closed under subsets. Furthermore, using ∇_β and closure under subsets we can assume such an $\boldsymbol{A}$ is β-normal. We prove $\{\boldsymbol{A}, \neg\boldsymbol{A}\} \notin \Gamma_\Sigma$ for any β-normal $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)$ by induction on the number of logical constants in $\boldsymbol{A}$.

If $\boldsymbol{A}$ is atomic (which includes primitive equations), this follows immediately from ∇_c. Suppose $\boldsymbol{A} \equiv \neg\boldsymbol{B}$ for some $\boldsymbol{B} \in \mathrm{cwff}_o(\Sigma)$ and $\{\neg\boldsymbol{B}, \neg\neg\boldsymbol{B}\} \in \Gamma_\Sigma$. By $\nabla_\neg$ and closure under subsets, we have $\{\neg\boldsymbol{B}, \boldsymbol{B}\} \in \Gamma_\Sigma$, contradicting the induction hypothesis for $\boldsymbol{B}$. Suppose $\boldsymbol{A} \equiv \boldsymbol{B} \vee \boldsymbol{C}$ for some $\boldsymbol{B}, \boldsymbol{C} \in \mathrm{cwff}_o(\Sigma)$ and $\{\boldsymbol{B} \vee \boldsymbol{C}, \neg(\boldsymbol{B} \vee \boldsymbol{C})\} \in \Gamma_\Sigma$. By $\nabla_\vee$, $\nabla_\wedge$ and closure under subsets, we have either $\{\boldsymbol{B}, \neg\boldsymbol{B}\} \in \Gamma_\Sigma$ or $\{\boldsymbol{C}, \neg\boldsymbol{C}\} \in \Gamma_\Sigma$, contradicting the induction hypotheses for $\boldsymbol{B}$ and $\boldsymbol{C}$. Suppose $\boldsymbol{A} \equiv \Pi^\alpha\boldsymbol{B}$ for some $\boldsymbol{B} \in \mathrm{cwff}_{\alpha\to o}(\Sigma)$ and $\{\Pi^\alpha\boldsymbol{B}, \neg(\Pi^\alpha\boldsymbol{B})\} \in \Gamma_\Sigma$. Since Σ_α is assumed to be infinite (by Remark 3.16), there is a parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in $\boldsymbol{A}$. Since

w is a parameter, the sentence $\boldsymbol{B}w$ clearly has one less logical constant than $\Pi^\alpha \boldsymbol{B}$. However, we cannot directly apply the induction hypothesis as $\boldsymbol{B}w$ may not be β-normal. Since $\boldsymbol{B}$ is β-normal, the only way $\boldsymbol{B}w$ can fail to be β-normal is if $\boldsymbol{B}$ has the form $\lambda X_\alpha.\boldsymbol{C}$ for some $\boldsymbol{C} \in \mathrm{wff}_o(\Sigma)$ where $\mathrm{free}(\boldsymbol{C}) \subseteq \{X_\alpha\}$. In this case, it is easy to show that the reduct $[w/X]\boldsymbol{C}$ is β-normal and contains the same number of logical constants as $\boldsymbol{B}$. In either case, we can let $\boldsymbol{N}$ be the β-normal form of $\boldsymbol{B}w$ and apply the induction hypothesis to obtain $\{\boldsymbol{N}, \neg\boldsymbol{N}\} \notin \Gamma_\Sigma$. On the other hand, $\nabla_\exists$, $\nabla_\forall$, ∇_β and closure under subsets implies $\{\boldsymbol{N}, \neg\boldsymbol{N}\} \in \Gamma_\Sigma$, a contradiction. ⊣

REMARK 6.11. Note that for the connectives $\vee$ and Π^α there is a positive and a negative condition given in the definition above, namely $\nabla_\vee/\nabla_\wedge$ for $\vee$ and $\nabla_\forall/\nabla_\exists$ for Π^α. For $\doteq^o$ and $\doteq^{\alpha\to\beta}$ the situation is different since we need only conditions for the negative cases. Positive counterparts can be inferred by expanding the Leibniz definition of equality (cf. Lemma 6.12).

LEMMA 6.12 (Leibniz equality). *Let Γ_Σ be an abstract consistency class. The following properties are valid for all $\Phi \in \Gamma_\Sigma$, $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_o(\Sigma)$, $\boldsymbol{C} \in \mathrm{cwff}_\alpha(\Sigma)$ and $\boldsymbol{F}, \boldsymbol{G} \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma)$.*

$\nabla^r_{\doteq}$: $\neg(\boldsymbol{C} \doteq^\alpha \boldsymbol{C}) \notin \Phi$.

$\nabla^{\to}_{\doteq}$: *If* $\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G} \in \Phi$, *then* $\Phi * \boldsymbol{FW} \doteq^\beta \boldsymbol{GW} \in \Gamma_\Sigma$ *for any closed* $\boldsymbol{W} \in \mathrm{cwff}_\alpha(\Sigma)$.

$\nabla^o_{\doteq}$: *If* $\boldsymbol{A} \doteq^o \boldsymbol{B} \in \Phi$, *then* $\Phi * \boldsymbol{A} * \boldsymbol{B} \in \Gamma_\Sigma$ *or* $\Phi * \neg\boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$.

PROOF. To show $\nabla^r_{\doteq}$, assume $\neg(\boldsymbol{C} \doteq \boldsymbol{C}) \in \Phi$. By subset closure $\{\neg(\boldsymbol{C} \doteq \boldsymbol{C})\} \in \Gamma_\Sigma$ and by $\nabla_\exists$ with some parameter p which does not occur in $\boldsymbol{C}$ and ∇_β we get $\{\neg(\boldsymbol{C} \doteq \boldsymbol{C}), \neg(\neg p\boldsymbol{C} \vee p\boldsymbol{C})\} \in \Gamma_\Sigma$. The contradiction follows by $\nabla_\wedge$, $\nabla_\neg$ and ∇_c. So, $\nabla^r_{\doteq}$ holds.

To show $\nabla^{\to}_{\doteq}$, suppose $\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G} \in \Phi$. By application of $\nabla_\forall$ with $\lambda X_{\alpha\to\beta}.\boldsymbol{FW} \doteq X\boldsymbol{W}$ and ∇_β we have $\Phi * (\neg(\boldsymbol{FW} \doteq \boldsymbol{FW}) \vee \boldsymbol{FW} \doteq \boldsymbol{GW}) \in \Gamma_\Sigma$. By $\nabla_\vee$ and subset closure we get $\Phi * \neg(\boldsymbol{FW} \doteq \boldsymbol{FW}) \in \Gamma_\Sigma$ or $\Phi * \boldsymbol{FW} \doteq \boldsymbol{GW} \in \Gamma_\Sigma$. The latter proves the assertion since the first option is ruled out by $\nabla^r_{\doteq}$ (shown above).

To show $\nabla^o_{\doteq}$, suppose $\boldsymbol{A} \doteq^o \boldsymbol{B} \in \Phi$. Applying $\nabla_\forall$ with $\lambda Y.Y$ we have $\Phi * (\lambda P_{o\to o}.\neg P\boldsymbol{A} \vee P\boldsymbol{B})(\lambda Y.Y) \in \Gamma_\Sigma$. By ∇_β and subset closure we get $\Phi * \neg\boldsymbol{A} \vee \boldsymbol{B} \in \Gamma_\Sigma$. Similarly, we further derive by $\nabla_\forall$ with $\lambda Y.\neg Y$, ∇_β, and subset closure that $\Phi * \neg\boldsymbol{A} \vee \boldsymbol{B} * \neg\neg\boldsymbol{A} \vee \neg\boldsymbol{B} \in \Gamma_\Sigma$. By applying $\nabla_\vee$ twice and subset closure we get the following four options: (i) $\Phi * \neg\boldsymbol{A} * \neg\neg\boldsymbol{A} \in \Gamma_\Sigma$, (ii) $\Phi * \neg\boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$, (iii) $\Phi * \boldsymbol{B} * \neg\neg\boldsymbol{A} \in \Gamma_\Sigma$, or (iv) $\Phi * \boldsymbol{B} * \neg\boldsymbol{B} \in \Gamma_\Sigma$. Cases (i) and (iv) are ruled out by non-atomic consistency. In case (iii) we furthermore get by $\nabla_\neg$ and subset closure that $\Phi * \boldsymbol{B} * \boldsymbol{A} \in \Gamma_\Sigma$. Thus, $\Phi * \neg\boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$ or $\Phi * \boldsymbol{B} * \boldsymbol{A} \in \Gamma_\Sigma$. ⊣

We could easily add respective properties for symmetry, transitivity, and congruence to the previous lemma. They can be shown analogously, i.e., they also follow from the properties of Leibniz equality.

In contrast to [1], we work with saturated abstract consistency classes in order to simplify the proofs of the model existence theorems. For a discussion of the consequences of this decision, see Section 8.2.

DEFINITION 6.13 (Saturatedness). We call an abstract consistency class Γ_Σ saturated if it satisfies ∇_{sat}.

REMARK 6.14. Clearly, not all abstract consistency classes are saturated, since the empty set is one that is not ($\mathrm{cwff}_o(\Sigma)$ is certainly non-empty since $\forall P_o.P \in \mathrm{cwff}_o(\Sigma)$).

REMARK 6.15. The saturation condition ∇_{sat} can be very difficult to verify in practice. For example, showing that an abstract consistency class induced from a sequent calculus (as in [1]) is saturated corresponds to showing cut-elimination (cf. [12]). Since Andrews [1] did not use saturation, he could use his results to give a model-theoretic proof of cut-elimination for a sequent calculus. We cannot use the results of this article to obtain similar cut-elimination results.

We now investigate derived properties of primitive equality.

LEMMA 6.16 (Primitive equality). *Let Γ_Σ be an abstract consistency class with primitive equality, i.e., $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ for all types $\alpha \in \mathscr{T}$, where $\nabla^r_=$ and $\nabla^{\doteq}_=$ hold. Then $\nabla^=_{\doteq}$ and $\nabla^s_=$ are valid. Furthermore, $\nabla^{\doteq^-}_{=^-}$ and $\nabla^{=^-}_{\doteq^-}$ are valid if Γ_Σ is saturated.*

PROOF. To show $\nabla^=_{\doteq}$ we derive from $(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \Phi$ by $\nabla_\forall$ with $\lambda X_\alpha.\boldsymbol{A} =^\alpha X$, ∇_β, and subset closure that $\Phi * \neg(\boldsymbol{A} = \boldsymbol{A}) \vee \boldsymbol{A} = \boldsymbol{B} \in \Gamma_\Sigma$. By $\nabla_\vee$ and subset closure we get $\Phi * \neg(\boldsymbol{A} = \boldsymbol{A}) \in \Gamma_\Sigma$ or $\Phi * \boldsymbol{A} = \boldsymbol{B} \in \Gamma_\Sigma$. The assertion follows from the latter option since the former is ruled out by $\nabla^r_=$.

In order to show $\nabla^s_=$ let $\boldsymbol{F}[\boldsymbol{A}]_p \in \Phi$, we derive from $\boldsymbol{A} =^\alpha \boldsymbol{B} \in \Phi$ by $\nabla^{\doteq}_=$ that $\Phi * (\boldsymbol{A} \doteq \boldsymbol{B}) \in \Gamma_\Sigma$. By $\nabla_\forall$ with $\lambda X.\boldsymbol{F}[X]_p$ (where $X \in \mathscr{V}_\alpha$ does not occur bound in $\boldsymbol{F}[\boldsymbol{A}]_p$), ∇_β, and subset closure we furthermore get that $\Phi * (\neg\boldsymbol{F}[\boldsymbol{A}]_p \vee \boldsymbol{F}[\boldsymbol{B}]_p) \in \Gamma_\Sigma$. Application of $\nabla_\vee$ and subset closure gives us $\Phi * \neg\boldsymbol{F}[\boldsymbol{A}]_p \in \Gamma_\Sigma$ or $\Phi * \boldsymbol{F}[\boldsymbol{B}]_p \in \Gamma_\Sigma$. The assertion follows from the latter option since the former is ruled out by $\boldsymbol{F}[\boldsymbol{A}]_p \in \Phi$ and non-atomic consistency.

The straightforward proof for $\nabla^{=^-}_{\doteq^-}$ employs saturation, $\nabla^{\doteq}_=$, and non-atomic consistency. Similarly, the proof for $\nabla^{\doteq^-}_{=^-}$ employs saturation, $\nabla^=_{\doteq}$, and atomic consistency. ⊣

The next theorem provides some alternatives to our choice of $\nabla^{\doteq}_=$ and $\nabla^r_=$ in the definition of abstract consistency classes with primitive equality provided that saturation holds. In practical applications the user may therefore choose the combination that suits best.

THEOREM 6.17 (Alternative properties for primitive equality). *Let Γ_Σ be an abstract consistency class and let $=^\alpha \in \Sigma_{\alpha\to\alpha\to o}$ for all types $\alpha \in \mathscr{T}$. If Γ_Σ is saturated and validates one of the following combinations of properties, then it also validates $\nabla^{\doteq}_=$ and $\nabla^r_=$. The combinations are*:

(1) $\nabla^s_=$ *and* $\nabla^r_=$.
(2) $\nabla^{\doteq}_=$ *and* $\nabla^=_{\doteq}$.
(3) $\nabla^{\doteq^-}_{=^-}$ *and* $\nabla^{=^-}_{\doteq^-}$.

PROOF. To prove (1) we only have to show $\nabla^{\doteq}_=$. Let $(\boldsymbol{A} = \boldsymbol{B}) \in \Phi$ and suppose $\Phi * (\boldsymbol{A} \doteq \boldsymbol{B}) \notin \Gamma_\Sigma$. Then by saturation $\Phi * \neg(\boldsymbol{A} \doteq \boldsymbol{B}) \in \Gamma_\Sigma$ and by application of $\nabla^s_=$ we get a contradiction to $\nabla^r_{\doteq}$ (cf. Lemma 6.12).

To prove (2) we only have to show $\nabla^r_=$. Since $\Phi * \neg(\boldsymbol{A} \doteq \boldsymbol{A}) \notin \Gamma_\Sigma$ by $\nabla^r_{\doteq}$ we get by saturation $\Phi * \boldsymbol{A} \doteq \boldsymbol{A} \in \Gamma_\Sigma$. By $\nabla^=_{\doteq}$ and subset closure, we have $\Phi * \boldsymbol{A} = \boldsymbol{A} \in \Gamma_\Sigma$. By atomic consistency, we have $\neg(\boldsymbol{A} = \boldsymbol{A}) \notin \Phi$.

For (3) we first show $\nabla^r_{=}$. Suppose $\neg(\boldsymbol{A} = \boldsymbol{A}) \in \Phi$. Then by $\nabla^{\doteq\,-}_{=\,-}$ we get $\Phi * \neg(\boldsymbol{A} \doteq \boldsymbol{A}) \in \Gamma_\Sigma$ contradicting $\nabla^r_{\doteq}$. To show $\nabla^{\doteq}_{=}$ let $\boldsymbol{A} = \boldsymbol{B} \in \Phi$ and suppose $\Phi * \boldsymbol{A} \doteq \boldsymbol{B} \notin \Gamma_\Sigma$. By saturation we get $\Phi * \neg(\boldsymbol{A} \doteq \boldsymbol{B}) \in \Gamma_\Sigma$ and by application of $\nabla^{=\,-}_{\doteq\,-}$ we get a contradiction to atomic consistency. ⊣

LEMMA 6.18 (Compactness of abstract consistency classes). *For each abstract consistency class Γ_Σ there exists a compact abstract consistency class Γ'_Σ satisfying the same ∇_* properties such that $\Gamma_\Sigma \subseteq \Gamma'_\Sigma$.*

PROOF (following and extending [6], Proposition 2506). We choose $\Gamma'_\Sigma := \{\, \Phi \subseteq \mathrm{cwff}_o \mid$ every finite subset of Φ is in $\Gamma_\Sigma \,\}$. Now suppose that $\Phi \in \Gamma_\Sigma$. Γ_Σ is closed under subsets, so every finite subset of Φ is in Γ_Σ and thus $\Phi \in \Gamma'_\Sigma$. Hence $\Gamma_\Sigma \subseteq \Gamma'_\Sigma$.

Next let us show that Γ'_Σ is compact. Suppose $\Phi \in \Gamma'_\Sigma$ and Ψ is an arbitrary finite subset of Φ. By definition of Γ'_Σ all finite subsets of Φ are in Γ_Σ and therefore $\Psi \in \Gamma'_\Sigma$. Thus all finite subsets of Φ are in Γ'_Σ whenever Φ is in Γ'_Σ. On the other hand, suppose all finite subsets of Φ are in Γ'_Σ. Then by the definition of Γ'_Σ the finite subsets of Φ are also in Γ_Σ, so $\Phi \in \Gamma'_\Sigma$. Thus Γ'_Σ is compact. Note that by Lemma 6.2 we have that Γ'_Σ is closed under subsets.

Next we show that if Γ_Σ satisfies ∇_*, then Γ'_Σ satisfies ∇_*.

∇_c: Let $\Phi \in \Gamma'_\Sigma$ and suppose there is an atom $\boldsymbol{A}$, such that $\{\boldsymbol{A}, \neg\boldsymbol{A}\} \subseteq \Phi$. $\{\boldsymbol{A}, \neg\boldsymbol{A}\}$ is clearly a finite subset of Φ and hence $\{\boldsymbol{A}, \neg\boldsymbol{A}\} \in \Gamma_\Sigma$ contradicting ∇_c for Γ_Σ.

$\nabla_\neg$: Let $\Phi \in \Gamma'_\Sigma$, $\neg\neg\boldsymbol{A} \in \Phi$, Ψ be any finite subset of $\Phi * \boldsymbol{A}$, and $\Theta := (\Psi \setminus \{\boldsymbol{A}\}) * \neg\neg\boldsymbol{A}$. Θ is a finite subset of Φ, so $\Theta \in \Gamma_\Sigma$. Since Γ_Σ is an abstract consistency class and $\neg\neg\boldsymbol{A} \in \Theta$, we get $\Theta * \boldsymbol{A} \in \Gamma_\Sigma$ by $\nabla_\neg$ for Γ_Σ. We know that $\Psi \subseteq \Theta * \boldsymbol{A}$ and Γ_Σ is closed under subsets, so $\Psi \in \Gamma_\Sigma$. Thus every finite subset Ψ of $\Phi * \boldsymbol{A}$ is in Γ_Σ and therefore by definition $\Phi * \boldsymbol{A} \in \Gamma'_\Sigma$.

$\nabla_\beta, \nabla_\eta, \nabla_\vee, \nabla_\wedge, \nabla_\forall, \nabla_\exists$: Analogous to $\nabla_\neg$.

∇_ξ: Let $\Phi \in \Gamma'_\Sigma$, $\neg(\lambda X_\alpha.\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X.\boldsymbol{N}) \in \Phi$ and Ψ be any finite subset of $\Phi * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})$, where $w \in \Sigma_\alpha$ is a parameter that does not occur in any sentence of Φ. We show that $\Psi \in \Gamma_\Sigma$. Clearly $\Theta := (\Psi \setminus \{\neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})\}) * \neg(\lambda X.\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X.\boldsymbol{N})$ is a finite subset of Φ and therefore $\Theta \in \Gamma_\Sigma$. Since Γ_Σ satisfies ∇_ξ and $\neg(\lambda X.\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X.\boldsymbol{N}) \in \Theta$, we have $\Theta * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N}) \in \Gamma_\Sigma$. Furthermore, $\Psi \subseteq \Theta * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})$ and Γ_Σ is closed under subsets, so $\Psi \in \Gamma_\Sigma$. Thus every finite subset Ψ of $\Phi * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})$ is in Γ_Σ, and therefore by definition we have $\Phi * \neg([w/X]\boldsymbol{M} \doteq^\alpha [w/X]\boldsymbol{N}) \in \Gamma'_\Sigma$.

$\nabla_\mathfrak{f}$: Analogous to ∇_ξ.

$\nabla_\mathfrak{b}$: Let $\Phi \in \Gamma'_\Sigma$ with $\neg(\boldsymbol{A} \doteq \boldsymbol{B}) \in \Phi$. Assume $\Phi * \boldsymbol{A} * \neg\boldsymbol{B} \notin \Gamma_\Sigma$ and $\Phi * \neg\boldsymbol{A} * \boldsymbol{B} \notin \Gamma_\Sigma$. Then there exists finite subsets Φ_1 and Φ_2 of Φ, such that $\Phi_1 * \boldsymbol{A} * \neg\boldsymbol{B} \notin \Gamma_\Sigma$ and $\Phi_2 * \neg\boldsymbol{A} * \boldsymbol{B} \notin \Gamma_\Sigma$. Now we choose $\Phi_3 := \Phi_1 \cup \Phi_2 * \neg(\boldsymbol{A} \doteq \boldsymbol{B})$. Obviously Φ_3 is a finite subset of Φ and therefore $\Phi_3 \in \Gamma_\Sigma$. Since Γ_Σ satisfies $\nabla_\mathfrak{b}$, we have that $\Phi_3 * \boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$ or $\Phi_3 * \neg\boldsymbol{A} * \boldsymbol{B} \in \Gamma_\Sigma$. From this and the fact that Γ_Σ is closed under subsets we get that $\Phi_1 * \boldsymbol{A} * \neg\boldsymbol{B} \in \Gamma_\Sigma$ or $\Phi_2 * \neg\boldsymbol{A} * \boldsymbol{B} \in \Gamma_\Sigma$, which contradicts our assumption.

∇_{sat}: Let $\Phi \in \Gamma'_\Sigma$. Assume neither $\Phi * \boldsymbol{A}$ nor $\Phi * \neg\boldsymbol{A}$ is in Γ'_Σ. Then there are finite subsets Φ_1 and Φ_2 of Φ, such that $\Phi_1 * \boldsymbol{A} \notin \Gamma_\Sigma$ and $\Phi_2 * \neg\boldsymbol{A} \notin \Gamma_\Sigma$. As $\Psi := \Phi_1 \cup \Phi_2$ is a finite subset of Φ, we have $\Psi \in \Gamma_\Sigma$. Furthermore,

$\Psi * \boldsymbol{A} \in \Gamma_\Sigma$ or $\Psi * \neg\boldsymbol{A} \in \Gamma_\Sigma$ because Γ_Σ is saturated. Γ_Σ is closed under subsets, so $\Phi_1 * \boldsymbol{A} \in \Gamma_\Sigma$ or $\Phi_2 * \neg\boldsymbol{A} \in \Gamma_\Sigma$. This is a contradiction, so we can conclude that if $\Phi \in \Gamma'_\Sigma$, then $\Phi * \boldsymbol{A} \in \Gamma'_\Sigma$ or $\Phi * \neg\boldsymbol{A} \in \Gamma'_\Sigma$.

In case primitive equality is present in the signature, we check the corresponding properties.

$\nabla^r_=$: Let $\Phi \in \Gamma'_\Sigma$ and assume $\neg(\boldsymbol{A} =^\alpha \boldsymbol{A}) \in \Phi$. $\{\neg(\boldsymbol{A} =^\alpha \boldsymbol{A})\}$ is clearly a finite subset of Φ and hence $\{\neg(\boldsymbol{A} =^\alpha \boldsymbol{A})\} \in \Gamma_\Sigma$ contradicting $\nabla^r_=$ in Γ_Σ.

$\nabla^{\doteq}_=, \nabla^s_=, \nabla^{=}_{\doteq}, \nabla^{\doteq^-}_{=^-}, \nabla^{=^-}_{\doteq^-}$ Analogous to $\nabla_\neg$. ⊣

6.2. Hintikka sets. Hintikka sets connect syntax with semantics as they provide the basis for the model constructions in the model existence theorems. We have defined eight different notions of abstract consistency classes by first defining properties ∇_*, then specifying which should hold in $\mathfrak{Acc}_*$. Similarly, we define Hintikka sets by first defining the desired properties.

DEFINITION 6.19 (Σ-Hintikka properties). Let $\mathscr{H}$ be a set of sentences. We define the following properties which $\mathscr{H}$ may satisfy, where $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_o$, $\boldsymbol{C}, \boldsymbol{D} \in \mathrm{cwff}_\alpha$, $\boldsymbol{F} \in \mathrm{cwff}_{\alpha\to o}$, and $(\lambda X_\alpha.\boldsymbol{M}), (\lambda X.\boldsymbol{N}), \boldsymbol{G}, \boldsymbol{H} \in \mathrm{cwff}_{\alpha\to\beta}$:

$\vec{\nabla}_c$: $\boldsymbol{A} \notin \mathscr{H}$ or $\neg\boldsymbol{A} \notin \mathscr{H}$.
$\vec{\nabla}_\neg$: If $\neg\neg\boldsymbol{A} \in \mathscr{H}$, then $\boldsymbol{A} \in \mathscr{H}$.
$\vec{\nabla}_\beta$: If $\boldsymbol{A} \in \mathscr{H}$ and $\boldsymbol{A} \equiv_\beta \boldsymbol{B}$, then $\boldsymbol{B} \in \mathscr{H}$.
$\vec{\nabla}_\eta$: If $\boldsymbol{A} \in \mathscr{H}$ and $\boldsymbol{A} \equiv_{\beta\eta} \boldsymbol{B}$, then $\boldsymbol{B} \in \mathscr{H}$.
$\vec{\nabla}_\vee$: If $\boldsymbol{A} \vee \boldsymbol{B} \in \mathscr{H}$, then $\boldsymbol{A} \in \mathscr{H}$ or $\boldsymbol{B} \in \mathscr{H}$.
$\vec{\nabla}_\wedge$: If $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \mathscr{H}$, then $\neg\boldsymbol{A} \in \mathscr{H}$ and $\neg\boldsymbol{B} \in \mathscr{H}$.
$\vec{\nabla}_\forall$: If $\Pi^\alpha \boldsymbol{F} \in \mathscr{H}$, then $\boldsymbol{F}\boldsymbol{W} \in \mathscr{H}$ for each $\boldsymbol{W} \in \mathrm{cwff}_\alpha$.
$\vec{\nabla}_\exists$: If $\neg\Pi^\alpha \boldsymbol{F} \in \mathscr{H}$, then there is a parameter $w_\alpha \in \Sigma_\alpha$ such that $\neg(\boldsymbol{F}w) \in \mathscr{H}$.
$\vec{\nabla}_\mathfrak{b}$: If $\neg(\boldsymbol{A} \doteq^o \boldsymbol{B}) \in \mathscr{H}$, then $\{\boldsymbol{A}, \neg\boldsymbol{B}\} \subseteq \mathscr{H}$ or $\{\neg\boldsymbol{A}, \boldsymbol{B}\} \subseteq \mathscr{H}$.
$\vec{\nabla}_\xi$: If $\neg(\lambda X_\alpha.\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X.\boldsymbol{N}) \in \mathscr{H}$, then there is a parameter $w_\alpha \in \Sigma_\alpha$ such that $\neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N}) \in \mathscr{H}$.
$\vec{\nabla}_\mathfrak{f}$: If $\neg(\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}) \in \mathscr{H}$, then there is a parameter $w_\alpha \in \Sigma_\alpha$ such that $\neg(\boldsymbol{G}w \doteq^\beta \boldsymbol{H}w) \in \mathscr{H}$.
$\vec{\nabla}_{sat}$: Either $\boldsymbol{A} \in \mathscr{H}$ or $\neg\boldsymbol{A} \in \mathscr{H}$.
$\vec{\nabla}^r_=$: $\neg(\boldsymbol{C} =^\alpha \boldsymbol{C}) \notin \mathscr{H}$.
$\vec{\nabla}^{\doteq}_=$: If $\boldsymbol{C} =^\alpha \boldsymbol{D} \in \mathscr{H}$, then $\boldsymbol{C} \doteq^\alpha \boldsymbol{D} \in \mathscr{H}$.

DEFINITION 6.20 (Σ-Hintikka set). A set $\mathscr{H}$ of sentences is called a Σ-*Hintikka set* if it satisfies $\vec{\nabla}_c$, $\vec{\nabla}_\neg$, $\vec{\nabla}_\beta$, $\vec{\nabla}_\vee$, $\vec{\nabla}_\wedge$, $\vec{\nabla}_\forall$ and $\vec{\nabla}_\exists$. When primitive equality is present in the signature and $\mathscr{H}$ is a Hintikka set satisfying $\vec{\nabla}^r_=$ and $\vec{\nabla}^{\doteq}_=$ we call $\mathscr{H}$ a Σ-*Hintikka set with primitive equality*. We define the following collections of Hintikka sets (with primitive equality): $\mathfrak{Hint}_\beta$, $\mathfrak{Hint}_{\beta\eta}$, $\mathfrak{Hint}_{\beta\xi}$, $\mathfrak{Hint}_{\beta\mathfrak{f}}$, $\mathfrak{Hint}_{\beta\mathfrak{b}}$, $\mathfrak{Hint}_{\beta\eta\mathfrak{b}}$, $\mathfrak{Hint}_{\beta\xi\mathfrak{b}}$, and $\mathfrak{Hint}_{\beta\mathfrak{f}\mathfrak{b}}$, where we indicate by indices which additional properties from $\{\vec{\nabla}_\eta, \vec{\nabla}_\xi, \vec{\nabla}_\mathfrak{f}, \vec{\nabla}_\mathfrak{b}\}$ are required. If primitive equality is in the signature, we require $\mathscr{H} \in \mathfrak{Hint}_*$ to be a Hintikka set with primitive equality.

We will construct Hintikka sets as maximal elements of abstract consistency classes. To obtain a Hintikka set, we must explicitly show the property $\vec{\nabla}_\exists$ (and $\vec{\nabla}_\xi$

or $\vec{\nabla}_{\mathfrak{f}}$ when appropriate). This will ensure that Hintikka sets have enough parameters which act as witnesses.

LEMMA 6.21 (Hintikka lemma). *Let Γ_Σ be an abstract consistency class in $\mathfrak{Acc}_*$. Suppose a set $\mathscr{H} \in \Gamma_\Sigma$ satisfies the following properties*:

(1) *$\mathscr{H}$ is subset-maximal in Γ_Σ (i.e., for each sentence $\boldsymbol{D} \in \mathrm{cwff}_o$ such that $\mathscr{H} * \boldsymbol{D} \in \Gamma_\Sigma$, we already have $\boldsymbol{D} \in \mathscr{H}$).*
(2) *$\mathscr{H}$ satisfies $\vec{\nabla}_\exists$.*
(3) *If $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$, then $\vec{\nabla}_\xi$ holds in $\mathscr{H}$.*
(4) *If $* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$, then $\vec{\nabla}_\mathfrak{f}$ holds in $\mathscr{H}$.*

Then, $\mathscr{H} \in \mathfrak{Hint}_*$. Furthermore, if Γ_Σ is saturated, then $\mathscr{H}$ satisfies $\vec{\nabla}_{sat}$.

PROOF. $\mathscr{H}$ satisfies $\vec{\nabla}_\exists$ by assumption. Also, if $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$ ($* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$), then we have explicitly assumed $\mathscr{H}$ satisfies $\vec{\nabla}_\xi$ ($\vec{\nabla}_\mathfrak{f}$). The fact that $\mathscr{H} \in \Gamma_\Sigma$ satisfies $\vec{\nabla}_c$ follows directly from non-atomic consistency (Lemma 6.10). Similarly, if primitive equality is in the signature, then $\mathscr{H}$ satisfies $\vec{\nabla}^r_=$ since $\mathscr{H} \in \Gamma_\Sigma$ and Γ_Σ satisfies $\nabla^r_=$. Every other $\vec{\nabla}_*$ property follows directly from the corresponding ∇_* property and maximality of $\mathscr{H}$ in Γ_Σ. For example, to show $\vec{\nabla}_\neg$, suppose $\neg\neg\boldsymbol{A} \in \mathscr{H}$. By $\nabla_\neg$, we know $\mathscr{H} * \boldsymbol{A} \in \Gamma_\Sigma$. By maximality of $\mathscr{H}$, we have $\boldsymbol{A} \in \mathscr{H}$. Checking $\vec{\nabla}_\beta$, $\vec{\nabla}_\eta$ (if $* \in \{\beta\eta, \beta\eta\mathfrak{b}\}$), $\vec{\nabla}_\wedge$, $\vec{\nabla}_\forall$, and $\vec{\nabla}^{\doteq}_=$ hold for $\mathscr{H}$ follows exactly this same pattern. Checking $\vec{\nabla}_\vee$, $\vec{\nabla}_\mathfrak{b}$ (if $* \in \{\beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$) and $\vec{\nabla}_{sat}$ (if Γ_Σ is saturated) follows a similar pattern, but with a simple case analysis. For example, to check $\vec{\nabla}_{sat}$, given $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)$, ∇_{sat} implies $\mathscr{H} * \boldsymbol{A} \in \Gamma_\Sigma$ or $\mathscr{H} * \neg\boldsymbol{A} \in \Gamma_\Sigma$. So, either $\boldsymbol{A} \in \mathscr{H}$ or $\neg\boldsymbol{A} \in \mathscr{H}$. ⊣

It is worth noting that the converse of $\vec{\nabla}^{\doteq}_=$ also holds in Hintikka sets with primitive equality.

LEMMA 6.22. *Suppose primitive equality is in the signature and $\mathscr{H}$ is a Hintikka set with primitive equality. Then, we have the following property for every type α and $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_\alpha(\Sigma)$:*

$\overline{\vec{\nabla}}^{\doteq}_=$: *$\boldsymbol{A} =^\alpha \boldsymbol{B} \in \mathscr{H}$ iff $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \mathscr{H}$.*

PROOF. If $\boldsymbol{A} =^\alpha \boldsymbol{B} \in \mathscr{H}$, then $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \mathscr{H}$ by $\vec{\nabla}^{\doteq}_=$. For the converse direction assume that $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \mathscr{H}$. From this we get by $\vec{\nabla}_\forall$ with $\lambda X.\boldsymbol{A} = X$ and ∇_β that $\neg(\boldsymbol{A} = \boldsymbol{A}) \vee \boldsymbol{A} = \boldsymbol{B} \in \mathscr{H}$. Since $\neg(\boldsymbol{A} = \boldsymbol{A}) \notin \mathscr{H}$ by $\vec{\nabla}^r_=$, $\vec{\nabla}_\vee$ implies $\boldsymbol{A} =^\alpha \boldsymbol{B} \in \mathscr{H}$. ⊣

It is helpful to note the following properties of Leibniz equality in Hintikka sets.

LEMMA 6.23. *Suppose $\mathscr{H}$ is a Hintikka set. For any $\boldsymbol{F}, \boldsymbol{G} \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma)$ and $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathrm{cwff}_\alpha(\Sigma)$ (for types α and β), we have the following*:

$\vec{\nabla}^r_{\doteq}$: *$\neg(\boldsymbol{A} \doteq^\alpha \boldsymbol{A}) \notin \mathscr{H}$.*
$\vec{\nabla}^{tr}_{\doteq}$: *If $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \mathscr{H}$ and $\boldsymbol{B} \doteq^\alpha \boldsymbol{C} \in \mathscr{H}$, then $\boldsymbol{A} \doteq^\alpha \boldsymbol{C} \in \mathscr{H}$.*
$\vec{\nabla}^{\to}_{\doteq}$: *If $(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G}) \in \mathscr{H}$ and $(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \mathscr{H}$, then $(\boldsymbol{FA} \doteq^\beta \boldsymbol{GB}) \in \mathscr{H}$.*

PROOF. To show $\vec{\nabla}^r_{\doteq}$, suppose $\neg(\boldsymbol{A} \doteq^\alpha \boldsymbol{A}) \in \mathscr{H}$. By $\vec{\nabla}_\exists$ and $\vec{\nabla}_\beta$, there must be some parameter $q_{\alpha\to o}$ such that $\neg(\neg q\boldsymbol{A} \vee q\boldsymbol{A}) \in \mathscr{H}$. By $\vec{\nabla}_\wedge$, we have $\neg\neg q\boldsymbol{A} \in \mathscr{H}$ and $\neg q\boldsymbol{A} \in \mathscr{H}$, contradicting $\vec{\nabla}_c$.

To show $\vec{\nabla}^{tr}_{\doteq}$, suppose $\boldsymbol{A} \doteq^\alpha \boldsymbol{B} \in \mathscr{H}$ and $\boldsymbol{B} \doteq^\alpha \boldsymbol{C} \in \mathscr{H}$. Let $\boldsymbol{Q}_{\alpha\to o}$ be the closed formula $(\lambda X_\alpha.\boldsymbol{A} \doteq^\alpha X)$. Applying $\vec{\nabla}_\forall$ to $\boldsymbol{B} \doteq^\alpha \boldsymbol{C} \in \mathscr{H}$ and $\boldsymbol{Q}$, we know

$\neg(\boldsymbol{QB}) \vee \boldsymbol{QC} \in \mathscr{H}$. By $\vec{\nabla}_{\vee}$, we know $\neg(\boldsymbol{QB}) \in \mathscr{H}$ or $\boldsymbol{QC} \in \mathscr{H}$. If $\neg(\boldsymbol{QB}) \in \mathscr{H}$, then $\neg(\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B}) \in \mathscr{H}$ by $\vec{\nabla}_{\beta}$, contradicting $\vec{\nabla}_{c}$. So, $\boldsymbol{QC} \in \mathscr{H}$ and hence $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{C} \in \mathscr{H}$ as desired.

To show $\vec{\nabla}^{\rightarrow}_{\doteq}$, let $\boldsymbol{P}_{(\alpha\to\beta)\to o}$ be the closed formula $(\lambda H_{\alpha\to\beta}.\boldsymbol{FA} \doteq^{\beta} H\boldsymbol{A})$, Applying $\vec{\nabla}_{\forall}$ to $(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G}) \in \mathscr{H}$ and $\boldsymbol{P}$, we have $\neg(\boldsymbol{PF}) \vee \boldsymbol{PG} \in \mathscr{H}$. By $\vec{\nabla}_{\vee}$, we know $\neg(\boldsymbol{PF}) \in \mathscr{H}$ or $\boldsymbol{PG} \in \mathscr{H}$. If $\neg(\boldsymbol{PF}) \in \mathscr{H}$, then $\neg(\boldsymbol{FA} \doteq^{\beta} \boldsymbol{FA}) \in \mathscr{H}$ by $\vec{\nabla}_{\beta}$, which contradicts $\vec{\nabla}^{r}_{\doteq}$. So, we must have $\boldsymbol{PG} \in \mathscr{H}$ and hence $(\boldsymbol{FA} \doteq^{\beta} \boldsymbol{GA}) \in \mathscr{H}$. Let $\boldsymbol{Q}_{\alpha\to o}$ be the closed formula $(\lambda X_{\alpha}.\boldsymbol{FA} \doteq^{\beta} \boldsymbol{GX})$. Applying $\vec{\nabla}_{\forall}$ and $\vec{\nabla}_{\vee}$ to $(\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B}) \in \mathscr{H}$, we know $\neg(\boldsymbol{QA}) \in \mathscr{H}$ or $\boldsymbol{QB} \in \mathscr{H}$. If $\neg(\boldsymbol{QA}) \in \mathscr{H}$, then $\neg(\boldsymbol{FA} \doteq^{\beta} \boldsymbol{GA}) \in \mathscr{H}$ by $\vec{\nabla}_{\beta}$, contradicting $\vec{\nabla}_{c}$. So, $\boldsymbol{QB} \in \mathscr{H}$ and hence $(\boldsymbol{FA} \doteq^{\beta} \boldsymbol{GB}) \in \mathscr{H}$ as desired. ⊣

Whenever a Hintikka set satisfies $\vec{\nabla}_{sat}$, we can prove far more closure properties. For example, we can prove converses of $\vec{\nabla}_{\neg}$, $\vec{\nabla}_{\beta}$, $\vec{\nabla}_{\vee}$, $\vec{\nabla}_{\wedge}$, $\vec{\nabla}_{\forall}$, $\vec{\nabla}_{\exists}$ and $\vec{\nabla}^{\doteq}_{=}$ (when primitive equality is in the signature). Also, if any of $\vec{\nabla}_{\eta}$, $\vec{\nabla}_{\mathfrak{b}}$, $\vec{\nabla}_{\xi}$ or $\vec{\nabla}_{\mathfrak{f}}$ hold, we can prove the corresponding converse. (We could call these properties $\overleftarrow{\nabla}_{*}$.) The proofs of the stronger properties $\overline{\nabla}_{\neg}$ and $\overline{\nabla}_{\vee}$ in Lemma 6.25 indicate how one would prove any of these converse properties.

DEFINITION 6.24 (Saturated set). We say a set of sentences $\mathscr{H}$ is *saturated* if it satisfies $\vec{\nabla}_{sat}$.

By Lemma 6.21, any Hintikka set constructed as a maximal member of a saturated abstract consistency class will be saturated. However, it is also possible for a maximal member of an abstract consistency class Γ_{Σ} to be saturated without Γ_{Σ} being saturated.

LEMMA 6.25 (Saturated sets lemma). *Suppose $\mathscr{H}$ is a saturated Hintikka set. Then we have the following properties for every $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_{o}(\Sigma)$, $\boldsymbol{F} \in \mathrm{cwff}_{\alpha\to o}(\Sigma)$, and $\boldsymbol{C} \in \mathrm{cwff}_{\alpha}(\Sigma)$ (for any type α):*

$\overline{\nabla}_{\neg}$: *$\neg\boldsymbol{A} \in \mathscr{H}$ iff $\boldsymbol{A} \notin \mathscr{H}$.*
$\overline{\nabla}_{\vee}$: *$(\boldsymbol{A} \vee \boldsymbol{B}) \in \mathscr{H}$ iff $\boldsymbol{A} \in \mathscr{H}$ or $\boldsymbol{B} \in \mathscr{H}$.*
$\overline{\nabla}_{\forall}$: *$(\Pi^{\alpha}\boldsymbol{F}) \in \mathscr{H}$ if and only if $\boldsymbol{FD} \in \mathscr{H}$ for every $\boldsymbol{D} \in \mathrm{cwff}_{\alpha}(\Sigma)$.*
$\overline{\nabla}^{\beta}_{\forall}$: *$(\Pi^{\alpha}\boldsymbol{F}) \in \mathscr{H}$ iff $(\boldsymbol{FD})\downarrow_{\beta} \in \mathscr{H}$ for every $\boldsymbol{D} \in \mathrm{cwff}_{\alpha}(\Sigma)\downarrow_{\beta}$.*
$\overline{\nabla}_{r}$: *$(\boldsymbol{C} \doteq^{\alpha} \boldsymbol{C}) \in \mathscr{H}$.*

PROOF. If $\neg\boldsymbol{A} \in \mathscr{H}$, then $\boldsymbol{A} \notin \mathscr{H}$ by $\vec{\nabla}_{c}$. If $\boldsymbol{A} \notin \mathscr{H}$, then $\neg\boldsymbol{A} \in \mathscr{H}$ since $\mathscr{H}$ is saturated. So, $\overline{\nabla}_{\neg}$ holds.

If $(\boldsymbol{A} \vee \boldsymbol{B}) \in \mathscr{H}$, then $\boldsymbol{A} \in \mathscr{H}$ or $\boldsymbol{B} \in \mathscr{H}$ by $\vec{\nabla}_{\vee}$. We prove the converse by contraposition. Suppose $(\boldsymbol{A} \vee \boldsymbol{B}) \notin \mathscr{H}$. By saturation we have $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \mathscr{H}$, and by $\vec{\nabla}_{\wedge}$ we get $\neg\boldsymbol{A} \in \mathscr{H}$ and $\neg\boldsymbol{B} \in \mathscr{H}$. So, by $\vec{\nabla}_{c}$, $\boldsymbol{A} \notin \mathscr{H}$ and $\boldsymbol{B} \notin \mathscr{H}$. Thus, $\overline{\nabla}_{\vee}$ holds.

One direction of $\overline{\nabla}_{\forall}$ is $\vec{\nabla}_{\forall}$. For one direction of $\overline{\nabla}^{\beta}_{\forall}$, note that if $(\Pi^{\alpha}\boldsymbol{F}) \in \mathscr{H}$, then for any $\boldsymbol{D} \in \mathrm{cwff}_{\alpha}(\Sigma)\downarrow_{\beta}$ we have $(\boldsymbol{FD})\downarrow_{\beta} \in \mathscr{H}$ by $\vec{\nabla}_{\forall}$ and $\vec{\nabla}_{\beta}$.

Suppose $(\Pi^{\alpha}\boldsymbol{F}) \notin \mathscr{H}$. By saturation, $\neg(\Pi^{\alpha}\boldsymbol{F}) \in \mathscr{H}$. By $\vec{\nabla}_{\exists}$, there is a parameter $w_{\alpha} \in \Sigma_{\alpha}$ such that $\neg(\boldsymbol{F}w) \in \mathscr{H}$. By $\vec{\nabla}_{c}$, we know $(\boldsymbol{F}w) \notin \mathscr{H}$. This shows the other direction of $\overline{\nabla}_{\forall}$. Furthermore, by $\vec{\nabla}_{\beta}$ we know $\neg(\boldsymbol{F}w)\downarrow_{\beta} \in \mathscr{H}$ and so $(\boldsymbol{F}w)\downarrow_{\beta} \notin \mathscr{H}$. Since w is β-normal, we also have the other direction of $\overline{\nabla}^{\beta}_{\forall}$.

Finally, $\overline{\nabla_r}$ follows directly from saturation and $\vec{\nabla}^r_{\doteq}$. ⊣

LEMMA 6.26 (Saturated sets lemma for $\mathfrak{b}$). *Suppose* $\mathscr{H} \in \mathfrak{Hint}_*$ *where* $* \in \{\beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. *If* $\mathscr{H}$ *is saturated, then the following property holds for all* $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_o(\Sigma)$.

$\overline{\nabla_{\mathfrak{b}}}$: $\boldsymbol{A} \doteq^o \boldsymbol{B} \in \mathscr{H}$ *or* $\boldsymbol{A} \doteq^o \neg\boldsymbol{B} \in \mathscr{H}$.

PROOF. Suppose $(\boldsymbol{A} \doteq^o \boldsymbol{B}) \notin \mathscr{H}$ and $(\boldsymbol{A} \doteq^o \neg\boldsymbol{B}) \notin \mathscr{H}$. By saturation, $\neg(\boldsymbol{A} \doteq^o \boldsymbol{B}) \in \mathscr{H}$ and $\neg(\boldsymbol{A} \doteq^o \neg\boldsymbol{B}) \in \mathscr{H}$. By $\vec{\nabla}_{\mathfrak{b}}$, we must have $\{\boldsymbol{A}, \neg\boldsymbol{B}\} \subseteq \mathscr{H}$ or $\{\neg\boldsymbol{A}, \boldsymbol{B}\} \subseteq \mathscr{H}$. We must also have $\{\boldsymbol{A}, \neg\neg\boldsymbol{B}\} \subseteq \mathscr{H}$ or $\{\neg\boldsymbol{A}, \neg\boldsymbol{B}\} \subseteq \mathscr{H}$. Each of the four cases leads to an immediate contradiction to $\vec{\nabla}_c$. ⊣

LEMMA 6.27 (Saturated sets lemma for η). *Suppose* $\mathscr{H} \in \mathfrak{Hint}_*$ *where* $* \in \{\beta\eta, \beta\eta\mathfrak{b}\}$. *If* $\mathscr{H}$ *is saturated, then the following property holds for every type* α *and* $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$:

$\overline{\nabla_\eta}$: $(\boldsymbol{A} \doteq^\alpha \boldsymbol{A}\downarrow_{\beta\eta}) \in \mathscr{H}$.

PROOF. If $(\boldsymbol{A} \doteq \boldsymbol{A}\downarrow_{\beta\eta}) \notin \mathscr{H}$, then by saturation $\neg(\boldsymbol{A} \doteq \boldsymbol{A}\downarrow_{\beta\eta}) \in \mathscr{H}$. So, by $\vec{\nabla}_\eta$ we have $\neg(\boldsymbol{A}\downarrow_{\beta\eta} \doteq^\alpha \boldsymbol{A}\downarrow_{\beta\eta}) \in \mathscr{H}$. But this contradicts $\vec{\nabla}^r_{\doteq}$. ⊣

LEMMA 6.28 (Saturated sets lemma for ξ). *Suppose* $\mathscr{H} \in \mathfrak{Hint}_*$ *where* $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$. *If* $\mathscr{H}$ *is saturated, then the following properties hold for all* $\alpha, \beta \in \mathscr{T}$ *and* $(\lambda X_\alpha . \boldsymbol{M}), (\lambda X . \boldsymbol{N}) \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma)$:

$\overline{\nabla_\xi}$: $(\lambda X . \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X . \boldsymbol{N}) \in \mathscr{H}$ *iff* $([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N}) \in \mathscr{H}$ *for every* $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$.

$\overline{\nabla_\xi}^\beta$: $(\lambda X . \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X . \boldsymbol{N}) \in \mathscr{H}$ *iff* $([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N})\big\downarrow_\beta \in \mathscr{H}$ *for every* $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$.

PROOF. Suppose $(\lambda X . \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X . \boldsymbol{N}) \in \mathscr{H}$ and $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$. We can apply $\vec{\nabla}_\forall$ and $\vec{\nabla}_\beta$ using the closed formula $(\lambda K_{\alpha\to\beta} . [\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta K\boldsymbol{A})$ to obtain

$$(\neg([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{M}) \vee [\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N}) \in \mathscr{H}.$$

Since $\neg([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{M}) \notin \mathscr{H}$ (by $\vec{\nabla}^r_{\doteq}$), we know $([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N}) \in \mathscr{H}$. This shows one direction of $\overline{\nabla_\xi}$. By $\vec{\nabla}_\beta$ we have $([\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N})\big\downarrow_\beta \in \mathscr{H}$. Since this holds in particular for any $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$, this shows one direction of $\overline{\nabla_\xi}^\beta$.

Suppose $(\lambda X . \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X . \boldsymbol{N}) \notin \mathscr{H}$. We show that there is a (β-normal) $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$ with $[\boldsymbol{A}/X]\boldsymbol{M} \doteq^\beta [\boldsymbol{A}/X]\boldsymbol{N} \notin \mathscr{H}$. By saturation, $\neg(\lambda X . \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X . \boldsymbol{N}) \in \mathscr{H}$. By $\vec{\nabla}_\xi$, there is a parameter $w_\alpha \in \Sigma_\alpha$ such that $\neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N}) \in \mathscr{H}$. By $\vec{\nabla}_c$, $[w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N} \notin \mathscr{H}$. Choosing $\boldsymbol{A} := w$ we have the other direction of $\overline{\nabla_\xi}$. Since w is β-normal and $([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})\big\downarrow_\beta \notin \mathscr{H}$ (using $\vec{\nabla}_\beta$), we have the other direction of $\overline{\nabla_\xi}^\beta$. ⊣

LEMMA 6.29 (Saturated sets lemma for $\mathfrak{f}$). *Suppose* $\mathscr{H} \in \mathfrak{Hint}_*$ *where* $* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$. *If* $\mathscr{H}$ *is saturated, then the following property holds for any types* α *and* β *and* $\boldsymbol{G}, \boldsymbol{H} \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma)$.

$\overline{\nabla_{\mathfrak{f}}}$: $\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H} \in \mathscr{H}$ *iff* $\boldsymbol{GA} \doteq^\beta \boldsymbol{HA} \in \mathscr{H}$ *for every* $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)$.

$\overline{\nabla}_{\mathfrak{f}}^{\beta}$: $\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H} \in \mathscr{H}$ *iff* $(\boldsymbol{GA} \doteq^{\beta} \boldsymbol{HA})\downarrow_{\beta} \in \mathscr{H}$ *for every* $\boldsymbol{A} \in \mathrm{cwff}_{\alpha}(\Sigma)\downarrow_{\beta}$.

PROOF. Suppose $(\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}) \in \mathscr{H}$ and $\boldsymbol{A} \in \mathrm{cwff}_{\alpha}(\Sigma)$. Since $(\boldsymbol{A} \doteq^{\alpha} \boldsymbol{A}) \in \mathscr{H}$ by $\overline{\nabla}_r$ we have $(\boldsymbol{GA} \doteq^{\beta} \boldsymbol{HA}) \in \mathscr{H}$ by $\vec{\nabla}_{\doteq}^{\to}$ (cf. Lemma 6.23). This shows one direction of $\overline{\nabla}_{\mathfrak{f}}$. By $\vec{\nabla}_{\beta}$ we have $(\boldsymbol{GA} \doteq^{\beta} \boldsymbol{HA})\downarrow_{\beta} \in \mathscr{H}$. Since this holds in particular for any $\boldsymbol{A} \in \mathrm{cwff}_{\alpha}(\Sigma)\downarrow_{\beta}$, this shows one direction of $\overline{\nabla}_{\mathfrak{f}}^{\beta}$.

Suppose $(\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}) \notin \mathscr{H}$. By saturation, $\neg(\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}) \in \mathscr{H}$. By $\vec{\nabla}_{\mathfrak{f}}$, there is a parameter $w_{\alpha} \in \Sigma_{\alpha}$ such that $\neg(\boldsymbol{G}w \doteq^{\beta} \boldsymbol{H}w) \in \mathscr{H}$. By $\vec{\nabla}_{c}$, $(\boldsymbol{G}w \doteq^{\beta} \boldsymbol{H}w) \notin \mathscr{H}$. Choosing $\boldsymbol{A} := w$ we have the other direction of $\overline{\nabla}_{\mathfrak{f}}$. Since w is β-normal and $(\boldsymbol{G}w \doteq^{\beta} \boldsymbol{H}w)\downarrow_{\beta} \notin \mathscr{H}$ (using $\vec{\nabla}_{\beta}$), we have the other direction of $\overline{\nabla}_{\mathfrak{f}}^{\beta}$. ⊣

In Lemma 3.24, we compared properties η, ξ and $\mathfrak{f}$ of models by showing $\mathfrak{f}$ is equivalent to η plus ξ. Similarly, Theorem 6.31 compares $\vec{\nabla}_{\eta}$, $\vec{\nabla}_{\xi}$, and $\vec{\nabla}_{\mathfrak{f}}$ as properties of Hintikka sets. Showing $\vec{\nabla}_{\mathfrak{f}}$ implies $\vec{\nabla}_{\eta}$ requires saturation and must be shown in several steps reflected by Lemma 6.30.

LEMMA 6.30. *Let $\mathscr{H}$ be a saturated Hintikka set satisfying $\vec{\nabla}_{\mathfrak{f}}$.*

(1) *For all $\boldsymbol{F} \in \mathrm{cwff}_{\alpha\to\beta}$ we have $(\lambda X_{\alpha}.\boldsymbol{F}X) \doteq^{\alpha\to\beta} \boldsymbol{F} \in \mathscr{H}$.*
(2) *For all $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_{\alpha}(\Sigma)$, if $\boldsymbol{A}$ η-reduces to $\boldsymbol{B}$ in one step, then $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B} \in \mathscr{H}$.*
(3) *For all $\boldsymbol{A} \in \mathrm{cwff}_{\alpha}(\Sigma)$, $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$.*
(4) *For all $\boldsymbol{A} \in \mathrm{cwff}_{o}(\Sigma)$, if $\boldsymbol{A} \in \mathscr{H}$, then $\boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$.*

PROOF. To show part (1), suppose $(\lambda X_{\alpha}.\boldsymbol{F}X) \doteq^{\alpha\to\beta} \boldsymbol{F} \notin \mathscr{H}$. By saturation, $\neg((\lambda X_{\alpha}.\boldsymbol{F}X) \doteq^{\alpha\to\beta} \boldsymbol{F}) \in \mathscr{H}$. By $\vec{\nabla}_{\mathfrak{f}}$, there is a parameter w_{α} such that

$$\neg(((\lambda X_{\alpha}.\boldsymbol{F}X)w) \doteq^{\beta} (\boldsymbol{F}w)) \in \mathscr{H}.$$

By $\vec{\nabla}_{\beta}$, $\neg((\boldsymbol{F}w) \doteq^{\beta} (\boldsymbol{F}w)) \in \mathscr{H}$, which contradicts $\vec{\nabla}_{\doteq}^{r}$ (cf. Lemma 6.23).

We prove part (2) by induction on the position of the η-redex in $\boldsymbol{A}$. If $\boldsymbol{A}$ is the η-redex reduced to obtain $\boldsymbol{B}$, then this follows from part (1). Suppose $\boldsymbol{A} \equiv (\boldsymbol{F}_{\gamma\to\alpha}\boldsymbol{C}_{\gamma})$ and $\boldsymbol{B} \equiv (\boldsymbol{G}_{\gamma\to\alpha}\boldsymbol{C})$ where $\boldsymbol{F}$ η-reduces to $\boldsymbol{G}$ in one step. By induction, we know $\boldsymbol{F} \doteq^{\gamma\to\alpha} \boldsymbol{G} \in \mathscr{H}$. By $\overline{\nabla}_r$, $\boldsymbol{C} \doteq^{\gamma} \boldsymbol{C} \in \mathscr{H}$. By $\vec{\nabla}_{\doteq}^{\to}$, we have $(\boldsymbol{FC}) \doteq^{\alpha} (\boldsymbol{GC}) \in \mathscr{H}$ as desired. The case in which $\boldsymbol{A} \equiv (\boldsymbol{F}_{\gamma\to\alpha}\boldsymbol{C}_{\gamma})$ and $\boldsymbol{B} \equiv (\boldsymbol{FD}_{\gamma})$ where $\boldsymbol{C}$ η-reduces to $\boldsymbol{D}$ in one step is analogous.

Suppose $\boldsymbol{A} \equiv (\lambda Y_{\beta}.\boldsymbol{C}_{\gamma})$ and $\boldsymbol{B} \equiv (\lambda Y_{\beta}.\boldsymbol{D}_{\gamma})$ where $\boldsymbol{C}$ η-reduces to $\boldsymbol{D}$ in one step. Let p be the position of the redex in $\boldsymbol{C}$. Assume $\boldsymbol{A} \doteq^{\beta\to\gamma} \boldsymbol{B} \notin \mathscr{H}$. By saturation, $\neg(\boldsymbol{A} \doteq^{\beta\to\gamma} \boldsymbol{B}) \in \mathscr{H}$. By $\vec{\nabla}_{\mathfrak{f}}$, there is some parameter w_{β} such that $\neg(\boldsymbol{A}w \doteq^{\gamma} \boldsymbol{B}w) \in \mathscr{H}$. By $\vec{\nabla}_{\beta}$, we know $\neg([w/Y]\boldsymbol{C} \doteq^{\gamma} [w/Y]\boldsymbol{D}) \in \mathscr{H}$. Note that $[w/Y]\boldsymbol{C}$ η-reduces to $[w/Y]\boldsymbol{D}$ in one step by reducing the redex at position p in $[w/Y]\boldsymbol{C}$. So, by the induction hypothesis, $[w/Y]\boldsymbol{C} \doteq^{\gamma} [w/Y]\boldsymbol{D} \in \mathscr{H}$, contradicting $\vec{\nabla}_{c}$.

Part (3) follows by induction on the number of $\beta\eta$-reductions from $\boldsymbol{A}$ to $\boldsymbol{A}\downarrow_{\beta\eta}$. If $\boldsymbol{A}$ is $\beta\eta$-normal, we have $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{A} \in \mathscr{H}$ by $\overline{\nabla}_r$. If $\boldsymbol{A}$ reduces to $\boldsymbol{A}\downarrow_{\beta\eta}$ in $n+1$ steps, then there is some $\boldsymbol{B}_{\alpha}$ such that $\boldsymbol{A}$ reduces to $\boldsymbol{B}$ in one step and $\boldsymbol{B}$ reduces to $\boldsymbol{A}\downarrow_{\beta\eta}$ in n steps. By induction, we have $\boldsymbol{B} \doteq^{\alpha} \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$. If $\boldsymbol{A}$ β-reduces to $\boldsymbol{B}$ in one step, then $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B} \in \mathscr{H}$ by $\overline{\nabla}_r$ and $\vec{\nabla}_{\beta}$. If $\boldsymbol{A}$ η-reduces to $\boldsymbol{B}$ in one step, then $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B} \in \mathscr{H}$

by part (2). Using $\vec{\nabla}^{tr}_{=}$, $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{B} \in \mathscr{H}$ and $\boldsymbol{B} \doteq^{\alpha} \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$ imply $\boldsymbol{A} \doteq^{\alpha} \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$ as desired.

Finally, to show part (4), suppose $\boldsymbol{A} \in \mathscr{H}$. By part (3), $\boldsymbol{A} \doteq^{o} \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$. By $\vec{\nabla}_{\forall}$, $\neg(\lambda X_{o\text{-}} X)\boldsymbol{A} \vee (\lambda X_{o\text{-}} X)\ \boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$. By $\vec{\nabla}_{\beta}$ and $\vec{\nabla}_{\vee}$, we have $\neg\boldsymbol{A} \in \mathscr{H}$ (contradicting $\vec{\nabla}_{c}$) or $\boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$. Hence, $\boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$. ⊣

THEOREM 6.31. *Let $\mathscr{H}$ be a Hintikka set.*

(1) *If $\mathscr{H}$ satisfies $\vec{\nabla}_{\eta}$ and $\vec{\nabla}_{\xi}$, then $\mathscr{H}$ satisfies $\vec{\nabla}_{\mathfrak{f}}$.*
(2) *If $\mathscr{H}$ satisfies $\vec{\nabla}_{\mathfrak{f}}$, then $\mathscr{H}$ satisfies $\vec{\nabla}_{\xi}$.*
(3) *If $\mathscr{H}$ is saturated and satisfies $\vec{\nabla}_{\mathfrak{f}}$, then $\mathscr{H}$ satisfies $\vec{\nabla}_{\eta}$.*

PROOF. Suppose $\mathscr{H}$ satisfies $\vec{\nabla}_{\eta}$ and $\vec{\nabla}_{\xi}$. Assume $\neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G}) \in \mathscr{H}$. By $\vec{\nabla}_{\eta}$, $\neg((\lambda X_{\alpha\text{-}}\boldsymbol{F}X) \doteq^{\alpha\to\beta} (\lambda X_{\text{-}}\boldsymbol{G}X)) \in \mathscr{H}$. By $\vec{\nabla}_{\xi}$, there is a parameter w_{α} such that $\neg((\boldsymbol{F}w) \doteq^{\beta} (\boldsymbol{G}w)) \in \mathscr{H}$. Thus, $\vec{\nabla}_{\mathfrak{f}}$ holds.

Suppose $\mathscr{H}$ satisfies $\vec{\nabla}_{\mathfrak{f}}$ and $\neg(\lambda X_{\alpha\text{-}}\boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X_{\text{-}}\boldsymbol{N}) \in \mathscr{H}$. By $\vec{\nabla}_{\mathfrak{f}}$, there is a parameter w_{α} such that $\neg((\lambda X_{\alpha\text{-}}\boldsymbol{M})w \doteq^{\beta} (\lambda X_{\text{-}}\boldsymbol{N})w) \in \mathscr{H}$. By $\vec{\nabla}_{\beta}$, we have $\neg([w/X]\boldsymbol{M} \doteq^{\beta} [w/X]\boldsymbol{N}) \in \mathscr{H}$. Thus, $\vec{\nabla}_{\xi}$ holds.

Suppose $\mathscr{H}$ is saturated and satisfies $\vec{\nabla}_{\mathfrak{f}}$. Assume $\boldsymbol{A} \in \mathscr{H}$, $\boldsymbol{B} \in \mathrm{cwff}_{o}(\Sigma)$, $\boldsymbol{A} \equiv_{\beta\eta} \boldsymbol{B}$ and $\boldsymbol{B} \notin \mathscr{H}$. By saturation, we know $\neg\boldsymbol{B} \in \mathscr{H}$. By Lemma 6.30(4), we know $\boldsymbol{A}\downarrow_{\beta\eta} \in \mathscr{H}$ and $\neg\ \boldsymbol{B}\downarrow_{\beta\eta} \in \mathscr{H}$. Since $\boldsymbol{A}\downarrow_{\beta\eta} \equiv \boldsymbol{B}\downarrow_{\beta\eta}$, this contradicts $\vec{\nabla}_{c}$. ⊣

6.3. Model existence theorems. 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 [26, 27].

LEMMA 6.32 (Abstract extension lemma). *Let Σ be a signature, Γ_{Σ} be a compact abstract consistency class in $\mathfrak{Acc}_{*}$, where $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$, and let $\Phi \in \Gamma_{\Sigma}$ be sufficiently Σ-pure. Then there exists a Σ-Hintikka set $\mathscr{H} \in \mathfrak{Hint}_{*}$, such that $\Phi \subseteq \mathscr{H}$. Furthermore, if Γ_{Σ} is saturated, then $\mathscr{H}$ is saturated.*

PROOF. In the following argument, note that α, β, and γ are types as usual, while δ, ε, σ and τ are ordinals.

By Remark 3.16, there is an infinite cardinal $\aleph_{s}$ which is the cardinality of Σ_{α} for each type α. This easily implies $\mathrm{cwff}_{\alpha}(\Sigma)$ is of cardinality $\aleph_{s}$ for each type α. Let ε be the first ordinal of this cardinality. (In the countable case, ε is ω.) Since the cardinality of $\mathrm{cwff}_{o}(\Sigma)$ is $\aleph_{s}$, we can use the well-ordering principle to enumerate $\mathrm{cwff}_{o}(\Sigma)$ as $(\boldsymbol{A}^{\delta})_{\delta<\varepsilon}$.

Let α be a type. For each $\delta < \varepsilon$, let U^{δ}_{α} be the set of constants of type α which occur in a sentence in the set $\{\, \boldsymbol{A}^{\sigma} \mid \sigma \leq \delta \,\}$. Since $\delta < \varepsilon$, the set $\{\, \boldsymbol{A}^{\sigma} \mid \sigma \leq \delta \,\}$ has cardinality less than $\aleph_{s}$. Hence, U^{δ}_{α} has cardinality less than $\aleph_{s}$. By sufficient purity, we know there is a set of parameters $P_{\alpha} \subseteq \Sigma_{\alpha}$ of cardinality $\aleph_{s}$ such that the parameters in P_{α} do not occur in the sentences of Φ. So, $P_{\alpha} \setminus U^{\delta}_{\alpha}$ must have cardinality $\aleph_{s}$ for any $\delta < \varepsilon$. Using the axiom of choice, we can find a sequence $(w^{\delta}_{\alpha})_{\delta<\varepsilon}$ where for each $\delta < \varepsilon$, $w^{\delta}_{\alpha} \in P_{\alpha} \setminus (U^{\delta}_{\alpha} \cup \{\, w^{\sigma}_{\alpha} \mid \sigma < \delta \,\})$. That is, for each type α, we know w^{δ}_{α} is a parameter of type α which does not occur in any sentence in $\Phi \cup \{\, \boldsymbol{A}^{\sigma} \mid \sigma \leq \delta \,\}$. As a consequence, if w^{δ}_{α} occurs in $\boldsymbol{A}^{\sigma}$, then $\delta < \sigma$. Also, we have ensured that if $w^{\delta}_{\alpha} \equiv w^{\sigma}_{\alpha}$, then $\delta \equiv \sigma$ for any $\delta, \sigma < \varepsilon$.

The parameters w_α^δ are intended to serve as witnesses. To ease the argument, we define two sequences of witnessing sentences related to the sequence $(\boldsymbol{A}^\delta)_{\delta<\varepsilon}$. For each $\delta < \varepsilon$, let $\boldsymbol{E}^\delta := \neg(\boldsymbol{B} w_\alpha^\delta)$ if $\boldsymbol{A}^\delta$ is of the form $\neg(\Pi^\alpha \boldsymbol{B})$, and let $\boldsymbol{E}^\delta := \boldsymbol{A}^\delta$ otherwise. If $* \in \{\beta\mathfrak{f}, \beta\mathfrak{fb}\}$ and $\boldsymbol{A}^\delta$ is of the form $\neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G})$, let $\boldsymbol{X}^\delta := \neg(\boldsymbol{F} w_\alpha^\delta \doteq^\beta \boldsymbol{G} w_\alpha^\delta)$. If $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$ and $\boldsymbol{A}^\delta$ is of the form $\neg((\lambda X_\alpha . \boldsymbol{M}) \doteq^{\alpha\to\beta} (\lambda X_\alpha . \boldsymbol{N}))$, let $\boldsymbol{X}^\delta := \neg([w_\alpha^\delta / X]\boldsymbol{M} \doteq^\beta [w_\alpha^\delta / X]\boldsymbol{N})$. Otherwise, let $\boldsymbol{X}^\delta := \boldsymbol{A}^\delta$. (Notice that any sentence $\neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G})$ is also of the form $\neg(\Pi^\gamma \boldsymbol{B})$, where γ is $(\alpha \to \beta) \to o$. So, whenever $\boldsymbol{X}^\delta \not\equiv \boldsymbol{A}^\delta$, we must also have $\boldsymbol{E}^\delta \not\equiv \boldsymbol{A}^\delta$.)

We construct $\mathscr{H}$ by inductively constructing a transfinite sequence $(\mathscr{H}^\delta)_{\delta<\varepsilon}$ such that $\mathscr{H}^\delta \in \Gamma_\Sigma$ for each $\delta < \varepsilon$. Then the Σ-Hintikka set is $\mathscr{H} := \bigcup_{\delta<\varepsilon} \mathscr{H}^\delta$. We define $\mathscr{H}^0 := \Phi$. For limit ordinals δ, we define $\mathscr{H}^\delta := \bigcup_{\sigma<\delta} \mathscr{H}^\sigma$.

In the successor case, if $\mathscr{H}^\delta * \boldsymbol{A}^\delta \in \Gamma_\Sigma$, then we let $\mathscr{H}^{\delta+1} := \mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta * \boldsymbol{X}^\delta$. If $\mathscr{H}^\delta * \boldsymbol{A}^\delta \notin \Gamma_\Sigma$, we let $\mathscr{H}^{\delta+1} := \mathscr{H}^\delta$.

We show by induction that for every $\delta < \varepsilon$, type α and parameter w_α^τ which occurs in some sentence in $\mathscr{H}^\delta$, we have $\tau < \delta$. The base case holds since no w_α^τ occurs in any sentence in $\mathscr{H}^0 \equiv \Phi$. For any limit ordinal δ, if w_α^τ occurs in some sentence in $\mathscr{H}^\delta$, then by definition of $\mathscr{H}^\delta$, w_α^τ already occurs in some sentence in $\mathscr{H}^\sigma$ for some $\sigma < \delta$. So, $\tau < \sigma < \delta$.

For any successor ordinal $\delta + 1$, suppose w_α^τ occurs in some sentence in $\mathscr{H}^{\delta+1}$. If it already occurred in a sentence in $\mathscr{H}^\delta$, then we have $\tau < \delta < \delta + 1$ by the inductive assumption. So, we need only consider the case where w_α^τ occurs in a sentence in $\mathscr{H}^{\delta+1} \setminus \mathscr{H}^\delta$. Note that $(\mathscr{H}^{\delta+1} \setminus \mathscr{H}^\delta) \subseteq \{\boldsymbol{A}^\delta, \boldsymbol{E}^\delta, \boldsymbol{X}^\delta\}$. In any case, note that if τ is δ, then we are done, since $\delta < \delta + 1$. If w_α^τ is any parameter with $\tau \not\equiv \delta$ and occurs in $\boldsymbol{E}^\delta$ or $\boldsymbol{X}^\delta$, then it must also occur in $\boldsymbol{A}^\delta$ (by noting that $w_\alpha^\tau \not\equiv w_\alpha^\delta$ and inspecting the possible definitions of $\boldsymbol{E}^\delta$ and $\boldsymbol{X}^\delta$), in which case $\tau < \delta < \delta + 1$.

In particular, we now know w_α^δ does not occur in any sentence of $\mathscr{H}^\delta$ for any $\delta < \varepsilon$ and type α.

Next we show by induction that $\mathscr{H}^\delta \in \Gamma_\Sigma$ for all $\delta < \varepsilon$. The base case holds by the assumption that $\mathscr{H}^0 \equiv \Phi \in \Gamma_\Sigma$. For any limit ordinal δ, assume $\mathscr{H}^\sigma \in \Gamma_\Sigma$ for every $\sigma < \delta$. We have $\mathscr{H}^\delta \equiv \bigcup_{\sigma<\delta} \mathscr{H}^\sigma \in \Gamma_\Sigma$ by compactness, since any finite subset of $\mathscr{H}^\delta$ is a subset of $\mathscr{H}^\sigma$ for some $\sigma < \delta$.

For any successor ordinal $\delta + 1$, we assume $\mathscr{H}^\delta \in \Gamma_\Sigma$. We have to show that $\mathscr{H}^{\delta+1} \in \Gamma_\Sigma$. This is trivial in case $\mathscr{H}^\delta * \boldsymbol{A}^\delta \notin \Gamma_\Sigma$ (for all abstract consistency classes) since $\mathscr{H}^{\delta+1} \equiv \mathscr{H}^\delta$. Suppose $\mathscr{H}^\delta * \boldsymbol{A}^\delta \in \Gamma_\Sigma$. We consider three sub-cases:

(i) If $\boldsymbol{E}^\delta \equiv \boldsymbol{A}^\delta$ and $\boldsymbol{X}^\delta \equiv \boldsymbol{A}^\delta$, then $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta * \boldsymbol{X}^\delta \in \Gamma_\Sigma$ since $\mathscr{H}^\delta * \boldsymbol{A}^\delta \in \Gamma_\Sigma$.

(ii) If $\boldsymbol{E}^\delta \not\equiv \boldsymbol{A}^\delta$ and $\boldsymbol{X}^\delta \equiv \boldsymbol{A}^\delta$, then $\boldsymbol{A}^\delta$ is of the form $\neg\Pi^\alpha \boldsymbol{B}$ and $\boldsymbol{E}^\delta \equiv \neg \boldsymbol{B} w_\alpha^\delta$. We conclude that $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta \in \Gamma_\Sigma$ by $\nabla_\exists$ since w_α^δ does not occur in $\boldsymbol{A}^\delta$ or any sentence of $\mathscr{H}^\delta$. Since $\boldsymbol{X}^\delta \equiv \boldsymbol{A}^\delta$, this is the same as concluding $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta * \boldsymbol{X}^\delta \in \Gamma_\Sigma$.

(iii) If $\boldsymbol{X}^\delta \not\equiv \boldsymbol{A}^\delta$, then $* \in \{\beta\xi, \beta\mathfrak{f}, \beta\xi\mathfrak{b}, \beta\mathfrak{fb}\}$ (by the definition of $\boldsymbol{X}^\delta$). $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta \in \Gamma_\Sigma$ by $\nabla_\exists$ since $w^\delta_{(\alpha\to\beta)\to o}$ does not occur in $\boldsymbol{A}^\delta$ or any sentence in $\mathscr{H}^\delta$. Now, w_α^δ (which is different from $w^\delta_{(\alpha\to\beta)\to o}$ since it has a different type) does not occur in any sentence in $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta$. We have $\mathscr{H}^\delta * \boldsymbol{A}^\delta * \boldsymbol{E}^\delta * \boldsymbol{X}^\delta \in \mathscr{H}$ by ∇_ξ (if $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$) or by $\nabla_\mathfrak{f}$ (if $* \in \{\beta\mathfrak{f}, \beta\mathfrak{fb}\}$).

Since Γ_Σ is compact, we also have $\mathscr{H} \in \Gamma_\Sigma$.

Now we know that our inductively defined set $\mathscr{H}$ is indeed in Γ_Σ and that $\Phi \subseteq \mathscr{H}$. In order to apply Lemma 6.21, we must check $\mathscr{H}$ is maximal, satisfies $\vec{\nabla}_\exists$, $\vec{\nabla}_\xi$ (if $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$), and $\vec{\nabla}_\mathfrak{f}$ (if $* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$). It is immediate from the construction that $\vec{\nabla}_\exists$ holds since if $\neg(\Pi^\alpha \boldsymbol{F}) \in \mathscr{H}$, then $\neg(\boldsymbol{F} w_\alpha^\delta) \in \mathscr{H}$ where δ is the ordinal such that $\boldsymbol{A}^\delta \equiv \neg(\Pi^\alpha \boldsymbol{F})$. If $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$, then we have ensured $\vec{\nabla}_\xi$ holds since $\neg([w_\alpha^\delta/X]\boldsymbol{M} \doteq^\beta [w_\alpha^\delta/X]\boldsymbol{N}) \in \mathscr{H}$ whenever $\neg((\lambda X_\alpha.\boldsymbol{M}) \doteq^{\alpha\to\beta} (\lambda X.\boldsymbol{N})) \in \mathscr{H}$ where δ is the ordinal such that $\boldsymbol{A}^\delta \equiv \neg((\lambda X_\alpha.\boldsymbol{M}) \doteq^{\alpha\to\beta} (\lambda X.\boldsymbol{N}))$. Similarly, we have ensured $\vec{\nabla}_\mathfrak{f}$ holds when $* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$ since $\neg(\boldsymbol{F} w_\alpha^\delta \doteq^\beta \boldsymbol{G} w_\alpha^\delta) \in \mathscr{H}$ whenever $\neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G}) \in \mathscr{H}$ where δ is the ordinal such that $\boldsymbol{A}^\delta \equiv \neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G})$.

It only remains to show that $\mathscr{H}$ is maximal in Γ_Σ. So, let $\boldsymbol{A} \in \mathrm{cwff}_o$ and $\mathscr{H} * \boldsymbol{A} \in \Gamma_\Sigma$ be given. Note that $\boldsymbol{A} \equiv \boldsymbol{A}^\delta$ for some $\delta < \varepsilon$. Since $\mathscr{H}$ is closed under subsets we know that $\mathscr{H}^\delta * \boldsymbol{A}^\delta \in \Gamma_\Sigma$. By definition of $\mathscr{H}^{\delta+1}$ we conclude that $\boldsymbol{A}^\delta \in \mathscr{H}^{\delta+1}$ and hence $\boldsymbol{A} \in \mathscr{H}$.

So, Lemma 6.21 implies $\mathscr{H} \in \mathfrak{Hint}_*$ and $\mathscr{H}$ is saturated if Γ_Σ is saturated. ⊣

We now use the Σ-Hintikka sets, guaranteed by Lemma 6.32, to construct a Σ-valuation for the Σ-term evaluation that turns it into a model.

THEOREM 6.33 (Model existence theorem for saturated sets). *For all* $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ *we have*: *If* $\mathscr{H}$ *is a saturated Hintikka set in* $\mathfrak{Hint}_*$ (*cf. Definition* 6.20), *then there exists a model* $\mathscr{M} \in \mathfrak{M}_*$ (*cf. Definition* 3.49) *that satisfies* $\mathscr{H}$. *Furthermore, each domain* $\mathscr{D}_\alpha$ *of* $\mathscr{M}$ *has cardinality at most* $\aleph_s$.

PROOF. We start with the construction of a Σ-model $\mathscr{M}_1^\mathscr{H}$ for $\mathscr{H}$ based on the term evaluation $\mathscr{TE}(\Sigma)^\beta$. This model may not be in the model class $\mathfrak{M}_*$ as it may not satisfy property $\mathfrak{q}$. However, we will be able to use Theorem 3.62 to obtain a model of $\mathscr{H}$ which is.

Note that since $\mathscr{H}$ is saturated, by Lemma 6.25, $\mathscr{H}$ satisfies $\overline{\nabla}_\neg$, $\overline{\nabla}_\vee$, and $\overline{\nabla}_\forall^\beta$.

The domain of type α of the evaluation $\mathscr{TE}(\Sigma)^\beta$ (cf. Definition 3.35 and Lemma 3.36) is $\mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$, which has cardinality $\aleph_s$. To construct $\mathscr{M}_1^\mathscr{H}$, we simply need to give a valuation function for this evaluation. This valuation function should be a function $\upsilon\colon \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta \longrightarrow \{\mathtt{T}, \mathtt{F}\}$. We define

$$\upsilon(\boldsymbol{A}) := \begin{cases} \mathtt{T} & \text{if } \boldsymbol{A} \in \mathscr{H}, \\ \mathtt{F} & \text{if } \boldsymbol{A} \notin \mathscr{H}. \end{cases}$$

To show υ is a valuation, we must check the logical constants are interpreted appropriately. For each $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$, we have $\upsilon(\neg\boldsymbol{A}) \equiv \mathtt{T}$ iff $\upsilon(\boldsymbol{A}) \equiv \mathtt{F}$ since $\neg\boldsymbol{A} \in \mathscr{H}$ iff $\boldsymbol{A} \notin \mathscr{H}$ by $\overline{\nabla}_\neg$. For each $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$, we have $\upsilon(\boldsymbol{A} \vee \boldsymbol{B}) \equiv \mathtt{T}$ iff $\upsilon(\boldsymbol{A}) \equiv \mathtt{T}$ or $\upsilon(\boldsymbol{B}) \equiv \mathtt{T}$, since $(\boldsymbol{A} \vee \boldsymbol{B}) \in \mathscr{H}$ iff $\boldsymbol{A} \in \mathscr{H}$ or $\boldsymbol{B} \in \mathscr{H}$ by $\overline{\nabla}_\vee$. Finally, for each type α and $\boldsymbol{F} \in \mathrm{cwff}_{\alpha\to o}(\Sigma)\big\downarrow_\beta$, $\overline{\nabla}_\forall^\beta$ implies $(\Pi^\alpha \boldsymbol{F}) \in \mathscr{H}$ iff $(\boldsymbol{F}\boldsymbol{A})\big\downarrow_\beta \in \mathscr{H}$ for every $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$. Thus, we have $\upsilon(\Pi^\alpha \boldsymbol{F}) \equiv \mathtt{T}$ iff $\upsilon(\boldsymbol{F} @^\beta \boldsymbol{A}) \equiv \mathtt{T}$ for every $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$.

This verifies $\mathscr{M}_1^\mathscr{H} := (\mathrm{cwff}\big\downarrow_\beta, @^\beta, \mathscr{E}^\beta, \upsilon)$ is a Σ-model. Clearly, $\mathscr{M}_1^\mathscr{H} \models \mathscr{H}$ since $\upsilon(\boldsymbol{A}) \equiv \mathtt{T}$ for every $\boldsymbol{A} \in \mathscr{H}$ by definition.

By Theorem 3.62, we have a congruence relation $\dot{\sim}$ on $\mathscr{M}_1^\mathscr{H}$ induced by Leibniz equality. Note that by Lemma 3.61 in the term model $\mathscr{M}_1^\mathscr{H}$, for every type α and

every $\boldsymbol{A}, \boldsymbol{B} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$, we have $\boldsymbol{A}_\alpha \stackrel{\cdot}{\sim} \boldsymbol{B}_\alpha$, iff $v(\boldsymbol{A} \doteq \boldsymbol{B}) \equiv \mathsf{T}$, iff $(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \mathscr{H}$. Furthermore, if primitive equality is in the signature, then $\mathscr{H} \in \mathfrak{Hint}_*$ is a Hintikka set with primitive equality. Hence, $\mathscr{H}$ satisfies $\overline{\nabla}_{\doteq}^{\doteq}$ by Lemma 6.22. We have $\boldsymbol{A} \stackrel{\cdot}{\sim} \boldsymbol{B}$, iff $(\boldsymbol{A} \doteq^\alpha \boldsymbol{B}) \in \mathscr{H}$, iff (by $\overline{\nabla}_{\doteq}^{\doteq}$) $(\boldsymbol{A} =^\alpha \boldsymbol{B}) \in \mathscr{H}$, iff $v(\mathscr{E}^\beta(=^\alpha)@^\beta \boldsymbol{A}@^\beta \boldsymbol{B}) \equiv \mathsf{T}$.

Let $\mathscr{M} := \mathscr{M}_1^{\mathscr{H}}/_{\stackrel{\cdot}{\sim}}$. Each domain of this model has cardinality at most $\aleph_s$ as it is the quotient of a set of cardinality $\aleph_s$. By Theorem 3.62, we know the quotient model $\mathscr{M}$ models $\mathscr{H}$, satisfies property $\mathfrak{q}$, and is a model with primitive equality (if primitive equality is in the signature). Hence, $\mathscr{M} \in \mathfrak{M}_\beta$. Now, we can use Lemma 3.58 to check $\mathscr{M} \in \mathfrak{M}_*$ by checking certain properties of $\stackrel{\cdot}{\sim}$.

When $* \in \{\beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$, we must check that $\stackrel{\cdot}{\sim}$ has only two equivalence classes in $\mathscr{D}_o^\beta$. To show this, first note that $\overline{\nabla}_{\mathfrak{b}}$ holds for $\mathscr{H}$ by Lemma 6.26. Choose any β-normal $\boldsymbol{B} \in \mathscr{H}$. By $\vec{\nabla}_c$, $\neg\boldsymbol{B} \notin \mathscr{H}$. By $\overline{\nabla}_{\mathfrak{b}}$, for every $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$ either $(\boldsymbol{A} \doteq^o \boldsymbol{B})$ or $(\boldsymbol{A} \doteq^o \neg\boldsymbol{B})$. That is, in $\mathscr{M}_1^{\mathscr{H}}$, for every $\boldsymbol{A} \in \mathrm{cwff}_o(\Sigma)\big\downarrow_\beta$ we either have $\boldsymbol{A} \stackrel{\cdot}{\sim} \boldsymbol{B}$ or $\boldsymbol{A} \stackrel{\cdot}{\sim} \neg\boldsymbol{B}$. So, we know $\mathscr{M}$ satisfies property $\mathfrak{b}$.

When $* \in \{\beta\eta, \beta\eta\mathfrak{b}\}$, the fact that $\stackrel{\cdot}{\sim}$ satisfies property η follows from $\overline{\nabla}_\eta$ which holds for $\mathscr{H}$ by Lemma 6.27.

When $* \in \{\beta\xi, \beta\xi\mathfrak{b}\}$, we must show that $\stackrel{\cdot}{\sim}$ satisfies property ξ. Let $\boldsymbol{M}, \boldsymbol{N} \in \mathrm{wff}_\beta(\Sigma)$, an assignment φ and a variable X_α be given. Suppose $\mathscr{E}^\beta_{\varphi,[\boldsymbol{A}/X]}(\boldsymbol{M}) \stackrel{\cdot}{\sim} \mathscr{E}^\beta_{\varphi,[\boldsymbol{A}/X]}(\boldsymbol{N})$ for every $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$. Let θ be the substitution defined by $\theta(Y) := \varphi(Y)$ for each variable $Y \in (\mathrm{free}(\boldsymbol{M}) \cup \mathrm{free}(\boldsymbol{N})) \setminus \{X\}$. So, for each $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$,

$$([\boldsymbol{A}/X]\theta(\boldsymbol{M}))\big\downarrow_\beta \equiv \mathscr{E}^\beta_{\varphi,[\boldsymbol{A}/X]}(\boldsymbol{M}) \stackrel{\cdot}{\sim} \mathscr{E}^\beta_{\varphi,[\boldsymbol{A}/X]}(\boldsymbol{N}) \equiv ([\boldsymbol{A}/X]\theta(\boldsymbol{N}))\big\downarrow_\beta.$$

That is, $([\boldsymbol{A}/X]\theta(\boldsymbol{M}) \doteq^\beta [\boldsymbol{A}/X]\theta(\boldsymbol{N}))\big\downarrow_\beta \in \mathscr{H}$ for every $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$. By $\overline{\nabla}_\xi^\beta$ (Lemma 6.28), we have $((\lambda X_\alpha.\theta(\boldsymbol{M})) \doteq^{\alpha\to\beta} \lambda X_\alpha.\theta(\boldsymbol{N}))\big\downarrow_\beta \in \mathscr{H}$. So,

$$\mathscr{E}^\beta_\varphi(\lambda X_\alpha.\boldsymbol{M}) \equiv (\lambda X_\alpha.\theta(\boldsymbol{M}))\big\downarrow_\beta \stackrel{\cdot}{\sim} (\lambda X_\alpha.\theta(\boldsymbol{N}))\big\downarrow_\beta \equiv \mathscr{E}^\beta_\varphi(\lambda X_\alpha.\boldsymbol{N}).$$

Thus, $\stackrel{\cdot}{\sim}$ satisfies ξ as desired.

When $* \in \{\beta\mathfrak{f}, \beta\mathfrak{f}\mathfrak{b}\}$, we must show $\stackrel{\cdot}{\sim}$ is functional. Let α and β be types and $\boldsymbol{G}, \boldsymbol{H} \in \mathrm{cwff}_{\alpha\to\beta}(\Sigma)\big\downarrow_\beta$. We need to show $\boldsymbol{G} \stackrel{\cdot}{\sim} \boldsymbol{H}$ iff $(\boldsymbol{G}\boldsymbol{A})\big\downarrow_\beta \stackrel{\cdot}{\sim} (\boldsymbol{H}\boldsymbol{A})\big\downarrow_\beta$ for every $\boldsymbol{A} \in \mathrm{cwff}_\alpha(\Sigma)\big\downarrow_\beta$. This follows directly from $\overline{\nabla}_{\mathfrak{f}}^\beta$.

This verifies the fact that $\mathscr{M} \in \mathfrak{M}_*$ whenever $\mathscr{H} \in \mathfrak{Hint}_*$. ⊣

THEOREM 6.34 (Model existence theorem). *Let Γ_Σ be a saturated abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ-pure set of sentences. For all $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we have: If Γ_Σ is an $\mathfrak{Acc}_*$ (cf. Definition 6.7), then there exists a model $\mathscr{M} \in \mathfrak{M}_*$ (cf. Definition 3.49) that satisfies Φ. Furthermore, each domain of $\mathscr{M}$ has cardinality at most $\aleph_s$.*

PROOF. Let Γ_Σ be an abstract consistency class. We can assume without loss of generality (cf. Lemma 6.18) that Γ_Σ is compact, so the preconditions of Lemma 6.32 are met. Therefore, there exists a saturated Hintikka set $\mathscr{H} \in \mathfrak{Hint}_*$ with $\Phi \subseteq \mathscr{H}$. The proof is completed by a simple appeal to the Theorem 6.33. ⊣

THEOREM 6.35 (Model existence for Henkin models). *Let* Γ_Σ *be a saturated abstract consistency class in* $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$ *and let* $\Phi \in \Gamma_\Sigma$ *be a sufficiently* Σ*-pure set of sentences. Then there is a Henkin model* (*cf. Definition* 3.50) *that satisfies* Φ. *Furthermore, each domain of the model has cardinality at most* $\aleph_s$.

PROOF. By Theorem 6.34, there is a model $\mathscr{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ with $\mathscr{M} \models \Phi$. By Theorem 3.68, there is a Henkin model $\mathscr{M}^{fr} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ isomorphic to $\mathscr{M}$. By the isomorphism, we have $\mathscr{M}^{fr} \models \Phi$ and that each domain of $\mathscr{M}^{fr}$ has the same cardinality as the corresponding domain of $\mathscr{M}$. ⊣

REMARK 6.36. The model existence theorems show there are "enough" models in each class $\mathfrak{M}_*$ to model sufficiently pure sets in saturated abstract consistency classes in $\mathfrak{Acc}_*$. These results are abstract forms of completeness. To complete the analysis, we can show abstract forms of soundness. One way to show this is to define a class of sentences

$$\Gamma^*_\Sigma := \{\, \Phi \subseteq \mathrm{cwff}_o \mid \exists \mathscr{M} \in \mathfrak{M}_* . \mathscr{M} \models \Phi \,\}$$

for each $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ and show Γ^*_Σ is a (saturated) $\mathfrak{Acc}_*$. We only sketch the proof here.

The fact that each Γ^*_Σ satisfy ∇_c, ∇_β, $\nabla_\neg$, $\nabla_\vee$, $\nabla_\wedge$, $\nabla_\forall$, and ∇_{sat} is straightforward. The proof that $\nabla_\exists$ holds has the technical difficulty that one must modify the evaluation of a parameter. Showing $\nabla_\mathfrak{b}$ [∇_η] holds when considering models with property $\mathfrak{b}$ [η] is also easy.

When showing $\nabla_\mathfrak{f}$ holds in $\Gamma^{\beta\mathfrak{f}}_\Sigma$ [$\Gamma^{\beta\mathfrak{f}\mathfrak{b}}_\Sigma$], one sees the importance of assuming property $\mathfrak{q}$ holds. Suppose $\Phi \in \Gamma^{\beta\mathfrak{f}}_\Sigma$ [$\Gamma^{\beta\mathfrak{f}\mathfrak{b}}_\Sigma$] and $\neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G}) \in \Phi$. Then there is a model $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon) \in \mathfrak{M}_{\beta\mathfrak{f}}$ [$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$] such that $\mathscr{M} \models \Phi$. This implies $\mathscr{M} \models \neg(\boldsymbol{F} \doteq^{\alpha\to\beta} \boldsymbol{G})$. Without using property $\mathfrak{q}$, it follows by Lemma 4.2(1) that $\mathscr{E}(\boldsymbol{F}) \not\equiv \mathscr{E}(\boldsymbol{G})$. By functionality, there is an $\mathsf{a} \in \mathscr{D}_\alpha$ such that $\mathscr{E}(\boldsymbol{F})@\mathsf{a} \not\equiv \mathscr{E}(\boldsymbol{G})@\mathsf{a}$. Let φ be any assignment into $\mathscr{M}$. Then $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{F}X) \not\equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{G}X)$. Now, using property $\mathfrak{q}$, we can conclude $\mathscr{M}_{\varphi,[\mathsf{a}/X]} \models \neg((\boldsymbol{F}X) \doteq^\beta (\boldsymbol{G}X))$ by Lemma 4.2(2). Let $w_\alpha \in \Sigma$ be a parameter that does not occur in any sentence of Φ. With some technical work which we omit, one can change the evaluation function to $\mathscr{E}'$ so that $\mathscr{E}'(\boldsymbol{A}) \equiv \mathscr{E}(\boldsymbol{A})$ for all $\boldsymbol{A} \in \Phi$, and $\mathscr{E}'(w) \equiv \mathsf{a}$. In the new model $\mathscr{M}' \equiv (\mathscr{D}, @, \mathscr{E}', \upsilon)$, we have $\mathscr{M}' \models \Phi$ and $\mathscr{M}' \models \neg(\boldsymbol{F}w \doteq^\beta \boldsymbol{G}w)$. Also, $\mathscr{M}' \in \mathfrak{Acc}_{\beta\mathfrak{f}}$ [$\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$]. This shows $\Phi * \neg(\boldsymbol{F}w \doteq^\beta \boldsymbol{G}w) \in \Gamma^{\beta\mathfrak{f}}_\Sigma$ [$\Gamma^{\beta\mathfrak{f}\mathfrak{b}}_\Sigma$]. The proof that ∇_ξ holds in $\Gamma^{\beta\xi}_\Sigma$ [$\Gamma^{\beta\xi\mathfrak{b}}_\Sigma$] is analogous.

We have now established a set of proof-theoretic conditions that are sufficient to guarantee the existence of a model.

§7. Characterizing higher-order natural deduction calculi. In this section we apply the model existence theorems above to prove some classical higher-order calculi of natural deduction sound and complete with respect to the model classes introduced in Section 3. The first calculus for such a formulation of higher-order logic was a Hilbert-style system introduced by Alonzo Church in [18][10]. Leon Henkin proves completeness (with respect to Henkin models) for a similar calculus with full extensionality in [26]. Peter Andrews introduced a weaker calculus $\mathfrak{T}_\beta$ [1], which lacks all

[10]Church included functional extensionality axioms but only mentions the Boolean extensionality axiom as an option.

$$\frac{\boldsymbol{A} \in \Phi}{\Phi \vdash \boldsymbol{A}}\,\mathfrak{NK}(Hyp) \qquad \frac{\boldsymbol{A} \equiv_\beta \boldsymbol{B} \quad \Phi \vdash \boldsymbol{A}}{\Phi \vdash \boldsymbol{B}}\,\mathfrak{NK}(\beta)$$

$$\frac{\Phi * \boldsymbol{A} \vdash \boldsymbol{F}_o}{\Phi \vdash \neg \boldsymbol{A}}\,\mathfrak{NK}(\neg I) \qquad \frac{\Phi \vdash \neg \boldsymbol{A} \quad \Phi \vdash \boldsymbol{A}}{\Phi \vdash \boldsymbol{C}}\,\mathfrak{NK}(\neg E)$$

$$\frac{\Phi \vdash \boldsymbol{A}}{\Phi \vdash \boldsymbol{A} \vee \boldsymbol{B}}\,\mathfrak{NK}(\vee I_L) \qquad \frac{\Phi \vdash \boldsymbol{B}}{\Phi \vdash \boldsymbol{A} \vee \boldsymbol{B}}\,\mathfrak{NK}(\vee I_R)$$

$$\frac{\Phi \vdash \boldsymbol{A} \vee \boldsymbol{B} \quad \Phi * \boldsymbol{A} \vdash \boldsymbol{C} \quad \Phi * \boldsymbol{B} \vdash \boldsymbol{C}}{\Phi \vdash \boldsymbol{C}}\,\mathfrak{NK}(\vee E)$$

$$\frac{\Phi \vdash \boldsymbol{G} w_\alpha \quad w \text{ parameter not occurring in } \Phi \text{ or } \boldsymbol{G}}{\Phi \vdash \Pi^\alpha \boldsymbol{G}}\,\mathfrak{NK}(\Pi I)^w$$

$$\frac{\Phi \vdash \Pi^\alpha \boldsymbol{G}}{\Phi \vdash \boldsymbol{G}\boldsymbol{A}}\,\mathfrak{NK}(\Pi E) \qquad \frac{\Phi * \neg \boldsymbol{A} \vdash \boldsymbol{F}_o}{\Phi \vdash \boldsymbol{A}}\,\mathfrak{NK}(Contr)$$

FIGURE 6. Inference rules for $\mathfrak{NK}_\beta$.

forms of extensionality. This calculus has been widely used as a syntactic measure of completeness for machine-oriented calculi [1, 32, 33, 34, 42, 36, 37].

Instead of applying our methods to Hilbert-style calculi, we will use a collection of natural deduction calculi to avoid the tedious details of proving a deduction theorem and propositional completeness. Moreover, natural deduction calculi are more relevant in practice. They form the logical basis for semi-automated theorem proving systems such as HOL [25], ISABELLE [46], or ΩMEGA [51].

DEFINITION 7.1 (The calculi $\mathfrak{NK}_*$). The calculus $\mathfrak{NK}_\beta$ consists of the inference rules[11] in Figure 6 for the provability judgment $\vdash$ between sets of sentences Φ and sentences $\boldsymbol{A}$. (We write $\vdash \boldsymbol{A}$ for $\emptyset \vdash \boldsymbol{A}$.) The rule $\mathfrak{NK}(\beta)$ incorporates β-equality into $\vdash$. The others characterize the semantics of the connectives and quantifiers.

For $* \in \{\beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we obtain the calculus $\mathfrak{NK}_*$ by adding the rules shown in Figure 7 when specified in $*$.

REMARK 7.2. It is worth noting that there is a derivation of $\vdash \boldsymbol{T}_o$ (i.e., $\vdash \forall P_0.\, P \vee \neg P$) which only uses the rules in Figure 6. Let p be a parameter of type o. A derivation of $\neg(p \vee \neg p) \vdash (p \vee \neg p)$ is shown in Figure 8. Using $\mathfrak{NK}(Hyp)$ and

[11] Recall that $\boldsymbol{F}_o$ is defined to be $\neg(\forall P_o.(P \vee \neg P))$ and $\mathscr{M} \not\models \boldsymbol{F}_o$ for each Σ-model $\mathscr{M}$ (cf. Lemma 3.43).

$$\frac{\boldsymbol{A}\equiv_{\beta\eta}\boldsymbol{B} \quad \Phi \Vdash \boldsymbol{A}}{\Phi \Vdash \boldsymbol{B}}\,\mathfrak{NK}(\eta) \qquad \frac{\Phi \Vdash \forall X_\alpha.\boldsymbol{M} \doteq^{\beta} \boldsymbol{N}}{\Phi \Vdash (\lambda X_\alpha.\boldsymbol{M}) \doteq^{\alpha\to\beta} (\lambda X_\alpha.\boldsymbol{N})}\,\mathfrak{NK}(\xi)$$

$$\frac{\Phi \Vdash \forall X_\alpha.\boldsymbol{G}X \doteq^{\beta} \boldsymbol{H}X}{\Phi \Vdash \boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}}\,\mathfrak{NK}(\mathfrak{f})$$

$$\frac{\Phi * \boldsymbol{A} \Vdash \boldsymbol{B} \quad \Phi * \boldsymbol{B} \Vdash \boldsymbol{A}}{\Phi \Vdash \boldsymbol{A} \doteq^{o} \boldsymbol{B}}\,\mathfrak{NK}(\mathfrak{b})$$

FIGURE 7. Extensional inference rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(p\vee\neg p), p \Vdash p}\,\mathfrak{NK}(Hyp)}{\neg(p\vee\neg p), p \Vdash (p\vee\neg p)}\,\mathfrak{NK}(\vee I_L) \qquad \dfrac{}{\neg(p\vee\neg p), p \Vdash \neg(p\vee\neg p)}\,\mathfrak{NK}(Hyp)}{\neg(p\vee\neg p), p \Vdash \boldsymbol{F}_o}\,\mathfrak{NK}(\neg E)}{\neg(p\vee\neg p) \Vdash \neg p}\,\mathfrak{NK}(\neg I)}{\neg(p\vee\neg p) \Vdash (p\vee\neg p)}\,\mathfrak{NK}(\vee I_R)$$

FIGURE 8. Derivation of $\neg(p \vee \neg p) \Vdash (p \vee \neg p)$.

$\mathfrak{NK}(\neg E)$, we obtain $\neg(p \vee \neg p) \Vdash \boldsymbol{F}_o$. So, we can conclude $\Vdash (p \vee \neg p)$ using $\mathfrak{NK}(Contr)$. Finally, we obtain a derivation of $\Vdash \boldsymbol{T}_o$ using $\mathfrak{NK}(\Pi I)^p$. Hence, $\Vdash \boldsymbol{T}_o$ is derivable in each calculus $\mathfrak{NK}_*$ where $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Also, we can apply the rule $\mathfrak{NK}(\Pi E)$ to the end of this derivation with any sentence $\boldsymbol{A}$ to derive $\Vdash (\boldsymbol{A} \vee \neg\boldsymbol{A})$.

Note that $\mathfrak{NK}_\beta$ and $\mathfrak{NK}_{\beta\mathfrak{f}\mathfrak{b}}$ correspond to the extremes of the model classes discussed in Section 3 (cf. Figure 1 in the introduction). Standard models do not admit (recursively axiomatizable) calculi that are sound and complete, $\mathfrak{NK}_{\beta\mathfrak{f}\mathfrak{b}}$ is complete for Henkin models, and $\mathfrak{NK}_\beta$ is complete for $\mathfrak{M}_\beta$. We will now show soundness and completeness of each $\mathfrak{NK}_*$ with respect to each corresponding model class $\mathfrak{M}_*$ by using the model existence theorems in Section 6.

THEOREM 7.3 (Soundness). *$\mathfrak{NK}_*$ is sound for $\mathfrak{M}_*$ for $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. That is, if $\Phi \Vdash_{\mathfrak{NK}_*} \boldsymbol{C}$ is derivable, then $\mathscr{M} \models \boldsymbol{C}$ for all models $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, v)$ in $\mathfrak{M}_*$ such that $\mathscr{M} \models \Phi$.*

PROOF. This can be shown by a simple induction on the derivation of $\Phi \Vdash_{\mathfrak{NK}_*} \boldsymbol{C}$. We distinguish based on the last rule of the derivation. The only base case is $\mathfrak{NK}(Hyp)$, which is trivial since $\mathscr{M} \models \boldsymbol{C}$ whenever $\mathscr{M} \models \Phi$ and $\boldsymbol{C} \in \Phi$.

$\mathfrak{NR}(\beta)$: Suppose $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash \boldsymbol{A}$ and $\boldsymbol{A} \equiv_\beta \boldsymbol{C}$. Let $\mathscr{M} \in \mathfrak{M}_*$ be a model of Φ. By induction, we know $\mathscr{M} \models \boldsymbol{A}$ and so $\mathscr{M} \models \boldsymbol{C}$ using Remark 3.19.

$\mathfrak{NR}(\mathit{Contr})$: Suppose $\mathscr{M} \in \mathfrak{M}_*$, $\mathscr{M} \models \Phi$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi * \neg\boldsymbol{C} \vdash \boldsymbol{F}_o$. By Lemma 3.43, $\mathscr{M} \not\models \boldsymbol{F}_o$. So, we must have $\mathscr{M} \not\models \neg\boldsymbol{C}$. Hence, $\mathscr{M} \models \boldsymbol{C}$.

$\mathfrak{NR}(\neg I)$: Analogous to $\mathfrak{NR}(\mathit{Contr})$.

$\mathfrak{NR}(\neg E)$: Suppose $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash \neg\boldsymbol{A}$ and $\Phi \vdash \boldsymbol{A}$. By induction, any model in $\mathfrak{M}_*$ of Φ would have to model both $\boldsymbol{A}$ and $\neg\boldsymbol{A}$. So, there is no such model of Φ and we are done.

$\mathfrak{NR}(\vee I_L)$: Suppose $\mathscr{M} \in \mathfrak{M}_*$, $\mathscr{M} \models \Phi$, $\boldsymbol{C}$ is $(\boldsymbol{A} \vee \boldsymbol{B})$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash \boldsymbol{A}$. By induction, $\mathscr{M} \models \boldsymbol{A}$ and so $\mathscr{M} \models (\boldsymbol{A} \vee \boldsymbol{B})$.

$\mathfrak{NR}(\vee I_R)$: Analogous to $\mathfrak{NR}(\vee I_L)$.

$\mathfrak{NR}(\vee E)$: Suppose $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash (\boldsymbol{A} \vee \boldsymbol{B})$, $\Phi * \boldsymbol{A} \vdash \boldsymbol{C}$ and $\Phi * \boldsymbol{B} \vdash \boldsymbol{C}$. Let $\mathscr{M} \in \mathfrak{M}_*$ be a model of Φ. By induction, $\mathscr{M} \models \boldsymbol{A} \vee \boldsymbol{B}$. If $\mathscr{M} \models \boldsymbol{A}$, then by induction $\mathscr{M} \models \boldsymbol{C}$ since $\Phi * \boldsymbol{A} \vdash \boldsymbol{C}$. If $\mathscr{M} \models \boldsymbol{B}$, then by induction $\mathscr{M} \models \boldsymbol{C}$ since $\Phi * \boldsymbol{B} \vdash \boldsymbol{C}$. In either case, $\Phi \vdash \boldsymbol{C}$.

$\mathfrak{NR}(\Pi I)$: Suppose $\boldsymbol{C}$ is $(\Pi^\alpha \boldsymbol{G})$ and $\Phi \vdash (\Pi^\alpha \boldsymbol{G})$ follows from $\Phi \vdash \boldsymbol{G}w$ where w_α is a parameter which does not occur in any sentence of Φ or in $\boldsymbol{G}$. Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon) \in \mathfrak{M}_*$ be a model of Φ. Assume $\mathscr{M} \not\models \Pi^\alpha \boldsymbol{G}$. Then there must be some $\mathsf{a} \in \mathscr{D}_\alpha$ such that $\upsilon(\mathscr{E}(\boldsymbol{G})@\mathsf{a}) \equiv \mathtt{F}$. From the evaluation function $\mathscr{E}$, one can define another evaluation function $\mathscr{E}'$ such that $\mathscr{E}'(w) \equiv \mathsf{a}$ and $\mathscr{E}'_\varphi(\boldsymbol{A}_\alpha) \equiv \mathscr{E}_\varphi(\boldsymbol{A}_\alpha)$ if w does not occur in $\boldsymbol{A}$. Let $\mathscr{M}' := (\mathscr{D}, @, \mathscr{E}', \upsilon)$. One can check $\mathscr{M}' \in \mathfrak{M}_*$ using the fact that $\mathscr{M} \in \mathfrak{M}_*$. Since $\mathscr{M}' \models \Phi$, by induction we have $\mathscr{M}' \models \boldsymbol{G}w$. This contradicts $\upsilon(\mathscr{E}'(\boldsymbol{G})@\mathsf{a}) \equiv \upsilon(\mathscr{E}(\boldsymbol{G})@\mathsf{a}) \equiv \mathtt{F}$. Thus, $\mathscr{M} \models \Pi^\alpha \boldsymbol{G}$.

$\mathfrak{NR}(\Pi E)$: Suppose $\boldsymbol{C}$ is $(\boldsymbol{GA})$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash (\Pi^\alpha \boldsymbol{G})$. Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon) \in \mathfrak{M}_*$ be a model of Φ. By induction, $\mathscr{M} \models (\Pi^\alpha \boldsymbol{G})$ and thus $\upsilon(\mathscr{E}(\boldsymbol{G}))@\mathsf{a} \equiv \mathtt{T}$ for every $\mathsf{a} \in \mathscr{D}_\alpha$. In particular, $\mathscr{M} \models \boldsymbol{GA}$.

We now check soundness of the rules in Figure 7 with respect to their model classes:

$\mathfrak{NR}(\eta)$: Analogous to $\mathfrak{NR}(\beta)$ using property η.

$\mathfrak{NR}(\xi)$: Suppose $\boldsymbol{C}$ is $(\lambda X_\alpha . \boldsymbol{M}) \doteq^{\alpha\to\beta} (\lambda X_\alpha . \boldsymbol{N})$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash \forall X_\alpha . \boldsymbol{M} \doteq^\beta \boldsymbol{N}$. Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon) \in \mathfrak{M}_*$ be a model of Φ. By induction, we have $\mathscr{M} \models \forall X_\alpha . \boldsymbol{M} \doteq^\beta \boldsymbol{N}$. So, for any assignment φ and $\mathsf{a} \in \mathscr{D}_\alpha$, $\mathscr{M} \models_{\varphi,[\mathsf{a}/X]} \boldsymbol{M} \doteq^\beta \boldsymbol{N}$. Note that property $\mathfrak{q}$ holds in $\mathscr{M}$ since $\mathscr{M} \in \mathfrak{M}_*$ (cf. Definition 3.49). By Lemma 4.2(2), $\mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{M}) \equiv \mathscr{E}_{\varphi,[\mathsf{a}/X]}(\boldsymbol{N})$. By property ξ, $\mathscr{E}_\varphi(\lambda X_\alpha . \boldsymbol{M}) \equiv \mathscr{E}_\varphi(\lambda X_\alpha . \boldsymbol{N})$ and thus $\mathscr{M} \models \boldsymbol{C}$ by Lemma 4.2(1).

$\mathfrak{NR}(\mathfrak{f})$: Suppose $\boldsymbol{C}$ is $\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H}$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi \vdash \forall X_\alpha . \boldsymbol{G}X \doteq^\beta \boldsymbol{H}X$. Let $\mathscr{M} \in \mathfrak{M}_*$ be a model of Φ. By induction, we know $\mathscr{M} \models \forall X_\alpha . \boldsymbol{G}X \doteq^\beta \boldsymbol{H}X$. Note that property $\mathfrak{q}$ holds for $\mathscr{M}$ since $\mathscr{M} \in \mathfrak{M}_*$. By Theorem 4.3(3), we must have $\mathscr{M} \models (\boldsymbol{G} \doteq^{\alpha\to\beta} \boldsymbol{H})$.

$\mathfrak{NR}(\mathfrak{b})$ Suppose $\boldsymbol{C}$ is $\boldsymbol{A} \doteq^o \boldsymbol{B}$ and $\Phi \vdash \boldsymbol{C}$ follows from $\Phi * \boldsymbol{A} \vdash \boldsymbol{B}$ and $\Phi * \boldsymbol{B} \vdash \boldsymbol{A}$. Let $\mathscr{M} \equiv (\mathscr{D}, @, \mathscr{E}, \upsilon) \in \mathfrak{M}_*$ be a model of Φ. If $\mathscr{M} \models \boldsymbol{A}$, then $\mathscr{M} \models \boldsymbol{B}$ by induction. If $\mathscr{M} \models \boldsymbol{B}$, then $\mathscr{M} \models \boldsymbol{A}$ by induction. These facts imply $\upsilon(\mathscr{E}(\boldsymbol{A})) \equiv \upsilon(\mathscr{E}(\boldsymbol{B}))$. By Lemma 3.48, we have $\mathscr{M} \models (\boldsymbol{A} \Leftrightarrow \boldsymbol{B})$. By Theorem 4.3(4), we must have $\mathscr{M} \models (\boldsymbol{A} \doteq^o \boldsymbol{B})$. ⊣

DEFINITION 7.4 ($\mathfrak{NR}_*$-consistent). A set of sentences Φ is $\mathfrak{NR}_*$-*inconsistent* if $\Phi \vDash_{\mathfrak{NR}_*} \boldsymbol{F}_o$, and $\mathfrak{NR}_*$-*consistent* otherwise.

Now, we use the model existence theorems for $\mathscr{HOL}$ to give short and elegant proofs of completeness for $\mathfrak{NR}_*$.

LEMMA 7.5. *The class* $\Gamma^*_\Sigma := \{\Phi \subseteq \mathrm{cwff}_o \mid \Phi \textit{ is } \mathfrak{NR}_*\textit{-consistent}\}$ *is a saturated* $\mathfrak{Acc}_*$.

PROOF. Obviously Γ^*_Σ is closed under subsets, since any subset of an $\mathfrak{NR}_*$-consistent set is $\mathfrak{NR}_*$-consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c: Suppose $\boldsymbol{A}, \neg\boldsymbol{A} \in \Phi$. We have $\Phi \vDash \boldsymbol{F}_o$ by $\mathfrak{NR}(\mathit{Hyp})$ and $\mathfrak{NR}(\neg E)$.

∇_β: Let $\boldsymbol{A} \in \Phi$, $\boldsymbol{A} \equiv_\beta \boldsymbol{B}$ and $\Phi * \boldsymbol{B}$ be $\mathfrak{NR}_*$-inconsistent. That is, $\Phi * \boldsymbol{B} \vDash \boldsymbol{F}_o$. By $\mathfrak{NR}(\neg I)$, we know $\Phi \vDash \neg\boldsymbol{B}$. Since $\boldsymbol{A} \in \Phi$, we know $\Phi \vDash \boldsymbol{B}$ by $\mathfrak{NR}(\mathit{Hyp})$ and $\mathfrak{NR}(\beta)$. Using $\mathfrak{NR}(\neg E)$, we know $\Phi \vDash \boldsymbol{F}_o$ and hence Φ is $\mathfrak{NR}_*$-inconsistent.

$\nabla_\neg$: Suppose $\neg\neg\boldsymbol{A} \in \Phi$ and $\Phi * \boldsymbol{A}$ is $\mathfrak{NR}_*$-inconsistent. From $\Phi * \boldsymbol{A} \vDash \boldsymbol{F}_o$ and $\mathfrak{NR}(\neg I)$, we have $\Phi \vDash \neg\boldsymbol{A}$. Since $\neg\neg\boldsymbol{A} \in \Phi$, we can apply $\mathfrak{NR}(\mathit{Hyp})$ and $\mathfrak{NR}(\neg E)$ to obtain $\Phi \vDash \boldsymbol{F}_o$.

$\nabla_\vee$: Suppose $(\boldsymbol{A} \vee \boldsymbol{B}) \in \Phi$ and both $\Phi * \boldsymbol{A}$ and $\Phi * \boldsymbol{B}$ are $\mathfrak{NR}_*$-inconsistent. By $\mathfrak{NR}(\mathit{Hyp})$ and $\mathfrak{NR}(\vee E)$, we have $\Phi \vDash \boldsymbol{F}_o$.

$\nabla_\wedge$: Suppose $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \Phi$ and $\Phi * \neg\boldsymbol{A} * \neg\boldsymbol{B}$ is $\mathfrak{NR}_*$-inconsistent. By $\mathfrak{NR}(\mathit{Contr})$ and $\mathfrak{NR}(\vee I_R)$, we have $\Phi, \neg\boldsymbol{A} \vDash \boldsymbol{A} \vee \boldsymbol{B}$. Using $\mathfrak{NR}(\neg E)$ with $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \Phi$, we have $\Phi, \neg\boldsymbol{A} \vDash \boldsymbol{F}_o$. By $\mathfrak{NR}(\mathit{Contr})$ and $\mathfrak{NR}(\vee I_L)$, we have $\Phi \vDash \boldsymbol{A} \vee \boldsymbol{B}$. Using $\mathfrak{NR}(\neg E)$ with $\neg(\boldsymbol{A} \vee \boldsymbol{B}) \in \Phi$, Φ is $\mathfrak{NR}_*$-inconsistent.

$\nabla_\forall$: Suppose $(\Pi^\alpha \boldsymbol{G}) \in \Phi$ and $\Phi * (\boldsymbol{GA})$ is $\mathfrak{NR}_*$-inconsistent. By $\mathfrak{NR}(\neg I)$, $\Phi \vDash \neg(\boldsymbol{GA})$. By $\mathfrak{NR}(\mathit{Hyp})$ and $\mathfrak{NR}(\Pi E)$, $\Phi \vDash \boldsymbol{GA}$. Finally, $\mathfrak{NR}(\neg E)$ implies $\Phi \vDash \boldsymbol{F}_o$.

$\nabla_\exists$: Suppose $\neg(\Pi^\alpha \boldsymbol{G}) \in \Phi$, w_α is a parameter which does not occur in Φ, and $\Phi * \neg(\boldsymbol{G}w)$ is $\mathfrak{NR}_*$-inconsistent. By $\mathfrak{NR}(\mathit{Contr})$, $\Phi \vDash \boldsymbol{G}w$. By $\mathfrak{NR}(\Pi I)^w$, $\Phi \vDash (\Pi^\alpha \boldsymbol{G})$. Using $\mathfrak{NR}(\neg E)$ with $\neg(\Pi^\alpha \boldsymbol{G}) \in \Phi$, Φ is $\mathfrak{NR}_*$-inconsistent.

∇_{sat}: Let $\Phi * \boldsymbol{A}$ and $\Phi * \neg\boldsymbol{A}$ be $\mathfrak{NR}_*$-inconsistent. We show that Φ is $\mathfrak{NR}_*$-inconsistent. Using $\mathfrak{NR}(\neg I)$, we know $\Phi \vDash \neg\boldsymbol{A}$ and $\Phi \vDash \neg\neg\boldsymbol{A}$. By $\mathfrak{NR}(\neg E)$, we have $\Phi \vDash \boldsymbol{F}_o$.

Thus we have shown that Γ^β_Σ is saturated and in $\mathfrak{Acc}_\beta$. Now let us check the conditions for the additional properties η, ξ, $\mathfrak{f}$, and $\mathfrak{b}$.

∇_η: If $*$ includes η, then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{NR}(\eta)$.

∇_ξ: Suppose $*$ includes ξ, $\neg(\lambda X_\alpha \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X_\alpha \boldsymbol{N}) \in \Phi$, and $\Phi * \neg([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})$ is $\mathfrak{NR}_*$-inconsistent for some parameter w_α which does not occur in any sentence of Φ. By $\mathfrak{NR}(\mathit{Contr})$, we have $\Phi \vDash ([w/X]\boldsymbol{M} \doteq^\beta [w/X]\boldsymbol{N})$. By $\mathfrak{NR}(\beta)$, we have $\Phi \vDash ((\lambda X_\alpha \boldsymbol{M} \doteq^\beta \boldsymbol{N})w)$. By $\mathfrak{NR}(\Pi I)$, $\Phi \vDash (\forall X_\alpha \boldsymbol{M} \doteq^\beta \boldsymbol{N})$. By $\mathfrak{NR}(\xi)$, $\Phi \vDash (\lambda X_\alpha \boldsymbol{M} \doteq^{\alpha\to\beta} \lambda X_\alpha \boldsymbol{N})$. By $\mathfrak{NR}(\neg E)$, Φ is $\mathfrak{NR}_*$-inconsistent.

$\nabla_\mathfrak{f}$: This case is analogous to the previous one, generalizing $\lambda X_\alpha \boldsymbol{M} \doteq \lambda X_\alpha \boldsymbol{N}$ to arbitrary $\boldsymbol{G} \doteq \boldsymbol{H}$ and using the extensionality rule $\mathfrak{NR}(\mathfrak{f})$ instead of $\mathfrak{NR}(\xi)$.

$\nabla_\mathfrak{b}$: Suppose $*$ includes $\mathfrak{b}$. Assume that $\neg(\boldsymbol{A} \doteq^o \boldsymbol{B}) \in \Phi$ but both $\Phi * \neg\boldsymbol{A} * \boldsymbol{B} \notin \Gamma^*_\Sigma$ and $\Phi * \boldsymbol{A} * \neg\boldsymbol{B} \notin \Gamma^*_\Sigma$. So both are $\mathfrak{NR}_*$-inconsistent and we have $\Phi * \boldsymbol{A} \vDash \boldsymbol{B}$ and $\Phi * \boldsymbol{B} \vDash \boldsymbol{A}$ by $\mathfrak{NR}(\mathit{Contr})$. By $\mathfrak{NR}(\mathfrak{b})$, we have $\Phi \vDash (\boldsymbol{A} \doteq^o \boldsymbol{B})$. Since $\neg(\boldsymbol{A} \doteq^o \boldsymbol{B}) \in \Phi$, Φ is $\mathfrak{NR}_*$-inconsistent. ⊣

THEOREM 7.6 (Henkin's theorem for $\mathfrak{NK}_*$). *Let* $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. *Every sufficiently* Σ*-pure* $\mathfrak{NK}_*$*-consistent set of sentences has an* $\mathfrak{M}_*$*-model.*

PROOF. Let Φ be a sufficiently Σ-pure $\mathfrak{NK}_*$-consistent set of sentences. By Theorem 7.5 we know that the class of sets of $\mathfrak{NK}_*$-consistent sentences constitute a saturated $\mathfrak{Acc}_*$, thus the Model Existence Theorem (Theorem 6.34) guarantees an $\mathfrak{M}_*$ model for Φ. ⊣

COROLLARY 7.7 (Completeness theorem for $\mathfrak{NK}_*$). *Let* Φ *be a sufficiently* Σ*-pure set of sentences,* $\boldsymbol{A}$ *be a sentence, and* $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. *If* $\boldsymbol{A}$ *is valid in all models* $\mathscr{M} \in \mathfrak{M}_*$ *that satisfy* Φ, *then* $\Phi \vDash_{\mathfrak{NK}_*} \boldsymbol{A}$.

PROOF. Let $\boldsymbol{A}$ be given such that $\boldsymbol{A}$ is valid in all $\mathfrak{M}_*$ models that satisfy Φ. So, $\Phi * \neg\boldsymbol{A}$ is unsatisfiable in $\mathfrak{M}_*$. Since only finitely many constants occur in $\neg\boldsymbol{A}$, $\Phi * \neg\boldsymbol{A}$ is sufficiently Σ-pure. So, $\Phi * \neg\boldsymbol{A}$ must be $\mathfrak{NK}_*$-inconsistent by Henkin's theorem above. Thus, $\Phi \vDash_{\mathfrak{NK}_*} \boldsymbol{A}$ by $\mathfrak{NK}(\mathit{Contr})$. ⊣

Finally we can use the completeness theorems obtained so far to prove a compactness theorem for our semantics.

COROLLARY 7.8 (Compactness theorem for $\mathfrak{NK}_*$). *Let* Φ *be a sufficiently* Σ*-pure set of sentences and* $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Φ *has an* $\mathfrak{M}_*$*-model iff every finite subset of* Φ *has an* $\mathfrak{M}_*$*-model.*

PROOF. If Φ has no $\mathfrak{M}_*$-model, then by Theorem 7.6 Φ is $\mathfrak{NK}_*$-inconsistent. Since every $\mathfrak{NK}_*$-proof is finite, this means some finite subset Ψ of Φ is $\mathfrak{NK}_*$-inconsistent. Hence, Ψ has no $\mathfrak{M}_*$-model. ⊣

REMARK 7.9 (Calculi with primitive equality). If primitive equality is included in the signature, a simple way of extending the calculi $\mathfrak{NK}_*$ in a sound and complete way is to include the rules $\mathfrak{NK}(=^r)$ and $\mathfrak{NK}(=^l)$ in Figure 9. These rules are clearly sound for models with primitive equality. One can argue completeness by showing $\Gamma^*_\Sigma := \{\Phi \subseteq \mathrm{wff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{NK}_*\text{-consistent}\}$ is a saturated $\mathfrak{Acc}_*$ with primitive equality. By Lemma 7.5, we already know Γ^*_Σ is a saturated $\mathfrak{Acc}_*$. To show the conditions for primitive equality, one can show Γ^*_Σ satisfies $\nabla^r_=$ using $\mathfrak{NK}(=^r)$ and $\nabla^{\doteq}_=$ using $\mathfrak{NK}(=^l)$.

$$\frac{}{\Phi \vDash \boldsymbol{A} =^\alpha \boldsymbol{A}}\,\mathfrak{NK}(=^r) \qquad \frac{\Phi \vDash \boldsymbol{C} =^\alpha \boldsymbol{D}}{\Phi \vDash \boldsymbol{C} \doteq^\alpha \boldsymbol{D}}\,\mathfrak{NK}(=^l)$$

FIGURE 9. Primitive equality in $\mathfrak{NK}_*$.

§8. Conclusion. In this article, we have given an overview of the landscape of semantics for classical higher-order logics. We have differentiated nine different possible notions and have tied the discerning properties to conditions of corresponding abstract consistency classes. The practical relevance of these notions has been illustrated by pointing to application scenarios within mathematics, programming languages, and computational linguistics.

Our model existence theorems are strong proof tools connecting syntax and semantics. A standard application is in completeness analysis of higher-order calculi. A calculus $\mathscr{C}$ is shown to be complete for a model class $\mathfrak{M}_*$ by showing that the class of $\mathscr{C}$-consistent or $\mathscr{C}$-irrefutable sets of sentences is in $\mathfrak{Acc}_*$. Then completeness follows from the model existence results. We have given an example of this by showing completeness for natural deduction calculi in Section 7.

8.1. Applications and related work. The generalized model classes $\mathfrak{M}_*$ have many possible applications. An example is higher-order logic programming [45] where the denotational semantics of programs can induce non-standard meanings for the classical connectives. For instance, given an SLD-like search strategy as in λ-PROLOG [43], conjunction is not commutative any more. Therefore, various authors have proposed model-theoretic semantics where property $\mathfrak{b}$ fails. David Wolfram, for instance, uses Andrews' υ-complexes [58] as a semantics for λ-PROLOG and Gopalan Nadathur uses "labeled structures" for the same purpose in [45]. Mary DeMarco [20] also develops a model theory for intuitionistic type theory and λ-prolog in which property $\mathfrak{b}$ may fail (James Lipton and Mary DeMarco are continuing this work). Till Mossakowski and Lutz Schröder have been studying non-functional Henkin models for a partial λ-calculus in the context of the HasCasl specification language [48, 49]. It is plausible to assume that the results of this article will be useful for further development in this direction. Further relevance of model-theoretic semantics where property $\mathfrak{q}$ fails, however, is not sufficiently investigated yet, but seems a promising line of research.

The article also provides a basis for the investigation of hyper-intensional semantics of natural languages. In fact early versions of this article have already influenced the work of Lappin and Pollard [40]. Hyper-intensional semantics provide theories for logics where Boolean extensionality (and thus the substitutability of equivalents) can fail. Linguistically motivated theories like the ones presented in [56, 17, 41, 40] introduce intensional (non-standard) variants of the connectives and quantifiers acting on a generalized domain of truth values. Interestingly, only [41] and [40] present formal model-theoretic semantics. The model construction in [41] strongly resembles Peter Andrew's υ-complexes (semantic objects are paired with syntactic representations; in this case linguistic parse trees). In [40], $\mathscr{D}_o$ is taken to be a pre-Boolean algebra, and possible worlds are associated with ultrafilters. A direct comparison is aggravated by the fact that Lappin and Pollard's work is situated in a Montague-style intensional (i.e., modal) context. A generalization of our work by techniques from [23] seems the way to go here.

8.2. Relaxing the saturation assumption. Unfortunately, the model existence theorems presented in this article do not support completeness proofs for most higher-order machine-oriented calculi, such as higher-order resolution [33, 13], higher-order paramodulation [11], or tableau-based calculi [5, 37]. This is because we had to assume saturation of abstract consistency classes to prove the model existence theorems. The problem is that machine oriented calculi are typically, in some sense, cut-free. This makes saturation very difficult to show.

For the same reason the results of this article also do not apply to another prominent application of model existence theorems: relatively simple (but non-constructive) cut-elimination theorems. In [1] Peter Andrews applies his "Unifying Principle" to cut-elimination in a cut-free non-extensional sequent calculus, by

proving the calculus complete (relative to $\mathfrak{T}_\beta$). He concludes that cut-elimination is valid for this calculus. Again, the saturation condition prevents us from obtaining variants of the extensional cut-elimination theorems in [54, 55] by Andrews' approach using our model existence theorem for Henkin models. In fact one can prove (cf. [12]) that the problem of showing that an abstract consistency class can be extended to a saturated one is equivalent to showing cut elimination for certain sequent or resolution calculi.

To account for the saturation problem we have additionally investigated model existence for the model classes presented in this article using an extension of Peter Andrews' υ-complexes (cf. [12]). The model construction in this technique requires an abstract consistency class to satisfy certain *acceptability* conditions which are much weaker than saturation. (For example, the acceptability conditions can be shown to hold for abstract consistency classes obtained from certain cut-free sequent calculi.) Because this technique is much more complex and subtle than the relatively simple quotients of term evaluations used in this article, we did not include the extended results here. The unsaturated model existence theorems imply that every acceptable abstract consistency class can be extended to a saturated one. Armed with this fact, we can use the model existence theorems presented here to rescue the general completeness and cut elimination results mentioned above. To show, for example, completeness of a higher-order machine-oriented calculus $\mathscr{C}$, we define the class Γ of $\mathscr{C}$-irrefutable sentences and show that it is an acceptable (but unsaturated) abstract consistency class. By the extension result in [12] there is a saturated abstract consistency class $\Gamma' \supseteq \Gamma$. By application of saturated model existence from this article we obtain a suitable model for every (sufficiently Σ-pure) $\Phi \in \Gamma'$ and thus for every (sufficiently Σ-pure) $\Phi \in \Gamma$. This immediately gives us completeness. Hence, the leverage added by this article together with [12] is that we can now extend non-extensional cut-elimination results to extensional cases.

Acknowledgments. The work presented in this paper has been supported by the "Deutsche Forschungsgemeinschaft" (DFG) under Grant SI 372/4 HOTEL, the National Science Foundation under Grant CCR-0097179 and a DFG Heisenberg stipend (Ko-1370/6-1) to the third author. The authors would like to thank Peter Andrews and Frank Pfenning for stimulating discussions and Claus-Peter Wirth and Andrey Paskevich for proof reading. We furthermore thank the referee of this article for his very fruitful comments.

REFERENCES

[1] PETER B. ANDREWS, *Resolution in type theory*, this JOURNAL, vol. 36 (1971), no. 3, pp. 414–432.

[2] ———, *General models and extensionality*, this JOURNAL, vol. 37 (1972), no. 2, pp. 395–397.

[3] ———, *General models descriptions and choice in type theory*, this JOURNAL, vol. 37 (1972), no. 2, pp. 385–394.

[4] ———, letter to Roger Hindley dated January 22, 1973.

[5] ———, *On connections and higher order logic*, ***Journal of Automated Reasoning***, vol. 5 (1989), pp. 257–291.

[6] ———, ***An introduction to mathematical logic and type theory: To truth through proof***, second ed., Kluwer Academic Publishers, 2002.

[7] PETER B. ANDREWS, MATTHEW BISHOP, and CHAD E. BROWN, *TPS: A theorem proving system for type theory*, ***Proceedings of the 17th international conference on automated deduction*** (Pittsburgh, USA) (David McAllester, editor), Lecture Notes in Artifical Intelligence, no. 1831, Springer-Verlag, 2000, pp. 164–169.

[8] PETER B. ANDREWS, MATTHEW BISHOP, SUNIL ISSAR, DAN NESMITH, FRANK PFENNING, and HONGWEI XI, *TPS: A theorem proving system for classical type theory*, ***Journal of Automated Reasoning***, vol. 16 (1996), no. 3, pp. 321–353.

[9] HENK P. BARENDREGT, ***The lambda calculus***, North-Holland, 1984.

[10] CHRISTOPH BENZMÜLLER, *Equality and extensionality in automated higher-order theorem proving*, ***Ph.D. thesis***, Saarland University, 1999.

[11] ———, *Extensional higher-order paramodulation and RUE-resolution*, ***Proceedings of the 16th international Conference on Automated Deduction*** (Trento, Italy) (Harald Ganzinger, editor), Lecture Notes in Artificial Intelligence, vol. 1632, Springer-Verlag, 1999, pp. 399–413.

[12] CHRISTOPH BENZMÜLLER, CHAD E. BROWN, and MICHAEL KOHLHASE, *Semantic techniques for higher-order cut-elimination*, manuscript, `http://www.ags.uni-sb.de/~chris/papers/R19.pdf`, 2002.

[13] CHRISTOPH BENZMÜLLER and MICHAEL KOHLHASE, *Extensional higher order resolution*, in Kirchner and Kirchner [35], pp. 56–72.

[14] ———, *LEO—a higher order theorem prover*, in Kirchner and Kirchner [35], pp. 139–144.

[15] ———, *Model existence for higher-order logic*, ***SEKI-Report SR-97-09***, Saarland University, 1997.

[16] Wolfgang Bibel and Peter Schmitt (editors), ***Automated deduction—a basis for applications***, Kluwer, 1998.

[17] GENNARO CHIERCHIA and RAYMOND TURNER, *Semantics and property theory*, ***Linguistics and Philosophy***, vol. 11 (1988), pp. 261–302.

[18] ALONZO CHURCH, *A formulation of the simple theory of types*, this JOURNAL, vol. 5 (1940), pp. 56–68.

[19] NICOLAAS GOVERT DE BRUIJN, *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with an application to the Church-Rosser theorem*, ***Indagationes Mathematicae***, vol. 34 (1972), no. 5, pp. 381–392.

[20] MARY DEMARCO, *Intuitionistic semantics for heriditarily harrop logic programming*, ***Ph.D. thesis***, Wesleyan University, 1999.

[21] GILLES DOWEK, THÉRÈSE HARDIN, and CLAUDE KIRCHNER, *HOL-λσ an intentional first-order expression of higher-order logic*, ***Mathematical Structures in Computer Science***, vol. 11 (2001), no. 1, pp. 1–25.

[22] MELVIN FITTING, ***First-order logic and automated theorem proving***, second ed., Graduate Texts in Computer Science, Springer-Verlag, 1996.

[23] ———, ***Types, tableaus, and Gödel's God***, Kluwer, 2002.

[24] KURT GÖDEL, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, ***Monatshefte der Mathematischen Physik***, vol. 38 (1931), pp. 173–198, English version in [57].

[25] M. J. C. GORDON and T. F. MELHAM, ***Introduction to HOL—a theorem proving environment for higher order logic***, Cambridge University Press, 1993.

[26] LEON HENKIN, *Completeness in the theory of types*, this JOURNAL, vol. 15 (1950), no. 2, pp. 81–91.

[27] ———, *The discovery of my completeness proofs*, ***The Bulletin of Symbolic Logic***, vol. 2 (1996), no. 2, pp. 127–158.

[28] ROGER J. HINDLEY and JONATHAN P. SELDIN, ***Introduction to combinators and lambda-calculs***, Cambridge University Press, Cambridge, 1986.

[29] K. J. J. HINTIKKA, *Form and content in quantification theory*, ***Acta Philosophica Fennica***, vol. 8 (1955), pp. 7–55.

[30] FURIO HONSELL and MARINA LENISA, *Coinductive characterizations of applicative structures*, ***Mathematical Structures in Computer Science***, vol. 9 (1999), pp. 403–435.

[31] FURIO HONSELL and DONALD SANNELLA, *Pre-logical relations*, ***Proceedings of computer science logic*** (CSL '99), Lecture Notes in Computer Science, vol. 1683, Springer-Verlag, 1999, pp. 546–561.

[32] GÉRARD P. HUET, *Constrained resolution: A complete method for higher order logic*, ***Ph. D. thesis***, Case Western Reserve University, 1972.

[33] ———, *A mechanization of type theory*, ***Proceedings of the 3rd international joint conference on artificial intelligence*** (Donald E. Walker and Lewis Norton, editors), 1973, pp. 139–146.

[34] D. C. JENSEN and THOMASZ PIETRZYKOWSKI, *A complete mechanization of* (ω)*-order type theory*, ***Proceedings of the ACM annual conference***, vol. 1, 1972, pp. 82–92.

[35] Claude Kirchner and Hélène Kirchner (editors), ***Proceedings of the 15th Conference on Automated Deduction***, Lecture Notes in Artificial Intelligence, vol. 1421, Springer-Verlag, 1998.

[36] MICHAEL KOHLHASE, *A mechanization of sorted higher-order logic based on the resolution principle*, ***Ph. D. thesis***, Saarland University, 1994.

[37] ———, *Higher-order tableaux*, ***Theorem proving with analytic tableaux and related methods*** (Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors), Lecture Notes in Artificial Intelligence, vol. 918, Springer-Verlag, 1995, pp. 294–309.

[38] MICHAEL KOHLHASE and ORTWIN SCHEJA, *Higher-order multi-valued resolution*, ***Journal of Applied Non-Classical Logics***, vol. 9 (1999), no. 4, pp. 155–178.

[39] SHALOM LAPPIN and CARL POLLARD, *Strategies for hyperintensional semantics*, manuscript, King's College, London and Ohio State University, 2000.

[40] ———, *A higher-order fine-grained logic for intensional semantics*, manuscript, 2002.

[41] RICHARD LARSON and GABRIEL SEGAL, ***Knowledge of meaning***, MIT Press, 1995.

[42] DALE MILLER, *Proofs in higher-order logic*, ***Ph. D. thesis***, Carnegie-Mellon University, 1983.

[43] ———, *A logic programming language with lambda-abstraction, function variables, and simple unification*, ***Journal of Logic and Computation***, vol. 4 (1991), no. 1, pp. 497–536.

[44] JOHN C. MITCHELL, ***Foundations for programming languages***, Foundations of Computing, MIT Press, 1996.

[45] GOPALAN NADATHUR and DALE MILLER, *Higher-order logic programming*, ***Technical Report CS-1994-38***, Department of Computer Science, Duke University, 1994.

[46] TOBIAS NIPKOW, LAWRENCE C. PAULSON, and MARKUS WENZEL, ***Isabelle/HOL—a proof assistant for higher-order logic***, Lecture Notes in Computer Science, vol. 2283, Springer-Verlag, 2002.

[47] J. ALAN ROBINSON and ANDREI VORONKOV, ***Handbook of automated reasoning***, MIT Press, 2001.

[48] L. SCHRÖDER and T. MOSSAKOWSKI, *Hascasl: towards integrated specification and development of functional programs*, ***Algebraic methodology and software technology***, Lecture Notes in Computer Science, vol. 2422, Springer-Verlag, 2002, pp. 99–116.

[49] LUTZ SCHRÖDER, *Henkin models for the partial λ-calculus*, manuscript, `http://www.informatik.uni-bremen.de/~lschrode/hascasl/henkin.ps`, 2002.

[50] KURT SCHÜTTE, *Semantical and syntactical properties of simple type theory*, this JOURNAL, vol. 25 (1960), pp. 305–326.

[51] JÖRG SIEKMANN, CHRISTOPH BENZMÜLLER, et al., *Proof development with OMEGA*, ***Proceedings of the 18th international conference on automated deduction*** (Copenhagen, Denmark) (Andrei Voronkov, editor), Lecture Notes in Artificial Intelligence, vol. 2392, Springer-Verlag, 2002, pp. 144–149.

[52] RAYMOND M. SMULLYAN, *A unifying principle for quantification theory*, ***Proceedings of the National Academy of Sciences***, vol. 49 (1963), pp. 828–832.

[53] ———, ***First-order logic***, Springer-Verlag, 1968.

[54] MOTO-O TAKAHASHI, *Cut-elimination in simple type theory with extensionality*, ***Journal of the Mathematical Society of Japan***, vol. 19 (1967), pp. 399–410.

[55] GAISI TAKEUTI, ***Proof theory***, North-Holland, 1987.

[56] R. TOMASON, *A model theory for propositional attitudes*, ***Linguistics and Philosophy***, vol. 4 (1980), pp. 47–70.

[57] JEAN VAN HEIJENOORT, ***From Frege to Gödel: a source book in mathematical logic 1879–1931,*** 3rd printing, 1997 ed., Source books in the history of the sciences series, Harvard University Press, Cambridge, MA, 1967.

[58] DAVID A. WOLFRAM, *A semantics for λ-PROLOG*, ***Theoretical Computer Science***, vol. 136 (1994), no. 1, pp. 277–289.

DEPARTMENT OF COMPUTER SCIENCE
SAARLAND UNIVERSITY
SAARBRÜCKEN, GERMANY
E-mail: chris@ags.uni-sb.de
URL: http://www.ags.uni-sb.de/~chris

DEPARTMENT OF MATHEMATICS
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PA 15213, USA
E-mail: cebrown@andrew.cmu.edu
URL: http://www.andrew.cmu.edu/~cebrown/

SCHOOL OF ENGINEERING AND SCIENCES
INTERNATIONAL UNIVERSITY BREMEN
BREMEN, GERMANY
and
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, USA
E-mail: m.kohlhase@iu-bremen.de
URL: http://www.cs.cmu.edu/~kohlhase

Cut-Simulation in Impredicative Logics

Christoph E. Benzmüller[1], Chad E. Brown[1], and Michael Kohlhase[2]

[1] Saarland University, Saarbrücken, Germany (chris|cebrown@ags.uni-sb.de)
[2] International University Bremen, Bremen, Germany (m.kohlhase@iu-bremen.de)

Abstract. We investigate cut-elimination and cut-simulation in impredicative (higher-order) logics. We illustrate that adding simple axioms such as Leibniz equations to a calculus for an impredicative logic — in our case a sequent calculus for classical type theory — is like adding cut. The phenomenon equally applies to prominent axioms like Boolean- and functional extensionality, induction, choice, and description. This calls for the development of calculi where these principles are built-in instead of being treated axiomatically.

1 Introduction

One of the key questions of automated reasoning is the following: "When does a set Φ of sentences have a model?" In fact, given reasonable assumptions about calculi, most inference problems can be reduced to determining (un)-satisfiability of a set Φ of sentences. Since building models for Φ is hard in practice, much research in computational logic has concentrated on finding sufficient conditions for satisfiability, e.g. whether there is a Hintikka set $\mathcal{H}$ extending Φ.

Of course in general the answer to the satisfiability question depends on the class of models at hand. In classical first-order logic, model classes are well-understood. In impredicative higher-order logic, there is a whole landscape of plausible model classes differing in their treatment of functional and Boolean extensionality. Satisfiability then strongly depends on these classes, for instance, the set $\Phi := \{a, b, qa, \neg qb\}$ is unsatisfiable in a model class where the universes of Booleans are required to have at most two members (see property $\mathfrak{b}$ below), but satisfiable in the class without this restriction.

In [5] we have shown that certain (i.e. *saturated*) Hintikka sets always have models and have derived syntactical conditions (so-called *saturated* abstract consistency properties) for satisfiability from this fact. The importance of abstract consistency properties is that one can check completeness for a calculus $\mathcal{C}$ by verifying proof-theoretic conditions (checking that $\mathcal{C}$-irrefutable sets of formulae have the saturated abstract consistency property) instead of performing model-theoretic analysis (for historical background of the method in first-order logic, cf. [10, 13, 14]). Unfortunately, the saturation condition (if Φ is abstractly consistent, then one of $\Phi \cup \{\mathbf{A}\}$ or $\Phi \cup \{\neg\mathbf{A}\}$ is as well for all sentences $\mathbf{A}$) is very difficult to prove for machine-oriented calculi (indeed as hard as cut elimination).

In this paper we investigate further the relation between the lack of the subformula property in the saturation condition (we need to "guess" whether

to extend Φ by $\mathbf{A}$ or $\neg\mathbf{A}$ on our way to a Hintikka set for all sentences $\mathbf{A}$) and the cut rule (where we have to "guess, i.e. search for in an automated reasoning setting" the cut formula $\mathbf{A}$). A side result is the insight that there exist "cut-strong" formulae which support the effective simulation of cut in calculi for impredicative logics.

In Section 2, we will fix notation and review the relevant results from [5]. We define in Section 3 a basic sequent calculus and study the correspondence between saturation in abstract consistency classes and cut-elimination. In Section 4 we introduce the notion of "cut-strong" formulae and sequents and show that they support the effective simulation of cut. In Section 5 we demonstrate that the pertinent extensionality axioms are cut-strong. We develop alternative extensionality rules which do not suffer from this problem. Further rules are needed to ensure Henkin completeness for this calculus with extensionality. These new rules correspond to the acceptability conditions we propose in Section 6 to ensure the existence of models and the existence of saturated extensions of abstract consistence classes.

2 Higher-Order Logic

In [5] we have re-examined the semantics of classical higher-order logic with the purpose of clarifying the role of extensionality. For this we have defined eight classes of higher-order models with respect to various combinations of Boolean extensionality and three forms of functional extensionality. We have also developed a methodology of abstract consistency (by providing the necessary model existence theorems) needed for instance, to analyze completeness of higher-order calculi with respect to these model classes. We now briefly summarize the main notions and results of [5] as required for this paper. Our impredicative logic of choice is Church's classical type theory.

Syntax: Church's Simply Typed λ-Calculus. As in [9], we formulate higher-order logic ($\mathcal{HOL}$) based on the simply typed λ-calculus. The set of simple types $\mathcal{T}$ is freely generated from basic types o and ι using the function type constructor $\to$.

For formulae we start with a set $\mathcal{V}$ of (typed) variables (denoted by $X_\alpha, Y, Z, X^1_\beta, X^2_\gamma \ldots$) and a signature Σ of (typed) constants (denoted by $c_\alpha, f_{\alpha\to\beta}, \ldots$). We let $\mathcal{V}_\alpha$ (Σ_α) denote the set of variables (constants) of type α. The signature Σ of constants includes the logical constants $\neg_{o\to o}$, $\vee_{o\to o\to o}$ and $\Pi^\alpha_{(\alpha\to o)\to o}$ for each type α; all other constants in Σ are called parameters. As in [5], we assume there is an infinite cardinal $\aleph_s$ such that the cardinality of Σ_α is $\aleph_s$ for each type α (cf. [5](3.16)). The set of $\mathcal{HOL}$-formulae (or terms) are constructed from typed variables and constants using application and λ-abstraction. We let $\mathit{wff}_\alpha(\Sigma)$ be the set of all terms of type α and $\mathit{wff}(\Sigma)$ be the set of all terms.

We use vector notation to abbreviate k-fold applications and abstractions as $\mathbf{A}\overline{\mathbf{U}^k}$ and $\lambda\overline{X^k}.\mathbf{A}$, respectively. We also use Church's dot notation so that . stands for a (missing) left bracket whose mate is as far to the right as possible (consistent with given brackets). We use infix notation $\mathbf{A} \vee \mathbf{B}$ for $((\vee\mathbf{A})\mathbf{B})$ and binder

notation $\forall X_\alpha . \mathbf{A}$ for $(\Pi^\alpha(\lambda X_\alpha . \mathbf{A}_o))$. We further use $\mathbf{A} \wedge \mathbf{B}$, $\mathbf{A} \Rightarrow \mathbf{B}$, $\mathbf{A} \Leftrightarrow \mathbf{B}$ and $\exists X_\alpha . \mathbf{A}$ as shorthand for formulae defined in terms of $\neg$, $\vee$ and Π^α (cf. [5]). Finally, we let $(\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha)$ denote the Leibniz equation $\forall P_{\alpha\to o} . (P\mathbf{A}) \Rightarrow . P\mathbf{B}$.

Each occurrence of a variable in a term is either bound by a λ or free. We use $free(\mathbf{A})$ to denote the set of free variables of $\mathbf{A}$ (i.e., variables with a free occurrence in $\mathbf{A}$). We consider two terms to be equal if the terms are the same up to the names of bound variables (i.e., we consider α-conversion implicitly). A term $\mathbf{A}$ is closed if $free(\mathbf{A})$ is empty. We let $cwff_\alpha(\Sigma)$ denote the set of closed terms of type α and $cwff(\Sigma)$ denote the set of all closed terms. Each term $\mathbf{A} \in wff_o(\Sigma)$ is called a proposition and each term $\mathbf{A} \in cwff_o(\Sigma)$ is called a sentence.

We denote substitution of a term $\mathbf{A}_\alpha$ for a variable X_α in a term $\mathbf{B}_\beta$ by $[\mathbf{A}/X]\mathbf{B}$. Since we consider α-conversion implicitly, we assume the bound variables of $\mathbf{B}$ avoid variable capture.

Two common relations on terms are given by β-reduction and η-reduction. A β-redex $(\lambda X . \mathbf{A})\mathbf{B}$ β-reduces to $[\mathbf{B}/X]\mathbf{A}$. An η-redex $(\lambda X . \mathbf{C}X)$ (where $X \notin free(\mathbf{C})$) η-reduces to $\mathbf{C}$. For $\mathbf{A}, \mathbf{B} \in wff_\alpha(\Sigma)$, we write $\mathbf{A} \equiv_\beta \mathbf{B}$ to mean $\mathbf{A}$ can be converted to $\mathbf{B}$ by a series of β-reductions and expansions. Similarly, $\mathbf{A} \equiv_{\beta\eta} \mathbf{B}$ means $\mathbf{A}$ can be converted to $\mathbf{B}$ using both β and η. For each $\mathbf{A} \in wff(\Sigma)$ there is a unique β-normal form (denoted $\mathbf{A}{\downarrow_\beta}$) and a unique $\beta\eta$-normal form (denoted $\mathbf{A}{\downarrow_{\beta\eta}}$). From this fact we know $\mathbf{A} \equiv_\beta \mathbf{B}$ ($\mathbf{A} \equiv_{\beta\eta} \mathbf{B}$) iff $\mathbf{A}{\downarrow_\beta} \equiv \mathbf{B}{\downarrow_\beta}$ ($\mathbf{A}{\downarrow_{\beta\eta}} \equiv \mathbf{B}{\downarrow_{\beta\eta}}$).

A non-atomic formula in $wff_o(\Sigma)$ is any formula whose β-normal form is of the form $[c\overline{\mathbf{A}^n}]$ where c is a logical constant. An atomic formula is any other formula in $wff_o(\Sigma)$.

Semantics: Eight Model Classes. For each $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ (the latter set will be abbreviated by $\boxplus$ in the remainder) we define $\mathfrak{M}_*$ to be the class of all Σ-models $\mathcal{M}$ such that $\mathcal{M}$ satisfies property $\mathfrak{q}$ and each of the additional properties $\{\eta, \xi, \mathfrak{f}, \mathfrak{b}\}$ indicated in the subscript $*$ (cf. [5](3.49)). Special cases of Σ-models are Henkin models and standard models (cf. [5](3.50 and 3.51)). Every model in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ is isomorphic to a Henkin model (see the discussion following [5](3.68)).

Saturated Abstract Consistency Classes and Model Existence. Finally, we review the model existence theorems proved in [5]. There are three stages to obtaining a model in our framework. First, we obtain an abstract consistency class $\mathit{\Gamma}_\Sigma$ (usually defined as the class of irrefutable sets of sentences with respect to some calculus). Second, given a (sufficiently pure) set of sentences Φ in the abstract consistency class $\mathit{\Gamma}_\Sigma$ we construct a Hintikka set $\mathcal{H}$ extending Φ. Third, we construct a model of this Hintikka set (and hence a model of Φ).

A Σ-abstract consistency class $\mathit{\Gamma}_\Sigma$ is a class of sets of Σ-sentences. An abstract consistency class is always required to be closed under subsets (cf. [5](6.1)). Sometimes we require the stronger property that $\mathit{\Gamma}_\Sigma$ is compact, i.e., a set Φ is in $\mathit{\Gamma}_\Sigma$ iff every finite subset of Φ is in $\mathit{\Gamma}_\Sigma$ (cf. [5](6.1,6.2)).

To describe further properties of abstract consistency classes, we use the notation $S * a$ for $S \cup \{a\}$ as in [5]. The following is a list of properties a class $\mathit{\Gamma}_\Sigma$ of sets of sentences can satisfy with respect to arbitrary $\Phi \in \mathit{\Gamma}_\Sigma$ (cf. [5](6.5)):

∇_c If $\mathbf{A}$ is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.
$\nabla_\neg$ If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.
∇_β If $\mathbf{A} \equiv_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
∇_η If $\mathbf{A} \equiv_{\beta\eta} \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
$\nabla_\vee$ If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
$\nabla_\wedge$ If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.
$\nabla_\forall$ If $\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi * \mathbf{FW} \in \Gamma_\Sigma$ for each $\mathbf{W} \in cwff_\alpha(\Sigma)$.
$\nabla_\exists$ If $\neg\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi * \neg(\mathbf{F}w) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
$\nabla_\mathfrak{b}$ If $\neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Phi$, then $\Phi * \mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$ or $\Phi * \neg\mathbf{A} * \mathbf{B} \in \Gamma_\Sigma$.
∇_ξ If $\neg(\lambda X_\alpha.\mathbf{M} \doteq^{\alpha\to\beta} \lambda X_\alpha.\mathbf{N}) \in \Phi$, then $\Phi * \neg([w/X]\mathbf{M} \doteq^\beta [w/X]\mathbf{N}) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
$\nabla_\mathfrak{f}$ If $\neg(\mathbf{G} \doteq^{\alpha\to\beta} \mathbf{H}) \in \Phi$, then $\Phi * \neg(\mathbf{G}w \doteq^\beta \mathbf{H}w) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ.
∇_{sat} Either $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \neg\mathbf{A} \in \Gamma_\Sigma$.

We say Γ_Σ is an abstract consistency class if it is closed under subsets and satisfies $\nabla_c, \nabla_\neg, \nabla_\beta, \nabla_\vee, \nabla_\wedge, \nabla_\forall$ and $\nabla_\exists$. We let $\mathfrak{Acc}_\beta$ denote the collection of all abstract consistency classes. For each $* \in \mathfrak{B}$ we refine $\mathfrak{Acc}_\beta$ to a collection $\mathfrak{Acc}_*$ where the additional properties $\{\nabla_\eta, \nabla_\xi, \nabla_\mathfrak{f}, \nabla_\mathfrak{b}\}$ indicated by $*$ are required (cf. [5](6.7)). We say an abstract consistency class Γ_Σ is saturated if ∇_{sat} holds.

Using ∇_c (atomic consistency) and the fact that there are infinitely many parameters at each type, we can show every abstract consistency class satisfies non-atomic consistency. That is, for every abstract consistency class Γ_Σ, $\mathbf{A} \in cwff_o(\Sigma)$ and $\Phi \in \Gamma_\Sigma$, we have either $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$ (cf. [5](6.10)).

In [5](6.32) we show that sufficiently Σ-pure sets in saturated abstract consistency classes extend to saturated Hintikka sets. (A set of sentences Φ is sufficiently Σ-pure if for each type α there is a set $\mathcal{P}_\alpha$ of parameters of type α with cardinality $\aleph_s$ and such that no parameter in $\mathcal{P}$ occurs in a sentence in Φ.)

In the Model Existence Theorem for Saturated Sets [5](6.33) we show that these saturated Hintikka sets can be used to construct models $\mathcal{M}$ which are members of the corresponding model classes $\mathfrak{M}_*$. Then we conclude (cf. [5](6.34)):

Model Existence Theorem for Saturated Abstract Consistency Classes: *For all $* \in \mathfrak{B}$, if Γ_Σ is a saturated abstract consistency class in $\mathfrak{Acc}_*$ and $\Phi \in \Gamma_\Sigma$ is a sufficiently Σ-pure set of sentences, then there exists a model $\mathcal{M} \in \mathfrak{M}_*$ that satisfies Φ. Furthermore, each domain of $\mathcal{M}$ has cardinality at most $\aleph_s$.*

In [5] we apply the abstract consistency method to analyze completeness for different natural deduction calculi. Unfortunately, the saturation condition is very difficult to prove for machine-oriented calculi (indeed as we will see in Section 3 it is equivalent to cut elimination), so Theorem [5](6.34) cannot be easily used for this purpose directly.

In Section 6 we therefore motivate and present a set of extra conditions for $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$ we call **acceptability** conditions. The new conditions are sufficient to prove model existence.

Basic Rules

$$\frac{\mathbf{A} \text{ atomic (and } \beta\text{-normal)}}{\Delta * \mathbf{A} * \neg\mathbf{A}}\,\mathcal{G}(init) \qquad \frac{\Delta * \mathbf{A}}{\Delta * \neg\neg\mathbf{A}}\,\mathcal{G}(\neg)$$

$$\frac{\Delta * \neg\mathbf{A} \quad \Delta * \neg\mathbf{B}}{\Delta * \neg(\mathbf{A} \vee \mathbf{B})}\,\mathcal{G}(\vee_-) \qquad \frac{\Delta * \mathbf{A} * \mathbf{B}}{\Delta * (\mathbf{A} \vee \mathbf{B})}\,\mathcal{G}(\vee_+)$$

$$\frac{\Delta * \neg\,(\mathbf{A}\mathbf{C})\downarrow_\beta \quad \mathbf{C} \in \mathit{cwff}_\alpha(\Sigma)}{\Delta * \neg\Pi^\alpha\mathbf{A}}\,\mathcal{G}(\Pi^{\mathbf{C}}_-) \qquad \frac{\Delta * (\mathbf{A}c)\downarrow_\beta \quad c_\alpha \in \Sigma \text{ new}}{\Delta * \Pi^\alpha\mathbf{A}}\,\mathcal{G}(\Pi^c_+)$$

Inversion Rule

$$\frac{\Delta * \neg\neg\mathbf{A}}{\Delta * \mathbf{A}}\,\mathcal{G}(Inv^\neg)$$

Weakening and Cut Rules

$$\frac{\Delta}{\Delta \cup \Delta'}\,\mathcal{G}(weak) \qquad \frac{\Delta * \mathbf{C} \quad \Delta * \neg\mathbf{C}}{\Delta}\,\mathcal{G}(cut)$$

Fig. 1. Sequent Calculus Rules

3 Sequent Calculi, Cut and Saturation

We will now study cut-elimination and cut-simulation with respect to (one-sided) sequent calculi.

Sequent Calculi $\mathcal{G}$. We consider a sequent to be a finite set Δ of β-normal sentences from $\mathit{cwff}_o(\Sigma)$. A sequent calculus $\mathcal{G}$ provides an inductive definition for when $\Vdash_{\mathcal{G}} \Delta$ holds. We say a sequent calculus rule

$$\frac{\Delta_1 \quad \cdots \quad \Delta_n}{\Delta}\,r$$

is **admissible** in $\mathcal{G}$ if $\Vdash_{\mathcal{G}} \Delta$ holds whenever $\Vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$. For any natural number $k \geq 0$, we call an admissible rule r **k-admissible** if any instance of r can be replaced by a derivation with at most k additional proof steps. Given a sequent Δ, a model $\mathcal{M}$, and a class $\mathfrak{M}$ of models, we say Δ is *valid for* $\mathcal{M}$ (or *valid for* $\mathfrak{M}$), if $\mathcal{M} \models \mathbf{D}$ for some $\mathbf{D} \in \Delta$ (or Δ is valid for every $\mathcal{M} \in \mathfrak{M}$). As for sets in abstract consistency classes, we use the notation $\Delta * \mathbf{A}$ to denote the set $\Delta \cup \{\mathbf{A}\}$ (which is simply Δ if $\mathbf{A} \in \Delta$). Figure 1 introduces several sequent calculus rules. Some of these rules will be used to define sequent calculi, while others will be shown admissible (or even k-admissible).

Abstract Consistency Classes for Sequent Calculi. For any sequent calculus $\mathcal{G}$ we can define a class $\mathit{\Gamma}^{\mathcal{G}}_\Sigma$ of sets of sentences. Under certain assumptions, $\mathit{\Gamma}^{\mathcal{G}}_\Sigma$ is an abstract consistency class. First we adopt the notation $\neg\Phi$ and $\Phi\downarrow_\beta$ for the sets $\{\neg\mathbf{A} | \mathbf{A} \in \Phi\}$ and $\{\mathbf{A}\downarrow_\beta | \mathbf{A} \in \Phi\}$, resp., where $\Phi \subseteq \mathit{cwff}_o(\Sigma)$. Furthermore, we assume this use of $\neg$ binds more strongly than $\cup$ or $*$, so that $\neg\Phi \cup \Delta$ means $(\neg\Phi) \cup \Delta$ and $\neg\Phi * \mathbf{A}$ means $(\neg\Phi) * \mathbf{A}$.

Definition 1 *Let $\mathcal{G}$ be a sequent calculus. We define $\Gamma^{\mathcal{G}}_{\Sigma}$ to be the class of all finite $\Phi \subset cwff_o(\Sigma)$ such that $\nvdash_{\mathcal{G}} \neg \Phi\downarrow_\beta$ does not hold.*

In a straightforward manner, one can prove the following results (see [7]).

Lemma 2 *Let $\mathcal{G}$ be a sequent calculus such that $\mathcal{G}(Inv^{\neg})$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma^{\mathcal{G}}_{\Sigma}$, then $\nvdash_{\mathcal{G}} \neg \Phi\downarrow_\beta \cup \Delta\downarrow_\beta$ holds.*

Theorem 3 *Let $\mathcal{G}$ be a sequent calculus. If the rules $\mathcal{G}(Inv^{\neg})$, $\mathcal{G}(\neg)$, $\mathcal{G}(weak)$, $\mathcal{G}(init)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi^{\mathbf{C}}_-)$ and $\mathcal{G}(\Pi^c_+)$ are admissible in $\mathcal{G}$, then $\Gamma^{\mathcal{G}}_{\Sigma} \in \mathfrak{Acc}_\beta$.*

We can furthermore show the following relationship between saturation and cut (see [7]).

Theorem 4 *Let $\mathcal{G}$ be a sequent calculus.*

1. *If $\mathcal{G}(cut)$ is admissible in $\mathcal{G}$, then $\Gamma^{\mathcal{G}}_{\Sigma}$ is saturated.*
2. *If $\mathcal{G}(\neg)$ and $\mathcal{G}(Inv^{\neg})$ are admissible in $\mathcal{G}$ and $\Gamma^{\mathcal{G}}_{\Sigma}$ is saturated, then $\mathcal{G}(cut)$ is admissible in $\mathcal{G}$.*

Since saturation is equivalent to admissibility of cut, we need weaker conditions than saturation. A natural condition to consider is the existence of saturated extensions.

Definition 5 (Saturated Extension) *Let $* \in \mathfrak{B}$ and $\Gamma_\Sigma, \Gamma'_\Sigma \in \mathfrak{Acc}_*$ be abstract consistency classes. We say Γ'_Σ is an **extension** of Γ_Σ if $\Phi \in \Gamma'_\Sigma$ for every sufficiently Σ-pure $\Phi \in \Gamma_\Sigma$. We say Γ'_Σ is a **saturated extension** of Γ_Σ if Γ'_Σ is saturated and an extension of Γ_Σ.*

There exist abstract consistency classes Γ in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$ which have no saturated extension.

Example 6 *Let $a_o, b_o, q_{o\to o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $\equiv_\beta$ and then taking the power set of Φ'. It is easy to check that this Γ_Σ is in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$. Then $\Phi \in \Gamma'_\Sigma$ since Φ is finite (hence sufficiently pure). By saturation, $\Phi * (a \doteq^o b) \in \Gamma'_\Sigma$ or $\Phi * \neg(a \doteq^o b) \in \Gamma'_\Sigma$. In the first case, applying $\nabla_\forall$ with the constant q, $\nabla_\vee$ and ∇_c contradicts $(qa), \neg(qb) \in \Phi$. In the second case, $\nabla_\mathfrak{b}$ and ∇_c contradict $a, b \in \Phi$.*

Existence of any saturated extension of a sound sequent calculus $\mathcal{G}$ implies admissibility of cut. The proof uses the model existence theorem for saturated abstract consistency classes (cf. [5](6.34)). The full proof is in [7].

Theorem 7 *Let $\mathcal{G}$ be a sequent calculus which is sound for $\mathfrak{M}_*$. If $\Gamma^{\mathcal{G}}_{\Sigma}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(cut)$ is admissible in $\mathcal{G}$.*

Sequent Calculus $\mathcal{G}_\beta$. We now study a particular sequent calculus $\mathcal{G}_\beta$ defined by the rules $\mathcal{G}(init)$, $\mathcal{G}(\neg)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi_-^{\mathbf{C}})$ and $\mathcal{G}(\Pi_+^c)$ (cf. Figure 1). It is easy to show that $\mathcal{G}_\beta$ is sound for the eight model classes and in particular for class $\mathfrak{M}_\beta$.

The reader may easily prove the following Lemma.

Lemma 8 *Let $\mathbf{A} \in cwff_o(\Sigma)$ be an atom, $\mathbf{B} \in cwff_\alpha(\Sigma)$, and Δ be a sequent. In $\mathcal{G}_\beta$*

1. *$\Delta * \mathbf{A} \Leftrightarrow \mathbf{A} := \Delta * \neg(\neg(\neg\mathbf{A} \vee \mathbf{A}) \vee \neg(\neg\mathbf{A} \vee \mathbf{A}))$ is derivable in 7 steps and*
2. *$\Delta * \mathbf{B} \doteq^\alpha \mathbf{B} := \Delta * \Pi^\alpha(\lambda P_{\alpha\to o}.\neg(P\mathbf{B}) \vee (P\mathbf{B}))$ is derivable in 3 steps.*

The proof of the next Lemma is by induction on derivations and is given in [7].

Lemma 9 *The rules $\mathcal{G}(Inv^\neg)$ and $\mathcal{G}(weak)$ are 0-admissible in $\mathcal{G}_\beta$.*

Theorem 10 *The sequent calculus $\mathcal{G}_\beta$ is complete for the model class $\mathfrak{M}_\beta$ and the rule $\mathcal{G}(cut)$ is admissible.*

Proof: By Theorem 3 and Lemma 9, $I_\Sigma^{\mathcal{G}_\beta} \in \mathfrak{Acc}_\beta$. Suppose $\nvdash_{\mathcal{G}_\beta} \Delta$ does not hold. Then $\neg\Delta \in \mathfrak{Acc}_\beta$ by Lemma 2. By the model existence theorem for $\mathfrak{Acc}_\beta$ (cf. [6](8.1)) there exists a model for $\neg\Delta$ in $\mathfrak{M}_\beta$. This gives completeness of $\mathcal{G}_\beta$. We can use completeness to conclude cut is admissible in $\mathcal{G}_\beta$. □

Andrews proves admissibility of cut for a sequent calculus similar to $\mathcal{G}_\beta$ in [1]. The proof in [1] contains the essential ingredients for showing completeness.

We will now show that $\mathcal{G}(cut)$ actually becomes k-admissible in $\mathcal{G}_\beta$ if certain formulae are available in the sequent Δ we wish to prove.

4 Cut-Simulation

Cut-Strong Formulae and Sequents. k-cut-strong formulae can be used to effectively simulate cut. Effectively means that the elimination of each application of a cut-rule introduces maximally k additional proof steps, where k is constant.

Definition 11 *Given a formula $\mathbf{A} \in cwff_o(\Sigma)$, and an arbitrary but fixed number $k > 0$. We call formula $\mathbf{A}$ **k-cut-strong** for $\mathcal{G}$ (or simply **cut-strong**) if the cut rule variant*

$$\frac{\Delta * \mathbf{C} \quad \Delta * \neg\mathbf{C}}{\Delta * \neg\mathbf{A}}\,\mathcal{G}(cut^{\mathbf{A}})$$

is k-admissible in $\mathcal{G}$.

Our examples below illustrate that cut-strength of a formula usually only weakly depends on calculus $\mathcal{G}$: it only presumes standard ingredients such as β-normalization, weakening, and rules for the logical connectives.

We present some simple examples of cut-strong formulae for our sequent calculus $\mathcal{G}_\beta$. A corresponding phenomenon is observable in other higher-order calculi, for instance, for the calculi presented in [1, 4, 8, 11].

Example 12 *Formula* $\forall P_o.P := \Pi^o(\lambda P_o.P)$ *is* 3*-cut-strong in* $\mathcal{G}_\beta$*. This is justified by the following derivation which actually shows that rule* $\mathcal{G}(cut^{\mathbf{A}})$ *for this specific choice of* $\mathbf{A}$ *is derivable in* $\mathcal{G}_\beta$ *by maximally* 3 *additional proof steps. The only interesting proof step is the instantiation of* P *with formula* $\mathbf{D} := \neg\mathbf{C} \vee \mathbf{C}$ *in rule* $\mathcal{G}(\Pi^{\mathbf{D}}_{-})$*. (Note that* $\mathbf{C}$ *must be* β*-normal; sequents such as* $\Delta{*}\mathbf{C}$ *by definition contain only* β*-normal formulae.)*

$$\dfrac{\dfrac{\dfrac{\Delta * \mathbf{C}}{\Delta * \neg\neg\mathbf{C}}\,\mathcal{G}(\neg) \qquad \Delta * \neg\mathbf{C}}{\Delta * \neg(\neg\mathbf{C} \vee \mathbf{C})}\,\mathcal{G}(\vee_{-})}{\Delta * \neg\Pi^o(\lambda P_o.P)}\,\mathcal{G}(\Pi^{\mathbf{D}}_{-})$$

Clearly, $\forall P_o.P$ *is not a very interesting cut-strong formula since it implies falsehood, i.e. inconsistency.*

Example 13 *The formula* $\forall P_o.P \Rightarrow P := \Pi^o(\lambda P_o.\neg P \vee P)$ *is* 3*-cut-strong in* $\mathcal{G}_\beta$*. This is an example of a tautologous cut-strong formula. Now* P *is simply instantiated with* $\mathbf{D} := \mathbf{C}$ *in rule* $\mathcal{G}(\Pi^{\mathbf{D}}_{-})$*. Except for this first step the derivation is identical to the one for Example 12.*

Example 14 *Leibniz equations* $\mathbf{M} \doteq^\alpha \mathbf{N} := \Pi^\alpha(\lambda P.\neg P\mathbf{M} \vee P\mathbf{N})$ *(for arbitrary formulae* $\mathbf{M}, \mathbf{N} \in cwff_\alpha(\Sigma)$ *and types* $\alpha \in \mathcal{T}$*) are* 3*-cut-strong in* $\mathcal{G}_\beta$*. This includes the special cases* $\mathbf{M} \doteq^\alpha \mathbf{M}$*. Now* P *is instantiated with* $\mathbf{D} := \lambda X_\alpha.\mathbf{C}$ *in rule* $\mathcal{G}(\Pi^{\mathbf{D}}_{-})$*. Except for this first step the derivation is identical to the one for Example 12.*

Example 15 *The original formulation of higher-order logic (cf. [12]) contained comprehension axioms of the form* $\mathcal{C} := \exists P_{\alpha^1 \to \dots \to \alpha^n \to o} \forall \overline{X^n}.P\overline{X^n} \Leftrightarrow \mathbf{B}_o.$ *where* $\mathbf{B}_o \in wff_o(\Sigma)$ *is arbitrary with* $P \notin free(\mathbf{B})$*. Church eliminated the need for such axioms by formulating higher-order logic using typed* λ*-calculus. We will now show that the instance* $\mathcal{C}^I := \exists P_{\iota \to o}.\forall X_\iota.PX \Leftrightarrow X \doteq^\iota X$ *is* 16*-cut-strong in* $\mathcal{G}_\beta$ *(note that* $\mathcal{G}(weak)$ *is* 0*-admissible). This motivates building-in comprehension principles instead of treating comprehension axiomatically.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\textit{3 steps; see Lemma 8}\\ \vdots \\ \Delta * \neg(pa \Rightarrow a \doteq^\iota a) * a \doteq^\iota a\end{array}}{\Delta * \neg(pa \Rightarrow a \doteq^\iota a) * \neg\neg(a \doteq^\iota a)}\,\mathcal{G}(\neg) \qquad \mathcal{D}}{\Delta * \neg(pa \Rightarrow a \doteq^\iota a) * \neg(\neg(a \doteq^\iota a) \vee pa)}\,\mathcal{G}(\vee_{-})}{\Delta * \neg(pa \Rightarrow a \doteq^\iota a) \vee \neg(a \doteq^\iota a \Rightarrow pa)}\,\mathcal{G}(\vee_{+})}{\Delta * \neg\neg(\neg(pa \Rightarrow a \doteq^\iota a) \vee \neg(a \doteq^\iota a \Rightarrow pa))}\,\mathcal{G}(\neg)}{\Delta * \neg\Pi^\iota(\lambda X_\iota.pX \Leftrightarrow X \doteq^\iota X)}\,\mathcal{G}(\Pi^{a_\iota}_{-})}{\Delta * \Pi^{\iota\to o}(\lambda P^{\iota \to o}.\neg\Pi^\iota(\lambda X_\iota.pX \Leftrightarrow X \doteq^\iota X))}\,\mathcal{G}(\Pi^{p_{\iota\to o}}_{+})}{\Delta * \mathcal{C}^I}\,\mathcal{G}(\neg)$$

Derivation $\mathcal{D}$ is:

$$\dfrac{\dfrac{\dfrac{}{\Delta * pa * \neg pa}\,\mathcal{G}(init)}{\Delta * \neg\neg pa * \neg pa}\,\mathcal{G}(\neg) \qquad \dfrac{\Delta * \mathbf{C} \quad \Delta * \neg\mathbf{C}}{\vdots \text{ 3 steps; see Example 14}} \quad \dfrac{\Delta * \neg(a \doteq^{o} a)}{\Delta * \neg(a \doteq^{\iota} a) * \neg pa}\,\mathcal{G}(weak)}{\Delta * \neg(\neg pa \vee a \doteq^{\iota} a) * \neg pa}\,\mathcal{G}(\vee_{-})$$

As we will show later, many prominent axioms for higher-order logic also belong to the class of cut-strong formulae.

Next we define cut-strong sequents.

Definition 16 *A sequent Δ is called k-cut-strong (or simply **cut-strong**) if there exists a a k-cut-strong formula $\mathbf{A} \in cwff_o(\Sigma)$ such that $\neg\mathbf{A} \in \Delta$.*

Cut-Simulation. The cut-simulation theorem is a main result of this paper. It says that cut-strong sequents support an effective simulation (and thus elimination) of cut in $\mathcal{G}_\beta$. Effective means that the size of cut-free derivation grows only linearly for the number of cut rule applications to be eliminated.

We first fix the following calculi: Calculus $\mathcal{G}_\beta^{cut}$ extends $\mathcal{G}_\beta$ by the rule $\mathcal{G}(cut)$ and calculus $\mathcal{G}_\beta^{cut^{\mathbf{A}}}$ extends $\mathcal{G}_\beta$ by the rule $\mathcal{G}(cut^{\mathbf{A}})$ for some arbitrary but fixed cut-strong formula $\mathbf{A}$.

Theorem 17 *Let Δ be a k-cut-strong sequent such that $\neg\mathbf{A} \in \Delta$ for some k-cut-strong formula $\mathbf{A}$. For each derivation $\mathcal{D}$: $\vdash_{\mathcal{G}_\beta^{cut}} \Delta$ with d proof steps there exists an alternative derivation $\mathcal{D}'$: $\vdash_{\mathcal{G}_\beta^{cut^{\mathbf{A}}}} \Delta$ with d proof steps.*

Proof: Note that the rules $\mathcal{G}(cut)$ and $\mathcal{G}(cut^{\mathbf{A}})$ coincide whenever $\neg\mathbf{A} \in \Delta$. Intuitively, we can replace each occurrence of $\mathcal{G}(cut)$ in $\mathcal{D}$ by $\mathcal{G}(cut^{\mathbf{A}})$ in order to obtain a $\mathcal{D}'$ of same size. Technically, in the induction proof one must weaken to ensure $\neg\mathbf{A}$ stays in the sequent and carry out a parameter renaming to make sure the eigenvariable condition is satisfied. □

Theorem 18 *Let Δ be a k-cut-strong sequent such that $\neg\mathbf{A} \in \Delta$ for some k-cut-strong formula $\mathbf{A}$. For each derivation $\mathcal{D}$: $\vdash_{\mathcal{G}_\beta^{cut^{\mathbf{A}}}} \Delta$ with d proof steps and with n applications of rule $\mathcal{G}(cut)$ there exists an alternative derivation $\mathcal{D}'$: $\vdash_{\mathcal{G}_\beta} \Delta$ with maximally $d + nk$ proof steps.*

Proof: $\mathbf{A}$ is k-cut-strong so by definition $\mathcal{G}(cut^{\mathbf{A}})$ is k-admissible in $\mathcal{G}_\beta$. This means that $\mathcal{G}(cut^{\mathbf{A}})$ can be eliminated in $\mathcal{D}$ and each single elimination of $\mathcal{G}(cut^{\mathbf{A}})$ introduces maximally k new proof steps. Now the assertion can be easily obtained by a simple induction over n. □

Corollary 19 *Let Δ be a k-cut-strong sequent. For each derivation $\mathcal{D}$: $\vdash_{\mathcal{G}_\beta^{cut}} \Delta$ with d proof steps and n applications of rule $\mathcal{G}(cut)$ there exists an alternative cut-free derivation $\mathcal{D}'$: $\vdash_{\mathcal{G}_\beta} \Delta$ with maximally $d + nk$ proof steps.*

5 The Extensionality Axioms are Cut-Strong

We have shown comprehension axioms can be cut-strong (cf. Example 15). Further prominent examples of cut-strong formulae are the Boolean and functional extensionality axioms. The Boolean extensionality axiom (abbreviated $\mathcal{B}_o$ in the remainder) is

$$\forall A_o.\forall B_o.(A \Leftrightarrow B) \Rightarrow A \doteq^o B$$

The infinitely many functional extensionality axioms (abbreviated $\mathcal{F}_{\alpha\beta}$) are parameterized over $\alpha, \beta \in \mathcal{T}$.

$$\forall F_{\alpha\to\beta}.\forall G_{\alpha\to\beta}.(\forall X_\alpha.FX \doteq^\beta GX) \Rightarrow F \doteq^{\alpha\to\beta} G$$

These axioms usually have to be added to higher-order calculi to reach Henkin completeness, i.e. completeness with respect to model class $\mathfrak{M}_{\beta\mathfrak{fb}}$. For example, Huet's constrained resolution approach as presented in [11] is not Henkin complete without adding extensionality axioms. For instance, the need for adding Boolean extensionality is actually illustrated by the set of unit literals $\Phi := \{a, b, (qa), \neg(qb)\}$ from Example 6. As the reader may easily check, this clause set Φ, which is inconsistent for Henkin semantics, cannot be proven by Huet's system without, e.g, adding the Boolean extensionality axiom. By relying on results in [1], Huet essentially shows completeness with respect to model class $\mathfrak{M}_\beta$ as opposed to Henkin semantics.

We will now investigate whether adding the extensionality axioms to a machine-oriented calculus in order to obtain Henkin completeness is a suitable option.

Theorem 20 *The Boolean extensionality axiom $\mathcal{B}_o$ is a 14-cut-strong formula in $\mathcal{G}_\beta$.*

Proof: The following derivation justifies this theorem (a_o is a parameter).

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\text{7 steps; see Lemma 8}\\ \vdots\\ \Delta * a \Leftrightarrow a\end{array}}{\Delta * \neg\neg(a \Leftrightarrow a)}\,\mathcal{G}(\neg) \qquad \begin{array}{c}\Delta * \mathbf{C}\\ \vdots\\ \Delta * \neg(a \doteq^o a)\end{array}\,\begin{array}{l}\Delta * \neg\mathbf{C}\\ \vdots\ \text{3 steps; see Example 14}\end{array}}{\Delta * \neg(\neg(a \Leftrightarrow a) \vee a \doteq^o a)}\,\mathcal{G}(\vee_-)}{\Delta * \neg\mathcal{B}_o}\,2 \times \mathcal{G}(\Pi^a_-)$$

□

Theorem 21 *The functional extensionality axioms $\mathcal{F}_{\alpha\beta}$ are 11-cut-strong formulae in $\mathcal{G}_\beta$.*

Proof: The following derivation justifies this theorem ($f_{\alpha\to\beta}$ is a parameter).

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\text{3 steps; see Lemma 8}\\ \vdots\\ \Delta * fa \doteq^\beta fa\end{array}}{\Delta * (\forall X_\alpha.fX \doteq^\beta fX)}\,\mathcal{G}(\Pi^{a_\alpha}_+) \qquad \begin{array}{c}\Delta * \mathbf{C}\\ \vdots\\ \Delta * \neg(f \doteq^{\alpha\to\beta} f)\end{array}\,\begin{array}{l}\Delta * \neg\mathbf{C}\\ \vdots\ \text{3 steps; see Example 14}\end{array}}{\Delta * \neg\neg\forall X_\alpha.fX \doteq^\beta fX}\,\mathcal{G}(\neg)}{\Delta * \neg(\neg(\forall X_\alpha.fX \doteq^\beta fX) \vee f \doteq^{\alpha\to\beta} f)}\,\mathcal{G}(\vee_-)}{\Delta * \neg\mathcal{F}_{\alpha\beta}}\,2 \times \mathcal{G}(\Pi^f_-)$$

□

$$\frac{\Delta * \neg \mathcal{F}_{\alpha\beta} \quad \alpha \to \beta \in \mathcal{T}}{\Delta}\,\mathcal{G}(\mathcal{F}_{\alpha\beta}) \qquad \frac{\Delta * \neg \mathcal{B}_o}{\Delta}\,\mathcal{G}(\mathcal{B})$$

Fig. 2. Axiomatic Extensionality Rules

In [4] and [8] we have already argued that the extensionality principles should not be treated axiomatically in machine-oriented higher-order calculi and there we have developed resolution and sequent calculi in which these principles are built-in. Here we have now developed a strong theoretical justification for this work: Theorems 20, 21 and 19 tell us that adding the extensionality principles $\mathcal{B}_o$ and $\mathcal{F}_{\alpha\beta}$ as axioms to a calculus is like adding a cut rule.

In Figure 2 we show rules that add Boolean and functional extensionality in an axiomatic manner to $\mathcal{G}_\beta$. More precisely we add rules $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ and $\mathcal{G}(\mathcal{B})$ allowing to introduce the axioms for any sequent Δ; this way we address the problem of the infinitely many possible instantiations of the type-schematic functional extensional axiom $\mathcal{F}_{\alpha\beta}$. Calculus $\mathcal{G}_\beta$ enriched by the new rules $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ and $\mathcal{G}(\mathcal{B})$ is called $\mathcal{G}^E_\beta$. Soundness of the the new rules is easy to verify: In [5](4.3) we show that $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ and $\mathcal{G}(\mathcal{B})$ are valid for Henkin models.

Replacing the Extensionality Axioms. In Figure 3 we define alternative extensionality rules which correspond to those developed for resolution and sequent calculi in [4] and [8]. Calculus $\mathcal{G}_\beta$ enriched by $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\mathfrak{b})$ is called $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$. Soundness of $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\mathfrak{b})$ for Henkin semantics is again easy to show.

Our aim is to develop a machine-oriented sequent calculus for automating Henkin complete proof search. We argue that for this purpose $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\mathfrak{b})$ are more suitable rules than $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ and $\mathcal{G}(\mathcal{B})$.

Our next step now is to show Henkin completeness for $\mathcal{G}^E_\beta$. This will be relatively easy since we can employ cut-simulation. Then we analyze whether calculus $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$ has the same deductive power as $\mathcal{G}^E_\beta$.

First we extend Theorem 3. The proof is given in [7].

Theorem 22 *Let $\mathcal{G}$ be a sequent calculus such that $\mathcal{G}(Inv^\neg)$ and $\mathcal{G}(\neg)$ are admissible.*

1. *If $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\Pi^c_+)$ are admissible, then $\Gamma^{\mathcal{G}}_\Sigma$ satisfies $\nabla_\mathfrak{f}$.*
2. *If $\mathcal{G}(\mathfrak{b})$ is admissible, then $\Gamma^{\mathcal{G}}_\Sigma$ satisfies $\nabla_\mathfrak{b}$.*

Theorem 23 *The sequent calculus $\mathcal{G}^E_\beta$ is Henkin complete and the rule $\mathcal{G}(cut)$ is* 12*-admissible.*

Proof: $\mathcal{G}(cut)$ can be effectively simulated and hence eliminated in $\mathcal{G}^E_\beta$ by combining rule $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ with the 11-step derivation presented in the proof of Theorem 21.

Let $\Gamma^{\mathcal{G}^E_\beta}_\Sigma$ be defined as in Definition 1. We prove Henkin completeness of $\mathcal{G}^E_\beta$ by showing that the class $\Gamma^{\mathcal{G}^E_\beta}_\Sigma$ is a saturated abstract consistency class in

$$\frac{\Delta * (\forall X_\alpha.\mathbf{A}X \doteq^\beta \mathbf{B}X)\big\downarrow_\beta}{\Delta * (\mathbf{A} \doteq^{\alpha\to\beta} \mathbf{B})}\,\mathcal{G}(\mathfrak{f}) \qquad \frac{\Delta * \neg\mathbf{A} * \mathbf{B} \quad \Delta * \neg\mathbf{B} * \mathbf{A}}{\Delta * (\mathbf{A} \doteq^{o} \mathbf{B})}\,\mathcal{G}(\mathfrak{b})$$

Fig. 3. Proper Extensionality Rules

$\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$. We here only analyze the crucial conditions $\nabla_\mathfrak{b}$, $\nabla_\mathfrak{f}$ and ∇_{sat}. For the other conditions we refer to Theorem 3. Note that 0-admissibility of $\mathcal{G}(Inv^\neg)$ and $\mathcal{G}(weak)$ can be shown for $\mathcal{G}^E_\beta$ by a suitable induction on derivations as in Lemma 9.

$\nabla_\mathfrak{f}$ $\mathcal{G}(\Pi^c_+)$ is a rule of $\mathcal{G}^E_\beta$ and thus admissible. According to Theorem 22 it is thus sufficient to ensure admissibility of rule $\mathcal{G}(\mathfrak{f})$ to show $\nabla_\mathfrak{f}$. This is justified by the following derivation where $\mathbf{N} := \mathbf{A} \doteq^{\alpha\to\beta} \mathbf{B}$ and $\mathbf{M} := (\forall X_\alpha.\mathbf{A}X \doteq^\beta \mathbf{B}X)\big\downarrow_\beta$ (for β-normal $\mathbf{A}, \mathbf{B}$).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Delta * (\forall X_\alpha.\mathbf{A}X \doteq^\beta \mathbf{B}X)\big\downarrow_\beta}{\Delta * \mathbf{N} * \mathbf{M}}\,\mathcal{G}(weak) \qquad \begin{array}{c}\text{derivable}\\ \vdots\\ \Delta * \mathbf{N} * \neg\mathbf{N}\end{array}}{\Delta * \mathbf{N} * \neg\neg\mathbf{M}}\,\mathcal{G}(\neg)}{\Delta * \mathbf{N} * \neg(\neg\mathbf{M} \vee \mathbf{N})}\,\mathcal{G}(\vee_-)}{\Delta * \mathbf{N} * \neg\mathcal{F}_{\alpha\beta}}\,\mathcal{G}(\Pi^{\mathbf{A}}_-), \mathcal{G}(\Pi^{\mathbf{B}}_-)}{\Delta * \mathbf{A} \doteq^{\alpha\to\beta} \mathbf{B}}\,\mathcal{G}(\mathcal{F}_{\alpha\beta})$$

$\nabla_\mathfrak{b}$ With a similar derivation using $\mathcal{G}(\mathcal{B})$ we can show that $\mathcal{G}(\mathfrak{b})$ is admissible. We conclude $\nabla_\mathfrak{b}$ by Theorem 22.

∇_{sat} Since $\mathcal{G}(cut)$ is admissible we get saturation by Theorem 4. □

Does $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$ have the same deductive strength as $\mathcal{G}^E_\beta$? I.e., is $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$ Henkin complete? We show this is not yet the case.

Theorem 24 *The sequent calculus $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$ is not complete for Henkin semantics.*

We illustrate the problem by a counterexample.

Example 25 *Consider the sequent $\Delta := \{\neg a, \neg b, \neg(qa), (qb)\}$ where a_o, b_o, $q_{o\to o} \in \Sigma$ are parameters. For any $\mathcal{M} \equiv (\mathcal{D}, @, \mathcal{E}, \upsilon) \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$, either $\upsilon(\mathcal{E}(a)) \equiv \mathtt{F}$, $\upsilon(\mathcal{E}(b)) \equiv \mathtt{F}$ or $\mathcal{E}(a) \equiv \mathcal{E}(b)$ by property $\mathfrak{b}$. Hence sequent Δ is valid for every $\mathcal{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$. However, $\nvdash_{\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}} \Delta$ does not hold. By inspection, Δ cannot be the conclusion of any rule.*

In order to reach Henkin completeness and to show cut-elimination we thus need to add further rules. Our example motivates the two rules presented in Figure 4. $\mathcal{G}(Init^{\doteq})$ introduces Leibniz equations such as $qa \doteq^o qb$ as is needed in our example and $\mathcal{G}(d)$ realizes the required decomposition into $a \doteq^o b$.

$$\frac{\Delta * (\mathbf{A} \doteq^o \mathbf{B}) \quad (\dagger)}{\Delta * \neg\mathbf{A} * \mathbf{B}}\,\mathcal{G}(Init^{\doteq}) \qquad \frac{\Delta * (\mathbf{A}^1 \doteq^{\alpha_1} \mathbf{B}^1) \cdots \Delta * (\mathbf{A}^n \doteq^{\alpha_n} \mathbf{B}^n) \quad (\ddagger)}{\Delta * (h\overline{\mathbf{A}^n} \doteq^{\beta} h\overline{\mathbf{B}^n})}\,\mathcal{G}(d)$$

($\dagger$) $\mathbf{A}$,$\mathbf{B}$ atomic

($\ddagger$) $n \geq 1, \beta \in \{o, \iota\}, h_{\overline{\alpha^n}\to\beta} \in \Sigma$ parameter

Fig. 4. Additional Rules $\mathcal{G}(Init^{\doteq})$ and $\mathcal{G}(d)$

We thus extend sequent calculus $\mathcal{G}^-_{\beta\mathfrak{f}\mathfrak{b}}$ to $\mathcal{G}_{\beta\mathfrak{f}\mathfrak{b}}$ by adding the decomposition rule $\mathcal{G}(d)$ and the rule $\mathcal{G}(Init^{\doteq})$ which generally checks if two atomic sentences of opposite polarity are provably equal (as opposed to syntactically equal).

Is $\mathcal{G}_{\beta\mathfrak{f}\mathfrak{b}}$ complete for Henkin semantics? We will show in the next Section that this indeed holds (cf. Theorem 28).

With $\mathcal{G}^E$ and $\mathcal{G}_{\beta\mathfrak{f}\mathfrak{b}}$ we have thus developed two Henkin complete calculi and both calculi are cut-free. However, as our exploration shows "cut-freeness" is not a well-chosen criterion to differentiate between their suitability for proof search automation: $\mathcal{G}^E$ inherently supports effective cut-simulation and thus cut-freeness is meaningless.

The criterion we propose for the analysis of calculi in impredicative logics is "freeness of effective cut-simulation".

Other Rules for Other Model Classes. In [6] we developed respective complete and cut-free sequent calculi not only for Henkin semantics but for five of the eight model classes. In particular, no additional rules are required for the β, $\beta\eta$ and $\beta\xi$ case. Meanwhile, the $\beta\mathfrak{f}$ case requires additional rules allowing η-conversion. The limited space does not allow us to present and analyze these cases here.

6 Acceptability Conditions

We now turn our attention again to the existence of saturated extension of abstract consistency classes.

As illustrated by the Example 6, we need some extra abstract consistency properties to ensure the existence of saturated extensions. We call these extra properties **acceptability conditions**. They actually closely correspond to additional rules $\mathcal{G}(Init^{\doteq})$ and $\mathcal{G}(d)$.

Definition 26 (Acceptability Conditions) *Let Γ_Σ be an abstract consistency class in $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$. We define the following properties:*

∇_m *If $\mathbf{A}, \mathbf{B} \in cwff_o(\Sigma)$ are atomic and $\mathbf{A}, \neg\mathbf{B} \in \Phi$, then $\Phi * \neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Gamma_\Sigma$.*

∇_d *If $\neg(h\overline{\mathbf{A}^n} \doteq^{\beta} h\overline{\mathbf{B}^n}) \in \Phi$ for some types α_i where $\beta \in \{o, \iota\}$ and $h_{\overline{\alpha^n}\to\beta} \in \Sigma$ is a parameter, then there is an i $(1 \leq i \leq n)$ such that $\Phi * \neg(\mathbf{A}^i \doteq^{\alpha^i} \mathbf{B}^i) \in \Gamma_\Sigma$.*

We now replace the strong saturation condition used in [5] by these acceptability conditions.

Definition 27 (Acceptable Classes) *An abstract consistency class $\Gamma_\Sigma \in \mathfrak{Acc}_{\beta\mathfrak{fb}}$ is called **acceptable** in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$ if it satisfies the conditions ∇_m and ∇_d.*

One can show a model existence theorem for acceptable abstract consistency classes in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$ (cf. [6](8.1)). From this model existence theorem, one can conclude $\mathcal{G}_{\beta\mathfrak{fb}}$ is complete for $\mathfrak{M}_{\beta\mathfrak{fb}}$ (hence for Henkin models) and that cut is admissible in $\mathcal{G}_{\beta\mathfrak{fb}}$.

Theorem 28 *The sequent calculus $\mathcal{G}_{\beta\mathfrak{fb}}$ is complete for Henkin semantics and the rule $\mathcal{G}(cut)$ is admissible.*

Proof: The argumentation is similar to Theorem 10 but here we employ the acceptability conditions ∇_m and ∇_d. □

One can further show the **Saturated Extension Theorem** (cf. [6](9.3)):

Theorem 29 *There is a saturated abstract consistency class in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$ that is an extension of all acceptable Γ_Σ in $\mathfrak{Acc}_{\beta\mathfrak{fb}}$.*

Given Theorem 7, one can view the Saturated Extension Theorem as an abstract cut-elimination result.

The proof of a model existence theorem employs Hintikka sets and in the context of studying Hintikka sets we have identified a phenomenon related to cut-strength which we call the **Impredicativity Gap**. That is, a Hintikka set $\mathcal{H}$ is saturated if any cut-strong formula $\mathbf{A}$ (e.g. a Leibniz equation $\mathbf{C} \doteq \mathbf{D}$) is in $\mathcal{H}$. Hence we can reasonably say there is a "gap" between saturated and unsaturated Hintikka sets. Every Hintikka set is either saturated or contains no cut-strong formulae.

7 Conclusion

We have shown that adding cut-strong formulae to a calculus for an impredicative logic is like adding cut. For machine-oriented automated theorem proving in impredicative logics — such as classical type theory — it is therefore not recommendable to naively add cut-strong axioms to the search space. In addition to the comprehension principle and the functional and Boolean extensionality axioms as elaborated in this paper the list of cut-strong axioms includes:

Other Forms of Defined Equality Formulas $\mathbf{A} \doteqdot^\alpha \mathbf{B}$ are 4-cut-strong in $\mathcal{G}_\beta$ where $\doteqdot^\alpha$ is $\lambda X_\alpha . \lambda Y_\alpha . \forall Q_{\alpha\to\alpha\to o} . (\forall Z_\alpha . (Q\ Z\ Z)) \Rightarrow (Q\ X\ Y)$ (cf. [3]).
Proof: Instantiate Q with $\lambda X_\alpha . \lambda Y_\alpha . \mathbf{C}$. □

Axiom of Induction The axiom of induction for the naturals $\forall P_{\iota\to o} . P0 \wedge (\forall X_\iota . PX \Rightarrow P(sX)) \Rightarrow \forall X_\iota . PX$ is 18-cut-strong in $\mathcal{G}_\beta$. (Other well-founded ordering axioms are analogous.)
Proof: Instantiate P with $\lambda X_\iota . a \doteq^o a$ for some parameter a_o. □

Axiom of Choice $\exists I_{(\alpha\to o)\to o}.\forall Q_{\alpha\to o}.\exists X_\alpha.QX \Rightarrow Q(IQ)$ is 7-cut-strong in $\mathcal{G}_\beta$.
Proof: Instantiate Q with $\lambda X_\alpha.\mathbf{C}$. □

Axiom of Description The description axiom $\exists I_{(\alpha\to o)\to o}.\forall Q_{\alpha\to o}.(\exists_1 Y_\alpha.QY) \Rightarrow Q(IQ)$ (see [2]), where $\exists_1 Y_\alpha.QY$ stands for $\exists Y_\alpha.QY \wedge (\forall Z_\alpha.QZ \Rightarrow Y \doteq Z)$ is 25-cut-strong in $\mathcal{G}_\beta$.
Proof: Instantiate Q with $\lambda X_\alpha.a \doteq^\alpha X$ for some parameter a_α. □

As Example 15 shows, comprehension axioms can be cut-strong. Church's formulation of type theory (cf. [9]) used typed λ-calculus to build comprehension principles into the language. One can view Church's formulation as a first step in the program to eliminate the need for cut-strong axioms. For the extensionality axioms a start has been made by the sequent calculi in this paper (and [6]), for resolution in [4] and for sequent calculi and extensional expansion proofs in [8]. The extensional systems in [8] also provide a complete method for using primitive equality instead of Leibniz equality. For improving the automation of higher-order logic our exploration thus motivates the development of higher-order calculi which directly include reasoning principles for equality, extensionality, induction, choice, description, etc., without using cut-strong axioms.

References

1. P. B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36(3):414–432, 1971.
2. P. B. Andrews. General models and extensionality. *Journal of Symbolic Logic*, 37(2):395–397, 1972.
3. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.
4. C. E. Benzmüller. *Equality and Extensionality in Automated Higher-Order Theorem Proving*. PhD thesis, Saarland University, 1999.
5. C. E. Benzmüller, C. E. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.
6. C. E. Benzmüller, C. E. Brown, and M. Kohlhase. Semantic techniques for higher-order cut-elimination. Seki Report SR-2004-07, Saarland University, 2004.
7. C. E. Benzmüller, C. E. Brown, and M. Kohlhase. Cut-simulation in impredicative logics (extended version). Seki Report SR-2006-01, Saarland University, 2006.
8. C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 2004.
9. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
10. K. J. J. Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica*, 8:7–55, 1955.
11. G. P. Huet. A mechanization of type theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 139–146, 1973.
12. B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.
13. R. M. Smullyan. A unifying principle for quantification theory. *Proc. Nat. Acad Sciences*, 49:828–832, 1963.
14. R. M. Smullyan. *First-Order Logic*. Springer, 1968.

CHRISTOPH BENZMÜLLER

COMPARING APPROACHES TO RESOLUTION BASED HIGHER-ORDER THEOREM PROVING

ABSTRACT. We investigate several approaches to resolution based automated theorem proving in classical higher-order logic (based on Church's simply typed λ-calculus) and discuss their requirements with respect to Henkin completeness and full extensionality. In particular we focus on Andrews' higher-order resolution (Andrews 1971), Huet's constrained resolution (Huet 1972), higher-order E-resolution, and extensional higher-order resolution (Benzmüller and Kohlhase 1997). With the help of examples we illustrate the parallels and differences of the extensionality treatment of these approaches and demonstrate that extensional higher-order resolution is the sole approach that can completely avoid additional extensionality axioms.

1. INTRODUCTION

It is a well known consequence of Gödel's first incompleteness theorem that there cannot be complete calculi for higher-order logic with respect to standard semantics. However, Henkin (1950) showed that there are indeed complete calculi if one gives up the intuitive requirement of full function domains in standard semantics and considers Henkin's general models instead. For higher-order calculi therefore Henkin completeness constitutes the most interesting notion of completeness.

A very challenging task for a calculus aiming at Henkin-completeness is to provide a suitable extensionality treatment. Unfortunately the importance of full extensionality in higher-order theorem proving, i.e., the suitable combination of functional and Boolean extensionality, has widely been overlooked so far. This might be due to the fact that (weak) functional extensionality is already built-in in the pure simply typed λ-calculus and that Boolean extensionality or the subtle interplay between Boolean and functional extensionality does simply not occur in this context. However, the situation drastically changes as soon as one is interested in a higher-order logic based on the simply typed λ-calculus, as now Boolean extensionality is of importance too.

We therefore investigate the extensionality treatment of several resolution based approaches to Henkin complete higher-order theorem proving:

Synthese **133:** 203–235, 2002.

Andrews' higher-order resolution (Andrews 1971), Huet's constrained resolution (Huet 1972), higher-order E-resolution, and extensional higher-order resolution (Benzmüller and Kohlhase 1998a). In order to ease the comparison we present them in a uniform way. Even though we focus on the resolution method in this paper the main results on the feasibility of extensionality reasoning in higher-order theorem proving do nevertheless apply to other theorem proving approaches as well.

For Andrews' and Huet's approach it is well known that generally infinitely many extensionality axioms are required in the search space in order to reach Henkin completeness. With the help of rather simple examples we will point out the shortcomings of this kind of extensionality treatment; namely a fair amount of non-goal directed search which contrasts the general idea of resolution based theorem proving.

Whereas the use of higher-order E-unification (cf. Snyder 1990; Nipkow and Qian 1991; Wolfram 1993; Qian and Wang 1996) instead of simple syntactical higher-order unification partially improves the situation, this idea nevertheless fails to provide a general solution and still requires additional extensionality axioms to ensure Henkin completeness.

The first calculus that generally takes into account, that higher-order theory unification with respect to theories including full extensionality is as hard as Henkin complete higher-order theorem proving itself, is the extensional higher-order resolution approach (Benzmüller and Kohlhase 1998a). This calculus very closely integrates higher-order unification and resolution by allowing for mutual recursive calls (instead of hierarchical calls solely from resolution to unification as in first-order). With its close integration of unification and resolution this approach ensures Henkin completeness without requiring additional extensionality axioms. With the help of our examples we show that this aspect is not only of theoretical but also of practical importance as proof problems requiring non-trivial extensionality reasoning can be solved in the extensional higher-order resolution approach in a more goal directed way.

As a theoretical result the paper presents Henkin completeness proofs for the resolution approaches of Andrews and Huet which have been examined in literature so far only with respect to Andrews' rather weak semantical notion of V-complexes.

The paper is organised as follows: Syntax and semantics of higher-order logic and a proof theoretic tool for analysing Henkin completeness are sketched in Section 2. Various resolution based calculi are then introduced in Sections 3 and their extensionality treatment is investigated with the help of examples in Section 4. Related work is addressed in Section 5, and Section 6 concludes the paper.

2. SYNTAX AND SEMANTICS OF HIGHER-ORDER LOGIC

2.1. *Classical Type Theory*

We consider a higher-order logic based on Church's simply typed λ-calculus (Church 1940) and choose $BT := \{\iota, o\}$ as *base types*, where ι denotes the set of individuals and o the set of truth values. *Functional types* are inductively defined over BT. A *signature* Σ contains for each type an infinite set of variables and constants, and particularly it provides the *logical constants* $\neg_{o\to o}$, $\vee_{o\to o\to o}$, and $\Pi_{(\alpha\to o)\to o}$ for every type α. As all other logical operators can be defined (e.g., $\mathbf{A} \wedge \mathbf{B} := \neg(\neg\mathbf{A} \vee \neg\mathbf{B})$, $\forall X_\alpha.\, \mathbf{P}\ X := \Pi_{((\alpha\to o)\to o)}(\lambda X_\alpha.\, \mathbf{P}\ X)$, and $\exists X_\alpha.\, \mathbf{P}\ X := \neg\forall X_\alpha.\, \neg(\mathbf{P}\ X)$)) the given logical constants are sufficient to define a classical higher-order logic.

The set of all Σ-terms (closed Σ-terms) of type α is denoted by wff_α ($cwff_\alpha$). Variables are printed as upper-case (e.g., X_α), constants as lower-case letters (e.g., c_α), and arbitrary terms appear as bold capital letters (e.g., $\mathbf{T}_\alpha$). If the type of a symbol is uniquely determined by the given context we omit it. We abbreviate function applications by $h_{\alpha_1\to\cdots\to\alpha_n\to\beta}\ \overline{\mathbf{U}^n_{\alpha_n}}$, which stands for $(\cdots(h_{\alpha_1\to\cdots\to\alpha_n\to\beta}\ \mathbf{U}^1_{\alpha_1})\cdots\mathbf{U}^n_{\alpha_n})$. For α-, β-, η-, $\beta\eta$-*conversion* and the definition of β-*normal,* $\beta\eta$-*normal, long* $\beta\eta$-*normal*, and *head-normal form* we refer to Barendregt (1984) as well as for the definition of *free* variables, *closed* formulas (also called *sentences*), and *substitutions*. Substitutions are represented as $[\mathbf{T}_1/X_1, \ldots, \mathbf{T}_n/X_n]$ where the X_i specify the variables to be replaced by the terms $\mathbf{T}_i$. The application of a substitution σ to a term (resp. literal or clause) $\mathbf{C}$ is printed $\mathbf{C}_\sigma$.

Higher-order unification and sets of *partial bindings* $\mathcal{GB}^h_\gamma$ are well explained in Snyder and Gallier (1989).

A calculus R provides a set of rules $\{r_n \mid 0 < n \leq i\}$ defined on clauses. We write $\Phi \vdash^{r_n} \mathcal{C}$ ($\mathcal{C}' \vdash^{r_n} \mathcal{C}$) iff clause $\mathcal{C}$ is the result of a one step application of rule $r_n \in R$ to premise clauses $\mathcal{C}'_i \in \Phi$ (to $\mathcal{C}'$ respectively). Multiple step derivations in calculus R are abbreviated by $\Phi_1 \vdash_R \Phi_k$ (or $\mathcal{C}_1 \vdash_R \mathcal{C}_k$).

2.2. *Clauses, Literals, and Unification Constraints*

The approaches studied in this paper are presented using a uniform notation for clauses, literals, and unification constraints (the notation is due to Kohlhase (1994)). *Literals*, e.g., $[\mathbf{A}]^\mu$, consist of a *literal atom* $\mathbf{A}$ and a *polarity* $\mu \in \{T, F\}$. For all rules presented in this paper we assume that the polarity specifiers $\mu, \nu \in \{T, F\}$ refer to complementary polarities, i.e., $\mu \neq \nu$. In particular we distinguish between *proper literals* and *pre-literals*. The (normalised) atom of a pre-literal has a logical constant at

head position, whereas this must not be the case for proper literals. For instance, $[\mathbf{A} \vee \mathbf{B}]^T$ is a pre-literal and $[p_{o\to o}\ (\mathbf{A} \vee \mathbf{B})]^T$ is a proper literal. Furthermore a literal is called *flexible* if its atom contains a variable at head position.

A unification problem between two terms $\mathbf{T}^1$ and $\mathbf{T}^2$ (between n terms $\mathbf{T}^1, \ldots, \mathbf{T}^n$) generated during the refutation process is called an *unification constraint* and is represented as $[\mathbf{T}^1 \neq^? \mathbf{T}^2]$ (resp. $[\neq^? (\mathbf{T}^1, \ldots, \mathbf{T}^n)]$). A unification constraint is called a *flex-flex pair* if both unification terms have *flexible* heads, i.e., variables at head position.

Clauses consist of disjunctions of literals or unification constraints. The unification constraints specify conditions under which the other literals are valid. For instance the clause $[p_{\alpha\to\beta\to o}\ \mathbf{T}^1_\alpha\ \mathbf{T}^2_\beta]^T \vee [\mathbf{T}^1_\alpha \neq^? \mathbf{S}^1_\alpha] \vee [\mathbf{T}^2_\beta \neq^? \mathbf{S}^2_\beta]$ can be informally read as: *if* $\mathbf{T}^1$ *is unifiable with* $\mathbf{S}^1$ *and* $\mathbf{T}^2$ *with* $\mathbf{S}^2$ *then* $(p\ \mathbf{T}^1\ \mathbf{T}^2)$ *holds*. We implicitly treat the disjunction operator $\vee$ in clauses as commutative and associative, i.e., we abstract from the particular order of the literals. Additionally we presuppose commutativity of $\neq^?$ and implicitly identify any two α-equal constraints or literals. Furthermore we assume that any two clauses have disjoint sets of free variables, i.e., for each freshly generated clause we choose new free variables.

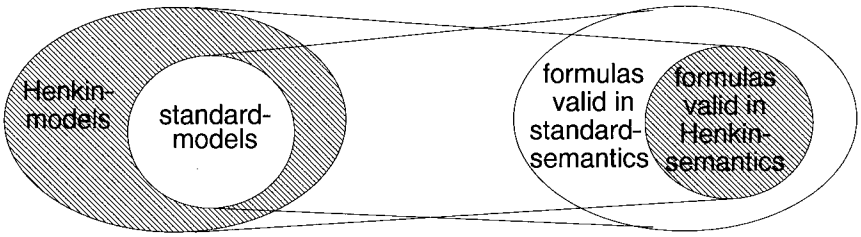
If a clause contains at least one pre-literal we call it a *pre-clause*, otherwise a *proper clause*. A clause is called *empty*, denoted by $\square$, if it consists only of (possibly none) *flex-flex* pairs.

An important aspect of clause normalisation is Skolemisation. In this paper we employ Miller's sound adaptation of traditional first-order Skolemisation (Miller 1983), which associates with each Skolem function the minimum number of arguments the Skolem function has to be applied to. Higher-order Skolemisation becomes sound, if any Skolem function f^n only occurs in a *Skolem term*, i.e., a formula $\mathbf{S} \equiv f^n\overline{\mathbf{A}^n}$, where none of the $\mathbf{A}^i$ contains a bound variable. Thus the Skolem terms only serve as descriptions of the existential witnesses and never appear as functions proper. Without this additional restriction the calculi do not really become unsound, but one can prove an instance of the axiom of choice. Andrews (1973) investigates the following instance: $\exists E_{(\iota\to o)\to o}.\ \forall P_{i\to o}.\ (\exists X_\iota.\ P\ X) \Rightarrow P\ (E\ P))$, which we want to treat as an optional axiom for the resolution calculi presented in this paper; for further details we refer to Miller (1983).

2.3. *Standard and Henkin Semantics*

A *standard model* for $\mathcal{HOL}$ provides a fixed set $\mathcal{D}_\iota$ of individuals, and a set $\mathcal{D}_o := \{\top, \bot\}$ of truth values. The domains for functional types are defined inductively: $\mathcal{D}_{\alpha\to\beta}$ is the set of all functions $f\colon \mathcal{D}_\alpha \to \mathcal{D}_\beta$. *Henkin models*

only require that $\mathcal{D}_{\alpha\to\beta}$ has enough members that any well-formed formula can be evaluated. Thus, the generalisation to Henkin models restricts the set of valid formulas sufficiently, such that complete calculi are possible. The following figure illustrates the sketched connection between standard- and Henkin semantics.



In Henkin and standard semantics Leibniz equality (which is defined as $\doteq^{\alpha} := \lambda X_{\alpha}.\, \lambda Y_{\alpha}.\, \forall P_{\alpha\to o}.\, P\ X \Rightarrow P\ Y$) denotes the intuitive identity relation and the (type parameterised) functional extensionality principles

$$\forall M_{\alpha\to\beta}.\, \forall N_{\alpha\to\beta}.\, (\forall X.\, M\ X \doteq N\ X) \Rightarrow (M \doteq N)$$

as well as the Boolean extensionality principle

$$\forall P_o.\, \forall Q_o.\, (P \Leftrightarrow Q) \Rightarrow (P \doteq Q)$$

are valid (cf. Benzmüller 1999a; Benzmüller and Kohlhase 1997). *Satisfiability* and *validity* ($\mathcal{M} \models \mathbf{F}$ or $\mathcal{M} \models \Phi$) of a formula $\mathbf{F}$ or set of formulas Φ in a model $\mathcal{M}$ are defined as usual.

We want to point out that the above statements on equality and extensionality do not apply to general models as originally introduced by Henkin (1950). Andrews (1972) showed that the sets $\mathcal{D}_{\alpha\to o}$ may be so sparse in Henkin's original notion of general models that Leibniz equality may denote a relation, which does not fulfil the functional extensionality principle. Due to lack of space we cannot present this general model here but refer to Andrews (1972) for further details. The solution suggested by Andrews is to presuppose the presence of the intuitive identity relations in all domains $\mathcal{D}_{\alpha\to\alpha\to o}$, which ensures the existence of unit sets $\{a\} \in \mathcal{D}_{\alpha\to o}$ for all elements $a \in \mathcal{D}_{\alpha}$. The existence of these unit sets in turn ensures that Leibniz equality indeed denotes the intended (fully extensional) identity relation.

In this paper, "Henkin semantics" means the corrected version of Henkin's original notion as given in Andrews (1972).

2.4. *Proving Completeness*

The abstract consistency proof principle (also called unifying principle) is a strong tool supporting the analysis of the connection between syntax and semantics for higher-order calculi. This proof principle has originally been introduced by Smullyan (1963) for first-order logic and has been adapted to higher-order logic by Andrews (1971). However, Andrews' adaptation allows completeness proofs only for the rather weak semantical notion of V-complexes (in which the axioms of extensionality may fail, cf. Benzmüller 1991; Benzmüller and Kohlhase 1997).

The following proof principle adapts Andrews abstract consistency proof principle to Henkin semantics.

DEFINITION 1 (*Acc* for Henkin Models). Let Σ be a signature and Γ_Σ a class of sets of Σ-sentences. If the following conditions hold for all $\mathbf{A}, \mathbf{B} \in cwff_o$, $\mathbf{F}, \mathbf{G} \in cwff_{\alpha\to\beta}$, and $\Phi \in \Gamma_\Sigma$, then we call Γ_Σ an *abstract consistency class for Henkin models*, abbreviated by *Acc*. (We want to point out that we assume an implicit treatment of α-convertibility here, whereas Andrews treats α-convertibility explicit in his notion of η-wffs; cf. Andrews (1971, 3.1.2, 2.7.5).)

saturated $\quad \Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ or $\Phi \cup \{\neg\mathbf{A}\} \in \Gamma_\Sigma$.

∇_c If $\mathbf{A}$ is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.

$\nabla_\neg$ If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$.

∇_β If $\mathbf{A} \in \Phi$ and $\mathbf{B}$ is the β-normal form of $\mathbf{A}$, then $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.

∇_η If $\mathbf{A} \in \Phi$ and $\mathbf{B}$ is the η-long form of $\mathbf{A}$, then $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.

$\nabla_\vee$ If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ or $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.

$\nabla_\wedge$ If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$.

$\nabla_\forall$ If $\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi \cup \{\mathbf{F}\ \mathbf{W}\} \in \Gamma_\Sigma$ for each $\mathbf{W} \in cwff_\alpha$.

$\nabla_\exists$ If $\neg\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi \cup \{\neg(\mathbf{F}\ w)\} \in \Gamma_\Sigma$ for any new constant $w \in \Sigma_\alpha$.

∇_b If $\neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Phi$, then $\Phi \cup \{\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$ or $\Phi \cup \{\neg\mathbf{A}, \mathbf{B}\} \in \Gamma_\Sigma$.

∇_q If $\neg(\mathbf{F} \doteq^{\alpha\to\beta} \mathbf{G}) \in \Phi$, then $\Phi \cup \{\neg(\mathbf{F}\ w \doteq^\beta \mathbf{G}\ w)\} \in \Gamma_\Sigma$ for any new constant $w \in \Sigma_\alpha$.

This definition extends Andrews notion of *abstract consistency classes for V-complexes* by the new requirements *saturated*, ∇_η, ∇_b, and ∇_q. *Saturatedness* turns the partial V-complexes into total structures and the latter two conditions ensure that Leibniz equality indeed denotes a fully extensional relation (which may not be the case in V-complexes, where Leibniz equality simply not necessarily denotes the intended identity relation; cf. Benzmüller 1991; Benzmüller and Kohlhase 1997).

The following model existence theorem is due to Andrews (1971).

THEOREM 2 (Henkin Model Existence (Andrews 1971)). Let Φ be a set of closed Σ-formulas, Γ_Σ be an abstract consistency class for V-complexes (i.e., Γ_Σ fulfils ∇_c, $\nabla_\neg$, ∇_β, $\nabla_\vee$, $\nabla_\wedge$, $\nabla_\forall$, $\nabla_\exists$), and let $\Phi \in \Gamma_\Sigma$. There exists a V-complex $\mathcal{M}$, such that $\mathcal{M} \models \Phi$.

The following related theorem addressing Henkin semantics (and additional ones addressing several notions in between Henkin semantics and V-complexes) is presented in Benzmüller (1999a); Benzmüller and Kohlhasse (1997).

THEOREM 3 (Henkin Model Existence (Benzmüller and Kohlhase 1998)). Let Φ be a set of closed Σ-formulas, Γ_Σ be an abstract consistency class for Henkin models, and let $\Phi \in \Gamma_\Sigma$. There exists a Henkin model $\mathcal{M}$, such that $\mathcal{M} \models \Phi$.

The complicated task of proving Henkin completeness for a given (resolution) calculus R can now be reduced to showing that the set of all sets Φ containing R-consistent closed formulas is an abstract consistency class for Henkin models, i.e., to verify the (syntactically checkable) conditions given in Definition 1.

3. HIGHER-ORDER RESOLUTION

In this section we introduce several higher-order resolution calculi. Additional approaches not mentioned here are briefly sketched and related to the presented ones in Section 5. The sketched approaches will be compared with respect to their extensionality treatment in Section 4.

3.1. *Andrews' Higher-Order Resolution $\mathcal{R}$*

We transform Andrews' higher-order resolution calculus (Andrews 1971) in our uniform notation. In the remainder of this paper we refer to this calculus with $\mathcal{R}$. Extending Andrews (1971) we show that $\mathcal{R}$ is Henkin

complete if one adds infinitely many extensionality axioms into the search space.

λ-*Conversion.* Calculus $\mathcal{R}$ provides two explicit rules addressing α-conversion and β-reduction (cf. Andrews 1971, 5.1.1) but does not provide a rule for η-conversion. Consequently η-equality of two terms (e.g., $f_{\iota\to\iota} \doteq \lambda X_\iota.\ f\ X$) cannot be proven in this approach without employing the functional extensionality axiom of appropriate type; cf. Section 4.1.

In our presentation we omit explicit rules for α- and β-convertibility and instead treat them implicitly, i.e., we assume that the presented rules operate on input and generate output in β-normal form and we automatically identify terms which differ only with respect to the names of bound variables.

Clause Normalisation. $\mathcal{R}$ introduces only four rules belonging to clause normalisation: negation elimination, conjunction elimination, existential elimination, and universal elimination (cf. Andrews 1971, 5.1.4.–5.1.7.). As our presentation of clauses in contrast to Andrews (1971) explicitly mentions the polarities of clauses and brackets the literal atoms we have to provide additional structural rules, e.g., the rule $\vee^T$.

- Negation elimination: $\dfrac{\mathbf{C} \vee [\neg\mathbf{A}]^T}{\mathbf{C} \vee [\mathbf{A}]^F}\ \neg^T \qquad \dfrac{\mathbf{C} \vee [\neg\mathbf{A}]^F}{\mathbf{C} \vee [\mathbf{A}]^T}\ \neg^F$
- Conjunction[1]/disjunction elimination:

$$\frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^T}{\mathbf{C} \vee [\mathbf{A}]^T \vee [\mathbf{B}]^T}\ \vee^T \qquad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{A}]^F}\ \vee_l^F \qquad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{B}]^F}\ \vee_r^F$$

- Existential[2]/universal elimination:

$$\frac{\mathbf{C} \vee [\Pi^\alpha \mathbf{A}]^T}{\mathbf{C} \vee [\mathbf{A}\ X_\alpha]^T}\ \Pi^T \qquad \frac{\mathbf{C} \vee [\Pi^\alpha \mathbf{A}]^F}{\mathbf{C} \vee [\mathbf{A}\ \mathrm{s}_\alpha]^F}\ \Pi^F$$

  X_α is a new free variable and s_α is a new Skolem term

Additionally Andrews presents rules addressing commutativity and associativity of the $\vee$-operator connecting the clauses literals (cf. Andrews 1971, 5.1.2.). We have already mentioned the implicit treatment of these aspects in Section 2.2.

In the remainder of this paper Cnf(**A**) denotes the set of clauses obtained from formula **A** by clause normalisation. It is easy to verify that clauses produced with Andrews' original normalisation rules can also be obtained with the rules presented here (and vice versa).

Resolution and Factorisation. Instead of a resolution and a factorisation rule – which work in connection with unification – Andrews presents a simplification and a cut rule. The cut rule is only applicable to clauses with two complementary literals which have identical atoms. Similarly Sim is defined only for clauses with two identical literals. In order to generate identical literal atoms during the refutation process these two rules have to be combined with the substitution rule Sub presented below.

- Simplification:
$$\frac{[\mathbf{A}]^{\mu} \vee [\mathbf{A}]^{\mu} \vee \mathbf{C}}{[\mathbf{A}]^{\mu} \vee \mathbf{C}}\ \text{Sim}$$

- Cut:
$$\frac{[\mathbf{A}]^{\mu} \vee \mathbf{C} \quad [\mathbf{A}]^{\nu} \vee \mathbf{D}}{\mathbf{C} \vee \mathbf{D}}\ \text{Cut}$$

Unification and Primitive Substitution. As higher-order unification was still an open problem in 1971 calculus $\mathcal{R}$ employs the British Museum Method instead, i.e., it provides a substitution rule that allows to blindly instantiate free variables by arbitrary terms. As the instantiated terms may contain logical constants, instantiation of variables in proper clauses may lead to pre-clauses, which must be normalised again with the clause normalisation rules.

- Substitution of arbitrary terms:
$$\frac{\mathcal{C}}{\mathcal{C}_{[\mathbf{T}_\alpha / X_\alpha]}}\ \text{Sub}$$

X_α is a free variable occurring in $\mathcal{C}$.

Extensionality Treatment. Calculus $\mathcal{R}$ does not provide rules addressing the functional and/or Boolean extensionality principles. Instead $\mathcal{R}$ assumes that the following extensionality axioms are (in form of respective clauses) explicitly added to the search space. And since the functional extensionality principle is parameterised over arbitrary functional types infinitely many functional extensionality axioms are required[3].

$$\text{EXT}^{\doteq}_{\alpha\to\beta}\colon \quad \forall F_{\alpha\to\beta}.\, \forall G_{\alpha\to\beta}.\, (\forall X_\beta.\, F\ X \doteq G\ X) \Rightarrow F \doteq G$$
$$\text{EXT}^{\doteq}_{o}\colon \quad \forall A_o.\, \forall B_o.\, (A \Leftrightarrow B) \Rightarrow A \doteq^{o} B$$

These are the crucial directions of the extensionality principles and the backward directions are not needed. The extensionality clauses derived from the extensionality axioms have the following form (note the many free variables, especially at literal head position, that are introduced into the search space – they heavily increase the amount of blind search in any attempt to automate the calculus):

$\mathcal{E}_1^{\alpha\to\beta} : [p\ (F\ s)]^T \vee [Q\ F]^F \vee [Q\ G]^T$ $\qquad$ $\mathcal{E}_1^{o} : [A]^F \vee [B]^F \vee [P\ A]^F \vee [P\ B]^T$
$\mathcal{E}_2^{\alpha\to\beta} : [p\ (G\ s)]^F \vee [Q\ F]^F \vee [Q\ G]^T$ $\qquad$ $\mathcal{E}_2^{o} : [A]^T \vee [B]^T \vee [P\ A]^F \vee [P\ B]^T$

$p_{\beta\to o}$, s_α are Skolem terms and $P_{(\alpha\to\beta)\to o}$, $Q_{(\alpha\to\beta)\to o}$ are new free variables.

Proof Search. Initially the proof problem is negated and normalised. The main proof search then starts with the normalised clauses and applies the cut and simplification rule in close connection with the substitution rule. An intermediate application of the clause normalisation rules may be needed to normalise temporarily generated pre-clauses. The extensionality treatment in $\mathcal{R}$ simply assumes to add at the beginning of the refutation process the above clauses obtained from the extensionality axioms.

When abstracting from the initial and intermediate normalisations the proof search can be illustrated as follows:

Completeness Results. Andrews (1971) gives a completeness proof for calculus $\mathcal{R}$ with respect to the semantical notion of V-complexes. As the extensionality principles are not valid in this rather weak semantical structures, the extensionality axioms are not needed in this completeness proof.

THEOREM 4 (V-completeness of $\mathcal{R}$). The calculus $\mathcal{R}$ is complete with respect to the notion of V-complexes.

Proof. We sketch the proof idea: 4(i) First show that the set of non-refutable sentences in $\mathcal{R}$ is an abstract consistency class for V-complexes. 4(ii) Then prove completeness of $\mathcal{R}$ with respect to V-complexes in an indirect argument: assuming non-completeness of $\mathcal{R}$ leads to an contradiction by 4(i) and Theorem 3. □

We now extend this result and prove Henkin completeness of calculus $\mathcal{R}$.

THEOREM 5 (Henkin completeness of $\mathcal{R}$). The calculus $\mathcal{R}$ is complete with respect to Henkin semantics provided that the infinitely many extensionality axioms are given.

Proof. 5(i) The crucial aspect is to prove that the set of non-refutable sentences in $\mathcal{R}$ enriched by the extensionality axioms is an abstract con-

sistency class for Henkin models. 5(ii) An indirect argument analogous to 4(ii) employing 5(i) and Theorem 3 ensures completeness.

In order to show 5(i) we have to verify the additional abstract consistency properties *saturated*, ∇_η, ∇_b, and ∇_q as specified in Definition 1.

saturated We show that $\Phi \cup \{\mathbf{A}\} \not\vdash_{\mathcal{R}} \square$ or $\Phi \cup \{\neg\mathbf{A}\} \not\vdash_{\mathcal{R}} \square$. Assume $\Phi \not\vdash_{\mathcal{R}} \square$ but $\Phi \cup \{\mathbf{A}\} \vdash_{\mathcal{R}} \square$ and $\Phi \cup \{\neg\mathbf{A}\} \vdash_{\mathcal{R}} \square$. By Lemma 6 (cf. below) we get $\Phi\{\mathbf{A} \vee \neg\mathbf{A}\} \vdash_{\mathcal{ER}} \square$, and hence, since $\mathbf{A} \vee \neg\mathbf{A}$ is a tautology, it must be the case that $\Phi \vdash_{\mathcal{ER}} \square$, which contradicts our assumption.

∇_η Assuming $\mathbf{A} \in \Phi$ and $\Phi \cup \{\mathbf{B}\} \vdash_{\mathcal{R}} \square$, we get $\Phi \vdash_{\mathcal{R}} \square$ by Lemma 7 (cf. below). This ensures the assertion by contraposition.

∇_b We first apply rule *Sub* and instantiate the variables A and B in the Boolean extensionality axioms $\mathcal{E}_1^o$ and $\mathcal{E}_2^o$ with terms $\mathbf{A}$ and $\mathbf{B}$. Now assume that $\neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Phi$ and $\Phi \cup \{\mathbf{A}, \neg\mathbf{B}\} \vdash_{\mathcal{R}} \square$ and $\Phi \cup \{\neg\mathbf{A}, \mathbf{B}\} \vdash_{\mathcal{R}} \square$. Employing the instantiated Boolean extensionality axioms it is easy to see that $\Phi \vdash_{\mathcal{R}} \square$, which ensures the assertion by contraposition.

∇_q Can be shown analogously to ∇_b when appropriately instantiating the functional extensionality axioms $\mathcal{E}_1^{\alpha\to\beta}$, $\mathcal{E}_2^{\alpha\to\beta}$.

LEMMA 6. Let Φ be a set of sentences and $\mathbf{A}, \mathbf{B}$ be sentences. If $\Phi \cup \{\mathbf{A}\} \vdash_{\mathcal{R}} \square$ and $\Phi \cup \{\mathbf{B}\} \vdash_{\mathcal{R}} \square$, then $\Phi \cup \{\mathbf{A} \vee \mathbf{B}\} \vdash_{\mathcal{R}} \square$.

Proof. We first verify that $\mathrm{Cnf}(\Phi * \mathbf{A} \vee \mathbf{B}) = \mathrm{Cnf}(\Phi) \cup (\mathrm{Cnf}(\mathbf{A}) \sqcup \mathrm{Cnf}(\mathbf{B}))$, where $\Gamma \sqcup \Delta = := \{\mathbf{C} \vee \mathbf{D} | \mathbf{C} \in \mathrm{Cnf}(A)\}, \mathbf{D} \in \mathrm{Cnf}(B)\}$. Then we use that $\Phi \cup (\Gamma_1 \sqcup \Gamma_2) \vdash_{\mathcal{R}} \square$, provided that $\Phi \cup \Gamma_1 \vdash_{\mathcal{R}} \square$ and $\Phi \cup \Gamma_2 \vdash_{\mathcal{R}} \square$. $\square$

LEMMA 7. Let Φ be a set of sentences and let $\mathbf{A}, \mathbf{B}$ be sentences in β-normal form, such that $\mathbf{A}$ can be transformed into $\mathbf{B}$ by (i) a one step η-expansion or (ii) a multiple step η-expansion. Then $\Phi \cup \{\mathbf{B}\} \vdash_{\mathcal{R}} \square$ implies $\Phi \cup \{\mathbf{A}\} \vdash_{\mathcal{R}} \square$.

Proof. Case (ii) can be proven by induction on the number of η-expansion steps employing (i) in the base case. To prove case (i) note that $\mathbf{A}$ and $\mathbf{B}$ differ (apart from α-equality) only with respect to a single subterm $\mathbf{T}_{\alpha\to\beta}$. More precisely, $\mathbf{A}_{[(\lambda X_\alpha \mathbf{T}\ X)/\mathbf{T}]}$ is equal to $\mathbf{B}$. Normalising sentences $\mathbf{A}$ (resp. $\mathbf{B}$) may result in several clauses $\mathcal{A}_1, \ldots, \mathcal{A}_n$ (resp. $\mathcal{B}_1, \ldots, \mathcal{B}_n$)

with duplicated occurrences of subterm $\mathbf{T}$ (resp. $\lambda X\mathbf{.}\ \mathbf{T}\ X$). We appropriately instantiate the functional extensionality axioms $\mathcal{E}_1^{\alpha\to\beta}$, $\mathcal{E}_2^{\alpha\to\beta}$ and derive the (Leibniz equation) clauses $\mathcal{C}_1 : [Q\ f]^T \vee [Q\ (\lambda X\mathbf{.}\ f\ X)]^F$ and $\mathcal{C}_2 : [Q'\ f]^F \vee [Q'\ (\lambda X\mathbf{.}\ f\ X)]^T$ (the latter can be obtained from the former by substituting $\lambda X\mathbf{.}\ \neg Q'\ X$ for Q). Obviously, we can derive for each $1 \leq i \leq n$ the clause $\mathcal{B}_i$ from its counterpart $\mathcal{A}_i$ with the help of $\mathcal{C}_1$ and $\mathcal{C}_2$ (formally we apply an induction on the occurrences of term $\mathbf{T}$ in $\mathcal{A}_i$). □

3.2. *Huet's Higher-Order Constrained Resolution $\mathcal{CR}$*

In this section we transform Huet's constrained resolution approach (Huet 1972, 1973a) to our uniform notation. The calculus here is the unsorted fragment of the variant of Huet's approach as presented in Kohlhase (1994). In the remainder of this paper we refer to this calculus as $\mathcal{CR}$. We extend (Huet 1972, 1973a) and show that $\mathcal{CR}$ is Henkin complete if we add infinitely many extensionality axioms to the search space.

λ-Conversion. Like $\mathcal{R}$ calculus $\mathcal{CR}$ assumes that terms, literals, and clauses are implicitly reduced to β-normal form. Furthermore we assume that α-equality is treated implicitly, i.e., we identify all terms that differ only with respect to the names of bound variables.

Clause Normalisation. Huet (1972) does not present clause normalisation rules but assumes that they are given. Here we employ the rules $\neg^T$, $\neg^F$, $\vee^T$, $\vee_l^F$, $\vee_r^F$, Π^T, and Π^F as already defined for calculus $\mathcal{R}$ in Section 3.1.

Resolution and Factorisation. As first-order unification is decidable and unitary it can be employed as a strong filter in first-order resolution (Robinson 1965). Unfortunately higher-order unification is not decidable (cf. Lucchesi 1972; Huet 1973b; Goldfarb 1981) and thus it can not be applied in the sense of a terminating side computation in higher-order theorem proving. Huet therefore suggests in Huet (1972, 1973a) to delay the unification process and to explicitly encode unification problems occurring during the refutation search as unification onstraints. In his original approach Huet presented a hyper-resolution rule which simultaneously resolves on the resolution literals $\mathbf{A}^1, \ldots \mathbf{A}^n$ $(1 \leq n)$ and $\mathbf{B}^1, \ldots \mathbf{B}^m$ $(1 \leq m)$ of two given clauses and adds the unification constraint $[\neq^? (\mathbf{A}^1, \ldots \mathbf{A}^n, \mathbf{B}^1, \ldots \mathbf{B}^m)]$ to the resolvent.

$$\frac{[\mathbf{A}^1]^\mu \vee \ldots \vee [\mathbf{A}^n]^\mu \vee \mathbf{C}[\mathbf{B}^1]^\mu \vee \ldots \vee [\mathbf{B}^m]^\mu \vee \mathbf{D}}{\mathbf{C} \vee \mathbf{D} \vee [\neq^? (\mathbf{A}^1, \ldots \mathbf{A}^n, \mathbf{B}^1, \ldots \mathbf{B}^m)]}\ \text{Hres}$$

In order to ease the comparison with the two other approaches discussed in this paper we instead employ a resolution rule Res and a factorisation rule Fac. Like Hres both rules encode the unification problem to be solved as a unification constraint.

- Constrained resolution:
$$\frac{[\mathbf{A}]^{\mu} \vee \mathbf{C} \quad [\mathbf{B}]^{\nu} \vee \mathbf{D}}{\mathbf{C} \vee \mathbf{D} \vee [\mathbf{A} \neq^{?} \mathbf{B}]} \text{ Res}$$

- Constrained factorisation:
$$\frac{[\mathbf{A}]^{\mu} \vee [\mathbf{B}]^{\mu} \vee \mathbf{C}}{[\mathbf{A}]^{\mu} \vee \mathbf{C} \vee [\mathbf{A} \neq^{?} \mathbf{B}]^{F}} \text{ Fac}$$

One can easily prove by induction on $n + m$ that each proof step applying rule Hres can be replaced by a corresponding derivation employing Res and Fac. For a formal proof note that the unification constraint $[\neq^{?} (\mathbf{A}^1, \ldots \mathbf{A}^n, \mathbf{B}^1, \ldots \mathbf{B}^m)]$ is equivalent to $[\mathbf{A}^1 \neq^{?} \mathbf{A}^2] \vee [\mathbf{A}^2 \neq^{?} \mathbf{A}^3] \vee \ldots \vee [\mathbf{A}^{n-1} \neq^{?} \mathbf{A}^n] \vee [\mathbf{A}^n \neq^{?} \mathbf{B}^1] \vee [\mathbf{B}^1 \neq^{?} \mathbf{B}^2] \vee [\mathbf{B}^2 \neq^{?} \mathbf{B}^3] \vee \ldots \vee [\mathbf{B}^{n-1} \neq^{?} \mathbf{B}^n]$.

Unification and Splitting. Huet (1975) introduces higher-order unification and higher-order pre-unification and shows that higher-order pre-unification is sufficient to verify the soundness of a refutation in which the occurring unification problems have been delayed until the end. The higher-order pre-unification rules presented here are discussed in detail in Benzmüller (1999a). They furthermore closely reflect the rules as presented in Snyder and Gallier (1989).

- Elimination of trivial pairs:
$$\frac{\mathbf{C} \vee [\mathbf{A} \neq^{?} \mathbf{A}]}{\mathbf{C}} \text{ Triv}$$

- Decomposition
$$\frac{\mathbf{C} \vee [\mathbf{A}_{\alpha\to\beta}\, \mathbf{C}_{\alpha} \neq^{?} \mathbf{B}_{\alpha\to\beta}\, \mathbf{D}_{\alpha}]}{\mathbf{C} \vee [\mathbf{A} \neq^{?} \mathbf{B}] \vee [\mathbf{C} \neq^{?} \mathbf{D}]} \text{ Dec}$$

- Elimination of λ-binders:
(weak functional extensionality)
$$\frac{\mathbf{C} \vee [\mathbf{M}_{\alpha\to\beta} \neq^{?} \mathbf{N}_{\alpha\to\beta}]}{\mathbf{C} \vee [\mathbf{M}\, s_{\alpha} \neq^{?} \mathbf{N}\, s_{\alpha}]} \text{ Func}$$
s_{α} is a new Skolem term.

- Imitation of rigid heads:
$$\frac{\mathbf{C} \vee [F_{\gamma}\, \overline{\mathbf{U}^n} \neq^{?} h\, \overline{\mathbf{V}^m}] \quad \mathbf{G} \in \mathcal{GB}^{h}_{\gamma}}{\mathbf{C} \vee [F \neq^{?} \mathbf{G}] \vee [F\, \overline{\mathbf{U}^n} \neq^{?} h\, \overline{\mathbf{V}^m}]} \text{ FlexRigid}$$
$\mathcal{GB}^{h}_{\gamma}$ is the set of partial bindings of type γ for head h as defined in Snyder and Gallier (1989).

Huet points to the usefulness of eager unification to filter out clauses with non-unifiable unification constraints or to back-propagate the solutions of easily solvable constraints (e.g., in case of first-order unification

problems occurring during the proof search). Many of the higher-order unification problems occurring in practice are decidable and have only finitely many solutions. Hence, even though higher-order unification is generally not decidable it is sensible in practice to apply the unification algorithm with a particular resource[4], such that only those unification problems which may have further solutions beyond this bound need to be delayed. In our presentation of calculus $\mathcal{CR}$ we explicitly address the aspect of eager unification and substitution by rule Subst. This rule back-propagates eagerly computed unifiers to the literal part of a clause.

- Eager unification and substitution:

$$\frac{\mathbf{C} \vee [X \stackrel{?}{\neq} \mathbf{A}] \quad X \notin \mathrm{free}(\mathbf{A})}{\mathbf{C}_{[\mathbf{A}/X]}}\ \mathrm{Subst}$$

  Rule Subst is applicable provided that $[X \stackrel{?}{\neq} \mathbf{A}]$ is solved with respect to the other unification constraints in $\mathbf{C}$, i.e., that there is no conflict with other unification constraints.

The literal heads of our clauses may consist of set variables and it may be necessary to instantiate them with terms introducing new logical constant at head position in order to find a refutation. Unfortunately not all appropriate instantiations can be computed with the calculus rules presented so far. To address this problem Huet's approach provides the following splitting rules:

1. Instantiate set variables:

$$\frac{[P\ \mathbf{A}]^T \vee \mathbf{C}}{[Q]^T \vee [R]^T \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ (Q_o \vee R_o)]}\ \mathrm{S}^T_{\vee}$$

$$\frac{[P\ \mathbf{A}]^\mu \vee \mathbf{C}}{[Q]^\nu \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ \neg Q_o]}\ \mathrm{S}^{TF}_{\neg} \qquad \frac{[P\ \mathbf{A}]^F \vee \mathbf{C}}{\begin{array}{l}[Q]^F \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ (Q_o \vee R_o)] \\ [R]^F \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ (Q_o \vee R_o)]\end{array}}\ \mathrm{S}^F_{\vee}$$

$$\frac{[P\ \mathbf{A}_{\alpha\to o}]^T \vee \mathbf{C}}{[M_{\alpha\to o}\ Z]^T \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ \Pi^\alpha M]}\ \mathrm{S}^T_{\Pi}$$

$$\frac{[P\ \mathbf{A}_{\alpha\to o}]^F \vee \mathbf{C}}{[M_{\alpha\to o}\ s]^F \vee \mathbf{C} \vee [P\ \mathbf{A}\ \stackrel{?}{\neq}\ \Pi^\alpha M]}\ \mathrm{S}^F_{\Pi}$$

   S^T_{Π} and S^F_{Π} are infinitely branching as they are parameterised over type α. Q_o, R_o, $M_{\alpha\to o}$, Z_α are new variables and s_α is a new Skolem constant.

A theorem which is not refutable in $\mathcal{CR}$ if the splitting rules are not available is $\exists A_o.A$. After negation this statement normalises to clause $\mathcal{C}_1 : [A]^F$, such that none but the splitting rules are applicable. With the help of

rule $S^{TF}_{\neg}$ and eager unification, however, we can derive $\mathcal{C}_2 : [A']^T$ which is then successfully resolvable against $\mathcal{C}_1$.

Extensionality Treatment. On the one hand η-convertibility is built-in in higher-order unification, such that calculus $\mathcal{CR}$ already supports functional extensionality reasoning to a certain extent. On the other hand $\mathcal{CR}$ nevertheless fails to address full extensionality as it does not realise the required subtle interplay between the functional and Boolean extensionality principles. For example, without employing additional Boolean and functional extensionality axioms $\mathcal{CR}$ cannot prove the rather simple Examples presented in Sections 4.2, 4.3, and 4.4.

Proof Search. Initially the proof problem is negated and normalised. The main proof search then operates on the generated clauses by applying the resolution, factorisation, and splitting rules. Despite the possibility of eager unification $\mathcal{CR}$ generally foresees to delay the higher-order unification process in order to overcome the undecidability problem. When deriving an empty clause $\mathcal{CR}$ then tests whether the accumulated unification constraints justifying this particular refutation are solvable. Like $\mathcal{R}$, the extensionality treatment of $\mathcal{CR}$ requires the addition of infinitely many extensionality axioms to the search space. The following figure graphically illustrates the main ideas of the proof search in $\mathcal{CR}$.

Completeness Results. Huet (1972, 1973a) analyses completeness of $\mathcal{CR}$ only with respect to Andrews V-complexes, i.e., Huet verifies that the set of non-refutable sentences in $\mathcal{CR}$ is an abstract consistency class for V-complexes.

THEOREM 8 (V-completeness of $\mathcal{CR}$). The calculus $\mathcal{CR}$ is complete with respect to the notion of V-complexes.

We now extend this result and prove Henkin completeness of calculus $\mathcal{CR}$.

THEOREM 9 (Henkin completeness of $\mathcal{CR}$). The calculus $\mathcal{CR}$ is complete wrt. Henkin semantics provided that the infinitely many extensionality axioms are given.

Proof. Analogously to the proof of Theorem 5 we can reduce the problem to verifying that the set of non-refutable sentences in $\mathcal{R}$ enriched by the extensionality axioms is an abstract consistency class for Henkin models. The assertion then follows in an indirect argument employing Theorem 3. In addition to the abstract consistency properties already examined in Huet (1972, 1973a) for Theorem 8 we have to verify *saturatedness*, ∇_η, ∇_b, and ∇_q as specified in Definition 1. The proofs of all four statements are analogous to the corresponding parts in the proof of Theorem 5. For *saturatedness* and ∇_η we use analogues of Lemmas 6 and 7.

LEMMA 10. Let Φ be a set of sentences and $\mathbf{A}, \mathbf{B}$ be sentences. If $\Phi \cup \{\mathbf{A}\} \vdash_{\mathcal{CR}} \square$ and $\Phi \cup \{\mathbf{B}\} \vdash_{\mathcal{CR}} \square$, then $\Phi \cup \{\mathbf{A} \vee \mathbf{B}\} \vdash_{\mathcal{CR}} \square$

Proof. Analogous to the proof of Lemma 6. □

LEMMA 11. Let Φ be a set of sentences and let $\mathbf{A}, \mathbf{B}$ be sentences in β-normal form, such that $\mathbf{A}$ can be transformed into $\mathbf{B}$ by (i) a one step η-expansion or (ii) a multiple step η-expansion. Then $\Phi \cup \{\mathbf{B}\} \vdash_{\mathcal{CR}} \square$ implies $\Phi \cup \{\mathbf{A}\} \vdash_{\mathcal{CR}} \square$.

Proof. The proof is analogous to Lemma 7. The main difference is with regard to the derivability of the clauses $\mathcal{B}_i$ from its counterparts $\mathcal{A}_i$ with the help of $\mathcal{C}_1$ and $\mathcal{C}_2$ obtained from the (suitably instantiated) functional extensionality axioms. It might be the case that the terms $\mathbf{T}$ occur inside flexible literals of the clauses $\mathcal{A}_i$. Resolving these flexible literals against $\mathcal{C}_1$ and $\mathcal{C}_2$ results then in flex-flex pairs that cannot be solved eagerly but have to be delayed. E.g., let $\mathcal{A}_j$ $(1 \leq j \leq n)$ be of form $[R\ (p\ \mathbf{T})]^\nu \vee \mathcal{D}$. Instead of $\mathcal{B}_j := [R\ (p\ (\lambda X_{\bullet}\ \mathbf{T}\ X))]^\nu \vee \mathcal{D}$ we can derive only $\mathcal{B}'_j := [Q\ (\lambda X_{\bullet}\ \mathbf{T}\ X)]^\nu \vee \mathcal{D} \vee [Q\ T \neq^?\ R\ (p\ (\lambda X_{\bullet}\ \mathbf{T}\ X))]$. Hence, we have to show (in a technically rather complicated inductive proof on the length of the derivation) that each refutation employing $\mathcal{B}'_j$ can be replaced by a corresponding one employing $\mathcal{B}_j$. □

3.3. *Higher-Order E-Resolution CRE*

Some more recent approaches to higher-order theorem proving employ equational higher-order unification instead of syntactical higher-order unification in order to ease and shorten proofs on the resolution layer by relocating particular computation or reasoning tasks to the unification process. For instance, equational higher-order unification has been investigated within the contexts of higher-order rewriting and narrowing (cf. Nipkow and Prehofer 1998; Prehofer 1998), and within the context of restricted higher-order E-resolution (Wolfram 1993).

In this Section we will sketch a higher-order E-resolution approach based on calculus $\mathcal{CR}$. In contrast to the other investigated calculi the aim thereby is not to provide a detailed description of the particular rules and the functioning of the calculus, but to provide a sufficient basis for the investigation to what extent equational higher-order unification can improve the extensionality reasoning in a higher-order theorem prover.

Generally unification of two (or several) terms $\mathbf{S}$ and $\mathbf{T}$ aims at computing sets of unifiers, i.e., substitutions σ, such that $\mathbf{S}_\sigma$ equals $\mathbf{T}_\sigma$ ($\mathbf{S}_\sigma = \mathbf{T}_\sigma$). Equational unification thereby extends syntactical unification in the sense that it tries to equalise $\mathbf{S}_\sigma$ and $\mathbf{T}_\sigma$ modulo a fixed equational theory E (written as $\mathbf{S}_\sigma =_E \mathbf{T}_\sigma$) instead of equalising them syntactically. A survey to unification theory is given in Baader and Siekmann (1994), and Siekmann (1989).

Within our higher-order context we assume that an equational theory E is defined by a fixed set of equations between closed λ-terms. For instance, equations expressing commutativity and associativity of the $\wedge$-operator are $(\lambda X_o.\, \lambda Y_o.\, X \wedge Y) = (\lambda X_o.\, \lambda Y_o.\, Y \wedge X)$ and $(\lambda X_o.\, \lambda Y_o.\, \lambda Z_o.\, (X \wedge Y) \wedge Z) = (\lambda X_o.\, \lambda Y_o.\, \lambda Z_o.\, X \wedge (Y \wedge Z))$.

And within this particular theory E (to be more precise modulo the congruence relation defined by this equations) the following two terms are unifiable by $[a/X]$: $(p_{o\to o}\ (b_o \wedge X_o) \wedge (X_o \wedge b_o))$ and $(p_{o\to o}\ a_o \wedge (a_o \wedge (b_o \wedge b_o)))$.

We want to point out that Huet's unification approach as presented for calculus $\mathcal{CR}$ is of course not a pure syntactical one as it already takes $\alpha\beta\eta$-equality into account. We nevertheless call Huet's approach *syntactical higher-order unification* in this paper in order to distinguish it from equational higher-order unification in the sense of this Subsection, where the theory E may contain additional higher-order equations.

Several, often restricted, approaches to higher-order E-unification have been discussed in literature. Wolfram (1993) a general higher-order E-unification approach which employs higher-order rewriting techniques. An approach restricted to first-order theories is given in Snyder (1990) and another restricted one, where as much computation as possible is pushed to a first-order E-unification procedure, is discussed in Qian and Wang (1996) and Nipkow and Qian (1991). Dougherty and Johann (1992) presents a restricted combinatory logic approach.

We now sketch our higher-order E-resolution approach $\mathcal{CRE}$.

Clause Normalisation, Resolution and Factorisation, and Splitting. We assume that calculus $\mathcal{CRE}$ coincides with calculus $\mathcal{CR}$ in all but the uni-

fication part. Thus $\mathcal{CR}$ provides the clause normalisation, resolution and factorisation, and splitting rules as introduced in Section 3.2.

Equational Unification. Instead of presenting a concrete set of rules for higher-order E-unification we refer to the respective approaches given in Snyder (1990), Nipkow and Qian (1991), Wolfram (1993), and Qian and Wang (1996). For our investigation of $\mathcal{CRE}$ it will be of minor importance which particular approach we choose and how general this approach is.

Whereas higher-order E-unification can indeed partially improve the extensionality treatment in $\mathcal{CRE}$, we will present simple theorems in Section 4 which cannot be proven in $\mathcal{CRE}$ (or in any of the related approaches mentioned above) without additional extensionality axioms. These counterexamples do not depend on the concrete choice of an equational theory E.

3.4. *Extensional Higher-Order Resolution $\mathcal{ER}$*

We now present the extensional higher-order resolution approach as introduced in Benzmüller and Kohlhase (1998a), Benzmüller (1991a). In the remainder of this paper we refer to this calculus as $\mathcal{ER}$. $\mathcal{ER}$ is Henkin complete without requiring additional extensionality axioms.

λ-Conversion. In contrast to $\mathcal{R}$ and $\mathcal{CR}$ calculus $\mathcal{ER}$ assumes that all terms, literals, and clauses are implicitly reduced to long $\beta\eta$-normal form.

Clause Normalisation, Resolution and Factorisation, and Unification and Splitting. $\mathcal{ER}$ employs the normalisation rules $\neg^T$, $\neg^F$, $\vee^T$, $\vee^F_l$, $\vee^F_r$, Π^T, Π^F, the resolution and factorisation rules Res, Fac, and the unification rules Triv, Dec, Func, FlexRigid, Subst as already defined for calculus $\mathcal{CR}$ in Section 3.2.

Additionally $\mathcal{ER}$ employs the infinitely branching unification rule FlexFlex, which guesses instances in case of flex-flex pairs (cf. Conjecture 13 in Section 3.4).

- Guess
$$\frac{\mathbf{C} \vee [F_{\overline{\gamma^n}\to\alpha}\ \overline{\mathbf{U}^n} = H_{\overline{\delta^m}\to\alpha}\ \overline{\mathbf{V}^m}]^F \quad \mathbf{G} \in \mathcal{GB}^h_{\overline{\gamma^n}\to\alpha}}{\mathbf{C} \vee [F\ \overline{\mathbf{U}^n} = H\ \overline{\mathbf{V}^m}]^F \vee [F = \mathbf{G}]^F}\ \mathit{FlexFlex}$$
$\mathcal{GB}^h_{\overline{\gamma^n}\to\alpha}$ is the set of partial bindings of type γ for a constant h in the given signature.

The splitting rules presented for $\mathcal{CR}$ in Section 3.2 are replaced in $\mathcal{ER}$ by the more elegant primitive substitution rule as first introduced by Andrews (1989).

- Primitive substitution
$$\frac{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \quad \mathbf{P} \in \mathcal{GB}_\gamma^{\{\neg,\vee\}\cup\{\Pi^\beta|\beta\in\mathcal{T}\}}}{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \vee [Q = \mathbf{P}]^F}\ \mathit{Prim}$$

  $\mathcal{GB}_\gamma^{\{\neg,\vee\}\cup\{\Pi^\beta|\beta\in\mathcal{T}\}}$ is the set of partial bindings of type o for a logical constant in the signature.

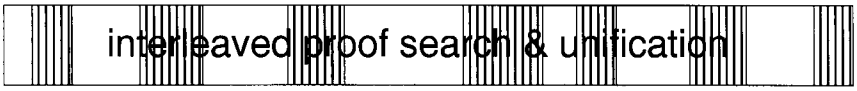
Extensionality Treatment. Instead of adding infinitely many extensionality axioms to the search space $\mathcal{CR}$ provides two new extensionality rules which closely connect refutation search and eager unification. The idea is to allow for recursive calls from higher-order unification to the overall refutation process. This turns the rather weak syntactical higher-order unification approach considered so far into a most general approach for *dynamic* higher-order theory unification.

- Unification and equivalence:
$$\frac{\mathbf{C} \vee [\mathbf{M}_o \neq^? \mathbf{N}_o]}{\mathbf{C} \vee [\mathbf{M}_o \Leftrightarrow \mathbf{N}_o]^F}\ \text{Equiv}$$
- Unification and Leibniz equality:
$$\frac{\mathbf{C} \vee [\mathbf{M}_\alpha \neq^? \mathbf{N}_\alpha]}{\mathbf{C} \vee [\forall P_{\alpha\to o}.\ P\ \mathbf{M} \Rightarrow P\ \mathbf{N}]^F}\ \text{Leib}$$

Proof Search. Initially the proof problem is negated and normalised. The main proof search then closely interleaves the refutation process on resolution layer and unification, i.e., the main proof search rules Res, Fac, and Prim and the unification rules are integrated at a common conceptual level. The calls from unification to the overall refutation process with rules Leib and Equiv introduce new clauses into the search space which can be resolved against already given ones.

This close interplay between unification and refutation search compensates the infinitely many extensionality axioms required in $\mathcal{R}$ and $\mathcal{CR}$ by a more goal-directed approach to full extensionality reasoning.

The following picture graphically illustrates the main ideas of the proof search in $\mathcal{ER}$.



Completeness Results. Henkin completeness of the presented approach with rule FlexFlex is analysed in detail in Benzmüller (1999a) and Benzmüller and Kohlhase (1998a). Here we only mention the main result:

THEOREM 12 (Henkin completeness of $\mathcal{ER}$). The calculus $\mathcal{ER}$ is complete with respect to Henkin semantics.

Benzmüller (1999a) presents but does not prove the following interesting claims which are of major practical importance as they will lead to an enormous reduction of the search spaces in $\mathcal{ER}$.

CONJECTURE 13 (FlexFlex-rule is not needed). Rule FlexFlex can be avoided in $\mathcal{ER}$ without affecting Henkin completeness.

CONJECTURE 14 (Base type restriction of rule Leib). Rule Leib can be restricted to base types α in $\mathcal{ER}$ without affecting Henkin completeness.

4. EXAMPLES

In this section we compare the extensionality treatment provided by the calculi $\mathcal{R}$, $\mathcal{CR}$, $\mathcal{CRE}$, and $\mathcal{ER}$ with the help of simple examples. Despite their simplicity the latter two of these examples are nevertheless challenging with respect to their automisation in a higher-order theorem prover.

4.1. *η-Equality*

EXAMPLE 15. $f_{\iota\to\iota} \doteq \lambda X_\iota.\ f\ X$

Solution in $\mathcal{R}$. In order to prove Example 15, which normalises after negation and expansion of Leibniz equality to $\mathcal{C}_1 : [q\ f]^F$ and $\mathcal{C}_2 : [q\ (\lambda X.\ f\ X)]^T$ where $q_{(\iota\to\iota)\to o}$ is a new Skolem term, we first have to appropriately instantiate the two functional extensionality clauses $\mathcal{E}_1^{\alpha\to\beta}$ and $\mathcal{E}_2^{\alpha\to\beta}$ with the help of rule Sub:

$$\mathcal{E}_1^{\iota\to\iota} : [p\ (f\ s)]^T \vee [Q\ f]^F \vee [Q\ (\lambda X.\ f\ X)]^T$$
$$\mathcal{E}_2^{\iota\to\iota} : [p\ (f\ s)]^F \vee [Q\ f]^F \vee [Q\ (\lambda X.\ f\ X)]^T$$

Employing cut and simplification we can derive

$$\mathcal{C}_3 : [Q\ f]^F \vee [Q\ (\lambda X.\ f\ X)]^T$$

which corresponds to the Leibniz equation between f and $(\lambda X.\ f\ X)$. With rule *Sub* we then substitute the term $\lambda M_{\iota\to\iota}.\ \neg(q\ M)$ for the predicate variable Q, re-normalise the generated pre-clause, and obtain

$$\mathcal{C}_4 : [q\ f]^T \vee [q\ (\lambda X.\ f\ X)]^F$$

By applying the cut rule to $\mathcal{C}_4$, $\mathcal{C}_1$, and $\mathcal{C}_2$ we then derive $\square$.

Solution in $\mathcal{CR}$, $\mathcal{CRE}$, and $\mathcal{ER}$. We first sketch the proof of Example 15 in $\mathcal{CR}$. Initially we resolve on $\mathcal{C}_1 : [q\ f]^F$ and $\mathcal{C}_2 : [q\ (\lambda X.\ f\ X)]^T$ and thereby obtain the unification constraint $\mathcal{C}_3 : \square \vee [f \neq^? (\lambda X.\ f\ X)]^F$. The η-equality of the two unification terms is shown with the help of the unification rule *Func* which derives the trivial unification constraint $\mathcal{C}_4 : \square \vee [f\ s \neq^?\ f\ s]^F$ (where s_ι is new Skolem term). This unification constraint can be subsequently eliminated with rule Triv. Our examples illustrates higher-order unification already addresses weak functional extensionality (η-equality).

An analogous refutation can clearly be employed in calculus $\mathcal{CRE}$ as weak functional extensionality is built-in in higher-order E-unification as well.

Example 15 is trivially solvable in $\mathcal{ER}$ due to the fact that we implicitly assume all terms to be in long $\beta\eta$-normal form, i.e., the clauses to be refuted are $\mathcal{C}_1 : [q(\lambda X.\ fX)]^F$ and $\mathcal{C}_2 : [q(\lambda X.\ fX)]^T$. Clearly, when considering long $\beta\eta$-normal forms instead of β-normal forms the problem is trivially solvable in calculi $\mathcal{R}$, $\mathcal{CR}$, and $\mathcal{CRE}$ as well.

4.2. *Set Descriptions*

In higher-order logic sets can be elegantly encoded by characteristic functions. An interesting problem then is to investigate whether two encodings describe the same set. The following trivial example demonstrates the importance of the extensionality principles for this purpose.

EXAMPLE 16. The set of all red balls equals the set of all balls that are red: $\{X|\text{red}\ X \wedge \text{ball}\ X\} = \{X|\text{ball}\ X \wedge \text{red}\ X\}$. This problem can be encoded as $(\lambda X_\iota.\ \text{red}\ X \wedge \text{ball}\ X) = (\lambda X_\iota.\ \text{ball}\ X \wedge \text{red}\ X)$.

Negation, expansion of Leibniz equality, and clause normalisation leads to the following clauses (where $p_{(\iota\to o)\to o}$ is a new Skolem constant):

$$\mathcal{C}_1 : [p\ (\lambda X.\ \text{red}\ X \wedge \text{ball}\ X)]^F \qquad \mathcal{C}_2 : [p\ (\lambda X.\ \text{ball}\ X \wedge \text{red}\ X)]^T$$

Solution in $\mathcal{R}$. As no rule is applicable to $\mathcal{C}_1$ and $\mathcal{C}_2$ Example 16 is not refutable in $\mathcal{R}$ without employing extensionality axioms. The only way to derive a contradiction is to employ suitable instances of the extensionality clauses in a rather complicated derivation:

1. With rule Sub instantiate the Boolean extensionality axioms $\mathcal{E}_1^o$ and $\mathcal{E}_2^o$ with the terms (red $Y \wedge$ ball Y) and (ball $Y \wedge$ red Y) for variables A

and B. By normalising and employing simplification exhaustively to the resulting pre-clauses we obtain among others:

$$\begin{array}{lllll} \mathcal{C}_3: & [\text{red } Y]^F \vee & [\text{ball } Y]^F \vee & [P\ \mathbf{F}^1]^F \vee & [P\ \mathbf{G}^1]^T \\ \mathcal{C}_4: & [\text{red } Y]^T \vee & & [P\ \mathbf{F}^1]^F \vee & [P\ \mathbf{G}^1]^T \\ \mathcal{C}_5: & & [\text{ball } Y]^T \vee & [P\ \mathbf{F}^1]^F \vee & [P\ \mathbf{G}^1]^T \end{array}$$

where $\mathbf{F}^1$ stands for the term (red $Y \wedge$ ball Y) and $\mathbf{G}^1$ for (ball $Y \wedge$ red Y).

From $\mathcal{C}_3$–$\mathcal{C}_5$ we derive $\mathcal{C}_6 : [P\ \mathbf{F}^1]^F \vee [P\ \mathbf{G}^1]^T$ by cut and simplification, where $\mathcal{C}_6$ corresponds to the clause normal form of $\forall Y.\ ((\lambda X.\ \text{red } X \wedge \text{ball } X)\ Y) \doteq ((\lambda X.\ \text{ball } X \wedge \text{red } X)\ Y)$.

2. With rule Sub we now instantiate the functional extensionality axioms $\mathcal{E}_1^{\iota\to o}$ and $\mathcal{E}_2^{\iota\to o}$ with terms $\mathbf{F}^2 := (\lambda X.\ \text{red } X \wedge \text{ball } X)$ for variable F and $\mathbf{G}^2 := (\lambda X.\ \text{ball } X \wedge \text{red } X)$ for variable G.

$$\mathcal{C}_7 : [q\ (\text{red } s \wedge \text{ball } s)]^T \vee [Q\ \mathbf{F}^2]^F \vee [Q\ \mathbf{G}^2]^T$$
$$\mathcal{C}_8 : [q\ (\text{ball } s \wedge \text{red } s)]^F \vee [Q\ \mathbf{F}^2]^F \vee [Q\ \mathbf{G}^2]^T$$

3. Applying substitution $[(\lambda Z.\ q\ Z)/P, s/Y]$ with rule Sub to clause $\mathcal{C}_6$ leads to:

$$\mathcal{C}_9 : [q\ (\text{red } s \wedge \text{ball } s)]^F \vee [q\ (\text{ball } s \wedge \text{red } s)]^T$$

Applying cut and simplification we combine the results of the above steps and derive from $\mathcal{C}_7$, $\mathcal{C}_8$, and $\mathcal{C}_9$

$$\mathcal{C}_{10} : [Q\ (\lambda X.\ \text{red } X \wedge \text{ball } X)]^F \vee [Q\ (\lambda X.\ \text{ball } X \wedge \text{red } X)]^T$$

which represent the Leibniz equation between $(\lambda X.\ \text{red } X \wedge \text{ball } X)$ and $(\lambda X.\ \text{ball } X \wedge \text{red } X)$. With the help of $\mathcal{C}_1$ and $\mathcal{C}_2$ we can now derive $\square$ after appropriately instantiating $\mathcal{C}_{10}$ with $[p/Q]$.

Note that in Steps 1 and 2 we had to guess the *right* instantiations of the extensionality axioms and to apply non-goal directed forward reasoning.

Solution in $\mathcal{CR}$. The only rule that is applicable to $\mathcal{C}_1$ and $\mathcal{C}_2$ in calculus $\mathcal{CR}$ is the resolution rule Res leading to the following unification constraint

$$\mathcal{C}_3 : \square \vee [p\ (\lambda X.\ \text{red } X \wedge \text{ball } X) \neq^?\ p\ (\lambda X.\ \text{ball } X \wedge \text{red } X)]$$

As this unification constraint is obviously not solvable by syntactical higher-order unification we cannot find a refutation on this derivation path.

As in calculus $\mathcal{R}$ the only way to find a refutation is to guess appropriate instances of the extensionality axioms and to derive from them clause $\mathcal{C}_{10}$ representing the Leibniz equation between $(\lambda X.\ \text{red } X \wedge \text{ball } X)$ and

$(\lambda X_{\cdot}\ \text{ball}\ X \wedge \text{red}\ X)$. A concrete derivation can be carried out analogously to the above derivation in $\mathcal{R}$. The only difference is that we employ resolution and factorisation instead of cut and simplification. In contrast to $\mathcal{R}$ we thereby gain additional guidance with respect to finding some of the required instantiations when combining the resolution/factorisation steps with eager unification attempts. But note that this only holds for the instantiation of non-formulas, e.g., as given in Step 3. The key step in the proof, namely the instantiation of the extensionality axioms in Step 1 with appropriate formulas as arguments, is not supported by unification. Instead the splitting rules have to be employed in order to guess the right instances. The problem with the splitting rules (or analogously the primitive substitution rule) is that each application introduces new clauses with flexible literals into the search space (in case of S^T_Π and S^F_Π even infinitely many) such that the splitting rules become recursively applicable to the new clauses as well.

Consequently, the extensionality treatment in $\mathcal{CR}$ is analogously to the one in $\mathcal{R}$ rather hard to guide in practice. Overwhelming the search space with extensionality clauses and applying forward reasoning to them furthermore principally contrasts the intended character of resolution based theorem proving.

Solution in $\mathcal{CRE}$. Analogous to the unsuccessful initial attempt in $\mathcal{CR}$ we first resolve between $\mathcal{C}_1$ and $\mathcal{C}_2$ and obtain

$$\mathcal{C}_3 : \Box \vee [p\ (\lambda X_{\cdot}\ \text{red}\ X \wedge \text{ball}\ X) \neq^{?} p\ (\lambda X_{\cdot}\ \text{ball}\ X \wedge \text{red}\ X)]$$

Whereas syntactical unification as employed in $\mathcal{CR}$ clashes on this unification constraint, calculus $\mathcal{CRE}$ can solve this E-unification problem provided that the employed E-unification algorithm covers associativity of the $\wedge$-operator (i.e., $E \models (\lambda X_o{\cdot}\ \lambda Y_o{\cdot}\ X \wedge Y) = (\lambda X_o{\cdot}\ \lambda Y_o{\cdot}\ Y \wedge X)$).

Hence, depending on the peculiarity of unification theory E calculus $\mathcal{CRE}$ can provide more goal directed solutions to particular examples and avoid applications of the extensionality axioms. However, the examples below will demonstrate that E-unification does not provide a general solution.

Solution in $\mathcal{ER}$. Calculus $\mathcal{ER}$ provides another goal directed solution avoiding the extensionality axioms. Instead of employing equational unification calculus $\mathcal{ER}$ analyses the unifiability of the unification constraint $\mathcal{C}_3$ with the help of a recursive call from within its unification algorithm to its own overall refutation process. Clearly, this idea can be seen as a very general form of equational unification, namely equational unification

modulo the theory defined by the given clause context and full higher-order logic.

Like above we initially resolve between $\mathcal{C}_1$ and $\mathcal{C}_2$ and obtain clause $\mathcal{C}_3$. Then we transform $\mathcal{C}_3$ with the unification rules Dec and Func into

$$\mathcal{C}_4 : \Box \vee [\text{red } s \wedge \text{ball } s \neq^{?} \text{ball } s \wedge \text{red } s]$$

and apply a recursive call to the overall refutation process with the Boolean extensionality rule Equiv. After normalisation and elimination of identical literals we thereby obtain the following trivially refutable set of propositional clauses

$$\mathcal{C}_5 : [\text{red } s]^F \vee [\text{ball } s]^F \qquad \mathcal{C}_6 : [\text{red } s]^T \qquad \mathcal{C}_7 : [\text{ball } s]^T$$

4.3. *Reasoning with Classical Logic*

The following theorem states that all unary logical operators $O_{o\to o}$ which map the propositions a and b to $\top$ consequently also map $a \wedge b$ to $\top$.

EXAMPLE 17. $\forall O_{o\to o}.\ (O\ a_o) \wedge (O\ b_o) \Rightarrow (O\ (a_o \wedge b_o))$.

Negation and normalisation leads to ($o_{o\to o}$ is a Skolem constant for O)

$$\mathcal{C}_1 : [o\ a]^T \qquad \mathcal{C}_2 : [o\ b]^T \qquad \mathcal{C}_3 : [o\ (a \wedge b)]^F$$

Solution in $\mathcal{R}$. Obviously there is no rule applicable to $\mathcal{C}_1$ – $\mathcal{C}_3$. As in Section 4.2 we are forced to appropriately instantiate the extensionality axioms. In particular we employ the following two instantiations of the Boolean extensionality principle $\text{EXT}^{\doteq}_o$:

$$(a \Leftrightarrow (a \wedge b)) \Leftrightarrow (a \doteq^o (a \wedge b))$$

and

$$(b \Leftrightarrow (a \wedge b)) \Leftrightarrow (b \doteq^o (a \wedge b))$$

That means we guess the substitutions $[a/A, (a \wedge b)/B]$, $[b/A, (a \wedge b)/B]$ and then instantiate the Boolean extensionality clauses $\mathcal{E}^o_1$ and $\mathcal{E}^o_2$ with rule Sub. From the instantiated clauses we can now derive

$$\mathcal{C}_4 : [P\ a]^F \vee [P\ (a \wedge b)]^T \vee [Q\ b]^F \vee [Q\ (a \wedge b)]^T$$

which represents that $(a \doteq (a \wedge b)) \vee (b \doteq (a \wedge b))$. By instantiating P and Q with o and simplification we obtain:

$$\mathcal{C}_5 : [o\ a]^F \vee [o\ b]^F \vee [o\ (a \wedge b)]^T$$

Resolving against $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ leads to $\square$.

Solution in $\mathcal{CR}$ and $\mathcal{CRE}$. There are only two possible proof steps at the very beginning: resolve between $\mathcal{C}_1$ and $\mathcal{C}_3$ and between $\mathcal{C}_2$ and $\mathcal{C}_3$. Thereby we get

$$\mathcal{C}_4 : \square \vee [p\ a \neq^?\ p\ (a \wedge b)] \quad \mathcal{C}_5 : \square \vee [p\ b \neq^?\ p\ (a \wedge b)]$$

Both unification constraints are neither solvable by syntactical higher-order unification nor by higher-order E-unification.

Successful refutations in $\mathcal{CR}$ and $\mathcal{CRE}$ therefore require the application of appropriately instantiated extensionality clauses as demonstrated within the refutation in calculus $\mathcal{R}$ above. Note that higher-order (E-)unification does not even provide any support for choosing the *right* instantiations of the extensionality axioms.

Hence both calculi, $\mathcal{CR}$ as well as $\mathcal{CRE}$, cannot be Henkin complete without additional extensionality axioms.

Solution in $\mathcal{ER}$. $\mathcal{ER}$ allows for a straightforward refutation of the clauses $\mathcal{C}_1 - \mathcal{C}_3$. Like in $\mathcal{CR}$ and $\mathcal{CRE}$ the only possible steps at the beginning are to resolve between $\mathcal{C}_1$ and $\mathcal{C}_3$ and between $\mathcal{C}_2$ and $\mathcal{C}_3$. Thereby we get

$$\mathcal{C}_4 : \square \vee [p\ a \neq^?\ p\ (a \wedge b)] \quad \mathcal{C}_5 : \square \vee [p\ b \neq^?\ p\ (a \wedge b)]$$

Decomposing both the unification constraints in both clauses leads to

$$\mathcal{C}_6 : \square \vee [a \neq^?\ (a \wedge b)] \quad \mathcal{C}_7 : \square \vee [b \neq^?\ (a \wedge b)]$$

When regarding both unification constraints isolated they are obviously neither syntactically nor semantically solvable. When considering them simultaneously, however, it is easy to see that at least one of both unification constraints must be solvable. Such a *non-constructive* reasoning on the simultaneous solvability/non-solvability of unification constraints is handled in $\mathcal{ER}$ by recursive calls from unification to the overall proof search. In this sense $\mathcal{ER}$ intuitively first assumes that the unification constraints are simultaneously not solvable and then tries to refute this assumption. More concretely, the recursive calls with rule Equiv applied

to $\mathcal{C}_6$ and $\mathcal{C}_7$ introduce after normalisation and factorisation the following clauses into the search space (note the importance of the fact that the generated clauses are analysed in a common context):

$$\mathcal{C}_5 : [a]^F \vee [b]^F \quad \mathcal{C}_6 : [a]^T \vee [b]^T \quad \mathcal{C}_7 : [a]^T \quad \mathcal{C}_8 : [b]^T$$

Clauses $\mathcal{C}_5$–$\mathcal{C}_8$ can be refuted immediately, which contradicts the assumption of the simultaneous semantical non-unifiability of the unification constraints in $\mathcal{C}_6$ and $\mathcal{C}_7$. Hence, either $\mathcal{C}_6$ or $\mathcal{C}_7$ must already be the empty clause, which justifies the proof.

4.4. *Mappings from Booleans to Booleans*

We already mentioned in Section 2.3 that in Henkin semantics the domain $\mathcal{D}_o$ of all Booleans contains exactly the truth values $\bot$ and $\top$. Consequently the domain of all mappings from Booleans to Booleans contains exactly[5] the denotations of the following four functions: $\lambda X_o.\, X_o$, $\lambda X_o.\, \neg X_o$, $\lambda X_o.\, \bot$, and $\lambda X_o.\, \top$. This theorem can be formulated as follows (where $f_{o\to o}$ is a constant):

$$(f = \lambda X_o.\, X_o) \vee (f = \lambda X_o.\, \neg X_o) \vee (f = \lambda X_o.\, \bot) \vee (f = \lambda X_o.\, \top)$$

By unfolding the definition of Leibniz equality, negating the theorem, and applying clause normalisation we obtain the following clauses (where $p^1, \ldots, p^4$ are Skolem constants):

$$\mathcal{D}_1 : [p^1\ f]^T \quad \mathcal{D}_2 : [p^1\ \lambda X_o.\, X_o]^F \quad \mathcal{D}_3 : [p^2\ f]^T \quad \mathcal{D}_4 : [p^2\ \lambda X_o.\, \neg X_o]^F$$

$$\mathcal{D}_5 : [p^3\ f]^T \quad \mathcal{D}_6 : [p^3\ \lambda X_o.\, \bot]^F \quad \mathcal{D}_7 : [p^4\ f]^T \quad \mathcal{D}_8 : [p^4\ \lambda X_o.\, \top]^F$$

Solution in $\mathcal{R}$, $\mathcal{CR}$, and $\mathcal{CRE}$. As the reader may easily check, none of the applicable resolution steps leads to a unification constraint that is solvable by higher-order unification or higher-order E-unification (independent from theory E).

In order to find a refutation appropriate instances of the extensionality principles are needed, just as illustrated in the previous example. Because of lack of space we do not present the quite lengthy refutation here.

Solution in $\mathcal{ER}$. In $\mathcal{ER}$ we can find the following goal directed refutation of the clauses $\mathcal{D}_1, \ldots, \mathcal{D}_8$. We first resolve between the related clauses $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$, $\mathcal{D}_5$ and $\mathcal{D}_6$, and $\mathcal{D}_7$ and $\mathcal{D}_8$, and immediately

decompose the head symbols in the unification pairs. Thereby we obtain the following four clauses consisting of exactly one unification constraint.

$$\mathcal{C}_1 : [p = \lambda x.\, x]^F \quad \mathcal{C}_2 : [p = \lambda x.\, \neg x]^F \quad \mathcal{C}_3 : [p = \lambda x.\, \bot]^F$$

$$\mathcal{C}_4 : [p = \lambda x.\, \top]^F$$

Whereas none of these unification constraints is solvable taken alone (even not by E-unification), it is possible in calculus $\mathcal{ER}$ to refute the assumption that these unification constraints are simultaneously not solvable. Like in the previous example the idea of the following derivation is to show that always one of these unification constraints must be solvable even though one cannot specify which one. The proof presented here has been automatically generated by the prototypical higher-order theorem prover LEO (Benzmüller and Kohlhase 1998b) (which implements calculus $\mathcal{ER}$) within 25 seconds on a Pentium II with 400MHz. Each line presented below introduces a new clause (the line numbering thereby corresponds to the clause numbering) by applying the specified calculus rules to previously derived clauses. For instance, line 32 describes that clause $\mathcal{C}_{32}$ is derived from clauses $\mathcal{C}_{17}$ and $\mathcal{C}_{16}$ by resolution with rule Res and immediate elimination of trivial unification constraints with rule Triv. In the proof below $s^1, \ldots, s^4$ are new Skolem constants of Boolean type introduced by the functional extensionality rule Func at the very beginning of the refutation.

5 :	Func($\mathcal{C}_4$)	$\mathcal{C}_5 : [(p\ s^3) = \top]^F$
6 :	Func($\mathcal{C}_3$)	$\mathcal{C}_6 : [(p\ s^2) = \bot]^F$
7 :	Func($\mathcal{C}_2$)	$\mathcal{C}_7 : [(p\ s^4) = (\neg\ s^4)]^F$
8 :	Func($\mathcal{C}_1$)	$\mathcal{C}_8 : [(p\ s^1) = s^1)]^F$
10 :	Equiv+Cnf($\mathcal{C}_5$)	$\mathcal{C}_{10} : [(p\ s^3)]^F$
13 :	Equiv+Cnf($\mathcal{C}_6$)	$\mathcal{C}_{13} : [(p\ s^2)]^T$
16 :	Equiv+Cnf($\mathcal{C}_7$)	$\mathcal{C}_{16} : [s^4]^T \vee [(p\ s^4)]^F$
17 :	Equiv+Cnf($\mathcal{C}_7$)	$\mathcal{C}_{17} : [(p\ s^4)]^T \vee [s^4]^F$
20 :	Equiv+Cnf($\mathcal{C}_8$)	$\mathcal{C}_{20} : [(p\ s^1)]^F \vee [s^1]^F$
21 :	Equiv+Cnf($\mathcal{C}_8$)	$\mathcal{C}_{21} : [s^1]^T \vee [(p\ s^1)]^T$
32 :	Res+Triv($\mathcal{C}_{17}$; $\mathcal{C}_{16}$)	$\mathcal{C}_{32} : [(p\ s^4)]^T \vee [(p\ s^4)]^F$
36 :	Res($\mathcal{C}_{20}$; $\mathcal{C}_{17}$)	$\mathcal{C}_{36} : [s^4]^F \vee [s^1]^F \vee [(p\ s^1) = (p\ s^4)]^F$
42 :	Dec($\mathcal{C}_{36}$)	$\mathcal{C}_{42} : [s^1]^F \vee [s^4]^F \vee [s^1 = s^4]^F$
56 :	Equiv+Cnf($\mathcal{C}_{42}$)	$\mathcal{C}_{56} : [s^1]^F \vee [s^4]^F$

76 :	Res($\mathcal{C}_{32}$; $\mathcal{C}_{21}$)	$\mathcal{C}_{76} : [s^1]^T \vee [(p\ s^4)]^T \vee [(p\ s^4) = (p\ s^1)]^F$
85 :	Dec($\mathcal{C}_{76}$)	$\mathcal{C}_{85} : [(p\ s^4)]^T \vee [s^1]^T \vee [s^4 = s^1]^F$
134 :	Equiv+Cnf($\mathcal{C}_{85}$)	$\mathcal{C}_{134} : [(p\ s^4)]^T \vee [s^1]^T \vee [s^4]^T$
141 :	Res+Triv($\mathcal{C}_{56}$; $\mathcal{C}_{16}$)	$\mathcal{C}_{141} : [(p\ s^4)]^F \vee [s^1]^F$
144 :	Res+Triv($\mathcal{C}_{56}$; $\mathcal{C}_{21}$)	$\mathcal{C}_{144} : [(p\ s^1)]^T \vee [s^4]^F$
163 :	Res+Triv($\mathcal{C}_{141}$; $\mathcal{C}_{21}$)	$\mathcal{C}_{163} : [(p\ s^1)]^T \vee [(p\ s^4)]^F$
211 :	Res($\mathcal{C}_{163}$; $\mathcal{C}_{13}$)	$\mathcal{C}_{211} : [(p\ s^1)]^T \vee [(p\ s^4) = (p\ s^2)]^F$
237 :	Dec($\mathcal{C}_{211}$)	$\mathcal{C}_{237} : [(p\ s^1)]^T \vee [s^4 = s^2]^F$
250 :	Res+Triv($\mathcal{C}_{134}$; $\mathcal{C}_{16}$)	$\mathcal{C}_{250} : [s^4]^T \vee [s^1]^T$
255 :	Res+Triv($\mathcal{C}_{134}$; $\mathcal{C}_{17}$)	$\mathcal{C}_{255} : [s^1]^T \vee [(p\ s^4)]^T$
387 :	Res+Triv($\mathcal{C}_{255}$; $\mathcal{C}_{20}$)	$\mathcal{C}_{387} : [(p\ s^4)]^T \vee [(p\ s^1)]^F$
458 :	Res($\mathcal{C}_{387}$; $\mathcal{C}_{10}$)	$\mathcal{C}_{458} : [(p\ s^1)]^F \vee [(p\ s^4) = (p\ s^3)]^F$
459 :	Res($\mathcal{C}_{387}$; $\mathcal{C}_{13}$)	$\mathcal{C}_{459} : [(p\ s^4)]^T \vee [(p\ s^1) = (p\ s^2)]^F$
492 :	Dec($\mathcal{C}_{458}$)	$\mathcal{C}_{492} : [(p\ s^1)]^F \vee [s^4 = s^3]^F$
493 :	Dec($\mathcal{C}_{459}$)	$\mathcal{C}_{493} : [(p\ s^4)]^T \vee [s^1 = s^2]^F$
519 :	Equiv+Cnf($\mathcal{C}_{493}$)	$\mathcal{C}_{519} : [(p\ s^4)]^T \vee [s^1]^F \vee [s^2]^F$
523 :	Equiv+Cnf($\mathcal{C}_{492}$)	$\mathcal{C}_{523} : [(p\ s^1)]^F \vee [s^4]^F \vee [s^3]^F$
558 :	Res+Triv($\mathcal{C}_{519}$; $\mathcal{C}_{141}$)	$\mathcal{C}_{558} : [s^2]^F \vee [s^1]^F$
592 :	Res+Triv($\mathcal{C}_{558}$; $\mathcal{C}_{21}$)	$\mathcal{C}_{592} : [(p\ s^1)]^T \vee [s^2]^F$
610 :	Res+Triv($\mathcal{C}_{558}$; $\mathcal{C}_{250}$)	$\mathcal{C}_{610} : [s^4]^T \vee [s^2]^F$
664 :	Res($\mathcal{C}_{592}$; $\mathcal{C}_{10}$)	$\mathcal{C}_{664} : [s^2]^F \vee [(p\ s^1) = (p\ s^3)]^F$
706 :	Dec($\mathcal{C}_{664}$)	$\mathcal{C}_{706} : [s^2]^F \vee [s^1 = s^3]^F$
783 :	Res+Triv($\mathcal{C}_{523}$; $\mathcal{C}_{144}$)	$\mathcal{C}_{783} : [s^3]^F \vee [s^4]^F$
820 :	Res+Triv($\mathcal{C}_{783}$; $\mathcal{C}_{610}$)	$\mathcal{C}_{820} : [s^2]^F \vee [s^3]^F$
824 :	Res+Triv($\mathcal{C}_{783}$; $\mathcal{C}_{16}$)	$\mathcal{C}_{824} : [(p\ s^4)]^F \vee [s^3]^F$
912 :	Res($\mathcal{C}_{824}$; $\mathcal{C}_{13}$)	$\mathcal{C}_{912} : [s^3]^F \vee [(p\ s^4) = (p\ s^2)]^F$
952 :	Dec($\mathcal{C}_{912}$)	$\mathcal{C}_{952} : [s^3]^F \vee [s^4 = s^2]^F$
1078 :	Equiv+Cnf($\mathcal{C}_{952}$)	$\mathcal{C}_{1078} : [s^2]^T \vee [s^4]^T \vee [s^3]^F$
1144 :	Res+Triv($\mathcal{C}_{1078}$; $\mathcal{C}_{783}$)	$\mathcal{C}_{1144} : [s^2]^T \vee [s^3]^F$
1218 :	Res+Triv($\mathcal{C}_{1144}$; $\mathcal{C}_{820}$)	$\mathcal{C}_{1218} : [s^3]^F$
1302 :	Equiv+Cnf($\mathcal{C}_{706}$)	$\mathcal{C}_{1302} : [s^3]^T \vee [s^1]^T \vee [s^2]^F$

1363 :	Res+Triv($\mathcal{C}_{1302}$; $\mathcal{C}_{558}$)	$\mathcal{C}_{1363} : [s^3]^T \vee [s^2]^F$
1377 :	Res+Triv($\mathcal{C}_{1363}$; $\mathcal{C}_{1218}$)	$\mathcal{C}_{1377} : [s^2]^F$
1454 :	Equiv+Cnf($\mathcal{C}_{237}$)	$\mathcal{C}_{1454} : [(p\ s^1)]^T \vee [s^2]^T \vee [s^4]^T$
1502 :	Res+Triv($\mathcal{C}_{1454}$; $\mathcal{C}_{144}$)	$\mathcal{C}_{1502} : [s^2]^T \vee [(p\ s^1)]^T$
1521 :	Res+Triv($\mathcal{C}_{1502}$; $\mathcal{C}_{1377}$)	$\mathcal{C}_{1521} : [(p\ s^1)]^T$
1560 :	Res($\mathcal{C}_{1521}$; $\mathcal{C}_{10}$)	$\mathcal{C}_{1560} : [(p\ s^1) = (p\ s^3)]^F$
1565 :	Res+Triv($\mathcal{C}_{1521}$; $\mathcal{C}_{20}$)	$\mathcal{C}_{1565} : [s^1]^F$
1576 :	Dec($\mathcal{C}_{1560}$)	$\mathcal{C}_{1576} : [s^1 = s^3]^F$
1643 :	Equiv+Cnf($\mathcal{C}_{1576}$)	$\mathcal{C}_{1643} : [s^3]^T \vee [s^1]^T$
1646 :	Res+Triv($\mathcal{C}_{1643}$; $\mathcal{C}_{1218}$)	$\mathcal{C}_{1646} : [s^1]^T$
1655 :	Res+Triv($\mathcal{C}_{1646}$; $\mathcal{C}_{1565}$)	$\mathcal{C}_{1655} : \square$

4.5. *Additional Examples and Case Studies*

Benzmüller (1999a) discusses several additional examples that require full extensionality reasoning – such as the following example on sets:

$$\wp(\emptyset) = \{\emptyset\}$$

It furthermore reports on case studies with the higher-order theorem prover LEO (Benzmüller and Kohlhase 1998) that demonstrate the feasibility of calculus $\mathcal{ER}$ in practice.

5. RELATED WORK

Related to calculus $\mathcal{CR}$ is the higher-order resolution approach of Jensen and Pietrzykowski (1972, 1976) which also employs a higher-order unification algorithm in order to guide the proof search. The undecidability problem of higher-order unification is thereby tackled by *dove-tailing* the generation of resolvents. Like $\mathcal{CR}$ this approach requires the extensionality axioms in the search space to ensure Henkin completeness.

Kohlhase (1994) presents a sorted variant of Huet's constrained resolution approach. Kohlhase (1995) discusses a higher-order tableaux calculus that is quite closely related to calculus $\mathcal{ER}$, as it already introduces additional calculus rules in order to improve its extensionality treatment. As is illustrated in detail in Benzmüller (1999a) the presented extensionality rules are unfortunately not sufficient to completely avoid additional extensionality axioms. The first sufficient set of extensionality rules in this sense

is presented in Benzmüller (1997), which introduces a variant of calculus $\mathcal{ER}$ as presented here.

The *theorem proving modulo* approach described in Dowek et al. (1998) is a way to remove computational arguments from proofs by reasoning modulo a congruence on propositions that is handled via rewrite rules and equations. In their paper the authors present a higher-order logic as a theory modulo.

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 rarely been addressed in literature so far. Approaches to integrate primitive equality in a Henkin complete higher-order theorem proving approach are discussed in Snyder and Lynch (1991), Benzmüller (1999a, b). Of course, the field of higher-order term rewriting and narrowing (Prehofer 1998; Nipkow and Prehofer 1998; Nipkow 1995) is very active. But calculi developed in this context typically only address functional extensionality and do not focus on the subtle interplay between functional and Boolean extensionality that is required in a Henkin complete theorem proving approach.

The most powerful automated higher-order theorem prover currently available is (to the best knowledge of the author) the TPS-system (Andrews 1996) which employs the mating method (Andrews 1976) as inference mechanism. TPS employs a clever extensionality pre-processing mechanism which transforms embedded equations in input formulas into more appropriate ones in order to avoid later applications of the extensionality axioms. However, this does not provide a general solution and many theorems requiring non-trivial extensionality reasoning, such as Examples 3.4 and 4.4, cannot be proven this way.

6. CONCLUSION

In this paper we investigated four approaches to resolution based higher-order theorem proving: Andrews' higher-order resolution approach $\mathcal{R}$, Huet's constrained resolution approach $\mathcal{CR}$, higher-order E-resolution $\mathcal{CRE}$, and extensional higher-order resolution $\mathcal{ER}$. Thereby we focused on the extensionality treatment of these approaches and pointed to the crucial role of full extensionality for ensuring Henkin completeness. The investigated examples demonstrate that simply adding (infinitely many) extensionality axioms to the search space – as suggested for $\mathcal{R}$ and $\mathcal{CR}$ – increases the amount of blind search and is thus rather infeasible in practice.

Whereas higher-order E-unification and E-resolution indeed improves the situation in particular contexts, it does still not provide a general solution.

Calculus $\mathcal{ER}$ is the sole studied approach that can completely avoid the extensionality axioms. It's extensionality treatment is based on goal directed extensionality rules which closely connect the overall refutation search with unification by allowing for mutual recursive calls. This suitably extends the higher-order E-unification and E-resolution idea, as it turns the unification mechanism into a most general, dynamic theory unification mechanism. Unification may now itself employ a Henkin complete higher-order theorem prover as a subordinated reasoning system and the considered theory (which is defined by the sum of all clauses in the actual search space) dynamically changes. Due to the close connection of unification and refutation search it is even possible in $\mathcal{ER}$ to realise a kind of *non-constructive* reasoning on E-unifiability, as was demonstrated in this paper.

ACKNOWLEDGEMENTS

I want to thank Volker Sorge and the anonymous referee of this paper for their useful comments and contributions. I am also grateful to Michael Kohlhase for many fruitful discussions on extensional higher-order theorem proving.

NOTES

1. Conjunction elimination is provided by the rules $\vee_l^F$ and $\vee_r^F$. We note that conjunction is defined with the help of disjunction and negation; cf. Section 2.1.
2. Existential elimination is realised by the rule Π^F. For this note that existential quantification is defined with the help of universal quantification (and universal quantification with the help of Π); cf. Section 2.1.
3. It is still an open problem whether it is possible to restrict the required instances of the functional extensionality axioms in dependence of a given proof problem.
4. One may choose a bound on the allowed number of nested branchings in the search tree with rule FlexRigid.
5. Since $\mathcal{D}_o$ contains two elements, $\mathcal{D}_{o\to o}$ contains in each Henkin model at most four elements. And because of the requirement, that the function domains in Henkin models must be rich enough such that every term has a denotation, it follows that $\mathcal{D}_{o\to o}$ contains exactly the pairwise distinct denotations of the four presented function terms.

REFERENCES

Andrews, P, B.: 1971, 'Resolution in Type Theory', *Journal of Symbolic Logic* **36**, 414–432.

Andrews, P. B.: 1972, 'General Models and Extensionality', *Journal of Symbolic Logic* **37**, 395–397.

Andrews, P. B.: 1973, *Letter to Roger Hindley* dated January 22.

Andrews, P. B.: 'Refutations by Matings', *IEEE Transactions on Computers* **C-25**, 801–807.

Andrews, P. B.: 1989, 'On Connections and Higher Order Logic', *Journal of Automated Reasoning* **5**, 257–291.

Andrews, P. B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., and Xi, H.: 1996, 'TPS: A Theorem Proving System for Classical Type Theory', *Journal of Automated Reasoning* **16**, 321–353.

Barendregt, H. P.: 1984, *The Lambda Calculus – Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics 103, Amsterdam.

Benzmüller, C.: 1997, *A calculus and a System Architecture for Extensional Higher-order Resolution*, Research Report 97-198, Department of Mathematical Sciences, Carnegie Mellon University.

Benzmüller, C.: 1999a, *Equality and Extensionality in Automated Higher-Order Theorem Proving*, Ph.D. thesis, Technische Fakultät, Universität des Saarlandes.

Benzmüller, C.: 1999b, in H. Ganzinger (ed.), *Proceedings of the 16th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 1632, pp. 399–413, Springer.

Benzmüller, C. and Kohlhase, M.: 1997, 'Model Existence for Higher-Order Logic', SEKI-Report SR-97-09, Fachbereich Informatik, Universität des Saarlandes.

Benzmüller, C. and Kohlhase, M.: 1998a, 'Extensional Higher-order Resolution', in Kirchner and Kirchner (eds.), *Proceedings of the 15th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 1421, pp.56–72, Springer.

Benzmüller, C. and Kohlhase, M. 1998b, 'LEO – A Higher-order Theorem Prover, in Kirchner and Kirchner (eds.), *Proceedings of the 15th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 1421, pp. 139–144, Springer.

Baader, F. and Siekmann, J.: 1994, 'Unification Theory', in D. M. Gabbay, C. J. Hogger, J. A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2: Deduction Methodologies*, Oxford, Chapter 2.2

Church, A.: 1940, 'A formulation of the Simple Theory of Types', *Journal of Symbolic Logic* **5**, 56–68.

Dowek, G., Hardin, T., and Kirchner, C.: 1998, *Theorem Proving Modulo, Rapport de Recherche 3400*, Institut National de Recherche en Informatique et en Automatique.

Dougherty, D. and Johann, P.: 1992, 'A Combinatory Logic Approach to Higher-order E-unification', in D. Kapur (ed.), *Proceedings of the 11th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 607, pp. 79–93, Springer.

Goldfarb, W. D. 1981, 'The Undecidability of the Second-order Unification Problem', *Theoretical Computer Science* **13**, 225–230.

Henkin, L.: 1950, 'Completeness in the Theory of Types', *Journal of Symbolic Logic* **15**, 81–91.

Huet, G. P.: 1972, *Constrained Resolution: A Complete Method for Higher Order Logic*, Ph.D. thesis, Case Western Reserve University.

Huet, G. P.: 1973, 'A Mechanization of Type Theory', in D. E. Walker and L. Norton (eds.), *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pp. 139–146.

Huet, G. P.: 1973, 'The Undecidability of Unification in Third Order Logic', *Information and Control* **22**, 257–267.

Huet, G. P.: 1975, 'A Unification Algorithm for Typed λ-calculus', *Theoretical Computer Science* **1**, 27–57.

Jensen, D. C. and Pietrzykowski, T.: 1972, 'A Complete Mechanization of ω-order Type Theory', in *Proceedings of the ACM annual Conference*, volume 1, 89–92.

Jensen, D. C. and Pietrzykowski, T.: 1976, 'Mechanizing ω-order Type Theory through Unification', *Theoretical Computer Science* **3**, 123–171.

Kohlhase, M.: 1994, *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*, Ph.D. thesis, Fachbereich Informatik, Universität des Saarlandes.

Kohlhase, M.: 1995, 'Higher-Order Tableaux', in P. Baumgartner, R. Hähnle, and J. Posegga (eds.), *Theorem Proving with Analytic Tableaux and Related Methods*, Lecture Notes in Artificial Intelligence 918, pp. 294–309, Springer.

Lucchesi, C. L.: 1972, *The Undecidability of the Unification Problem for Third Order Languages*, Report CSRR 2059, University of Waterloo, Waterloo, Canada.

Miller, D.: 1983, *Proofs in Higher-Order Logic*, Ph.D. thesis, Carnegie Mellon University.

Nipkow, T.: 1995, 'Higher-order Rewrite Systems', in J. Hsiang (ed.), *Rewriting Techniques and Applications*, *6th International Conference*, Lecture Notes in Computer Science 914, Springer.

Nipkow, T. and Prehofer, C.: 1998, 'Higher-order Rewriting and Equational Reasoning', in W. Bibel and P. Schmitt (eds.), *Automated Deduction – A Basis for Applications*, Dordrecht, Applied Logic Series, pp. 399–430.

Nipkow, T. and Qian, Z,: 1991, 'Modular Higher-order E-unification', in R. V. Book (ed.), *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Artificial Intelligence 488, pp. 200–214, Springer.

Prehofer, C.: 1998, *Solving Higher-Order Equations: From Logic to Programming*, Progress in theoretical computer science, Birkhäuser.

Qian, Z. and Wang K.: 1996, 'Modular Higher-order Equational Preunification', *Journal of Symbolic Computation* **22**, 401–424.

Robinson, J. A.: 1965, 'A Machine-oriented Logic Based on the Resolution Principle', *Journal of the Association for Computing Machinery* **12**, 23–41.

Siekmann, J. H.: 1989, 'Unification Theory', *Journal of Symbolic Computation* **7**, 207–274.

Smullyan, R. M. 1963, 'A Unifying Principle for Quantification Theory', *Proceedings of the National Acadamy of Sciences, USA 49*, pp. 828–832.

Snyder, W.: 1990, 'Higher Order E-unification', in M. Stickel (ed.), *Proceedings of the 10th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 449, pp. 573–578, Springer.

Snyder, W. and Gallier, J.: 1989, 'Higher-order Unification Revisited: Complete Sets of Transformations', *Journal of Symbolic Computation* **8**, 101–140.

Snyder, W. and Lynch, C.: 1991, 'Goal-directed Strategies for Paramodulation', in R. V. Book (ed.), *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Artificial Intelligence 488, pp. 200–214, Springer.

Wolfram, D. A.: 1993, *The Clausal Theory of Types*, Cambridge, Cambridge Tracts in Theoretical Computer Science 21.

Fachbereich Informatik,
Universität des Saarlandes
D-66041 Saarbrücken
Germany
E-mail: chris@ags.uni-sb.de

Extensional Higher-Order Paramodulation and RUE-Resolution

Christoph Benzmüller

Fachbereich Informatik, Universität des Saarlandes
chris@ags.uni-sb.de

Abstract. This paper presents two approaches to primitive equality treatment in higher-order (HO) automated theorem proving: a calculus $\mathcal{EP}$ adapting traditional first-order (FO) paramodulation [RW69] , and a calculus $\mathcal{ERUE}$ adapting FO RUE-Resolution [Dig79] to classical type theory, i.e., HO logic based on Church's simply typed λ-calculus. $\mathcal{EP}$ and $\mathcal{ERUE}$ extend the extensional HO resolution approach $\mathcal{ER}$ [BK98a]. In order to reach Henkin completeness without the need for additional extensionality axioms both calculi employ new, positive extensionality rules analogously to the respective negative ones provided by $\mathcal{ER}$ that operate on unification constraints. As the extensionality rules have an intrinsic and unavoidable difference-reducing character the HO paramodulation approach loses its pure term-rewriting character. On the other hand examples demonstrate that the extensionality rules harmonise quite well with the difference-reducing HO RUE-resolution idea.

1 Introduction

Higher-Order (HO) Theorem Proving based on the resolution method has been first examined by Andrews [And71] and Huet [Hue72]. Whereas the former avoids unification the latter generally delays the computation of unifiers and instead adds unification constraints to the clauses in order to tackle the undecidability problem of HO unification. More recent papers concentrate on the adaption of sorts [Koh94] or theory unification [Wol93] to HO logic. Common to all these approaches is that they do not sufficiently solve the extensionality problem in HO automated theorem proving, i.e., all these approaches require the extensionality axioms to be added into the search space in order to reach Henkin completeness (which is the most general notion of semantics that allows complete calculi [Hen50]). This leads to a search space explosion that is awkward to manage in practice. A solution to the problem is provided by the extensional HO resolution calculus $\mathcal{ER}$ [BK98a]. This approach avoids the extensionality axioms and instead extends the syntactical (pre-)unification process by additional extensionality rules. These new rules allow for recursive calls during the (pre-) unification process to the overall refutation search whenever pure syntactical HO unification is too weak to show that two terms can be equalised modulo the extensionality principles. $\mathcal{ER}$ has been implemented in LEO [BK98b] and case studies have demonstrated its suitability, especially for reasoning about sets.

There are many possibilities to improve the extensional HO resolution approach and the probably most promising one concerns the treatment of equality. $\mathcal{ER}$ assumes that equality is defined by the Leibniz principle (two things

H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 399–413, 1999.

are equal iff they have the same properties) or by any other valid definition principle, and thus provides no support for primitive equality. But a primitive equality treatment seems to be more appropriate as it avoids the many flexible literals introduced when using defined equality, which unfortunately increase the amount of blind search with $\mathcal{ER}$'s primitive substitution rule *Prim*. Therefore we adapt two well known first-order (FO) approaches to primitive equality: the paramodulation approach [RW69] (the basis of many successful refinements such as the superposition approach) and the RUE-resolution approach [Dig79] (a generalisation of *E*-resolution [Dar68]). The main goal thereby is to preserve Henkin completeness. We will show that therefore positive extensionality rules are needed (which operate on positive equation literals) as in contrast to FO logic single positive equations can be contradictory by themselves in HO logic.[1]

This paper summarises the Chapt. 6, 7, and 8 of [Ben99] and because of lack of space the preliminaries and the formal proofs can only be sketched here.

The preliminaries are concisely presented in Sect. 2 and calculus $\mathcal{ER}$ is reviewed in 3. Section 4 discusses interesting aspects on primitive and defined equality, before the extensional HO paramodulation calculus $\mathcal{EP}$ and the extensional HO RUE-resolution approach $\mathcal{ERUE}$ are discussed in 5 and 6. Both approaches are briefly compared by examples in 7 and the conclusion is presented in 8.

2 Higher-Order (HO) Logic

We consider a HO logic based on Church's simply typed λ-calculus [Chu40] and choose $BT := \{\iota, o\}$ as *base types*, where ι denotes the set of individuals and o the set of truth values. *Functional types* are inductively defined over BT. A *signature* Σ ($\Sigma^{=}$) contains for each type an infinite set of variables and constants and provides the *logical connectives* $\neg_{o\to o}$, $\vee_{o\to o\to o}$, and $\Pi_{(\alpha\to o)\to o}$ (additionally $=^{\alpha} := =_{\alpha\to\alpha\to o}$) for every type α. The set of all Σ-terms (closed Σ-terms) of type α is denoted by wff_{α} ($cwff_{\alpha}$). Variables are printed as upper-case (e.g. X_{α}), constants as lower-case letters (e.g. c_{α}) and arbitrary terms appear as bold capital letters (e.g. $\mathbf{T}_{\alpha}$). If the type of a symbol is uniquely determined by the given context we omit it. We abbreviate function applications by $h_{\alpha_1\to\cdots\to\alpha_n\to\beta}\ \overline{\mathbf{U}^n_{\alpha_n}}$, which stands for $(\cdots(h_{\alpha_1\to\cdots\to\alpha_n\to\beta}\ \mathbf{U}^1_{\alpha_1})\cdots\mathbf{U}^n_{\alpha_n})$. For α-, β-, η-, $\beta\eta$-*conversion* and the definition of $\beta\eta$- and *head-normal form* (*hnf*) for a term $\mathbf{T}$ we refer to [Bar84] as well as for the definition of *free* variables, *closed* formulas, and *substitutions*. *Unification* and sets of *partial bindings* $\mathcal{AB}^h_{\gamma}$ are well explained in [SG89]. An example for a *pre-clause*, i.e., not in proper clause normal form, consisting of a positive literal, a negative literal, and a special negative equation literal (also called *unification constraint*) is $\mathcal{C} : [\neg(P_{\iota\to o}\ \mathbf{T}_{\iota})]^T \vee [h_{\overline{\gamma}\to o}\ \overline{Y^n_{\gamma_n}}]^F \vee [Q_{\iota\to\iota}\ a_{\iota} = Y_{\iota\to\iota}\ b_{\iota}]^F$. The corresponding *proper clause*, i.e., properly normalised, is $\mathcal{C}' : [P\ \mathbf{T}]^F \vee [h\ \overline{Y^n}]^F \vee [Q\ a = Y\ b]^F$. The unification constraint in $\mathcal{C}$ and $\mathcal{C}'$ is called a *flex-flex* pair as both unification terms have *flexible* heads. A clause is called *empty*, denoted by $\square$, if it consists

[1] Consider, e.g. the positive literal $[a_o = \neg a_o]^T$ or $[G\ X = f]^T$ (resulting from the following formulation of Cantor's theorem: $\neg\exists G_{\iota\to\iota\to o}.\ \forall P_{\iota\to o}.\ \exists X_{\iota}.\ G\ X = P$).

only of *flex-flex* unification constraints. A clause $\mathcal{C}_1$ generalises a clause $\mathcal{C}_2$, iff there is a substitution σ, such that the $\beta\eta$-normal form of $\sigma(\mathcal{C}_1)$ is an α-variant of the $\beta\eta$-normal form of $\mathcal{C}_2$.

A calculus R provides a set of rules $\{r_n |\ 0 < n \leq i\}$ defined on clauses. We write $\Phi \vdash^{r_n} \mathcal{C}$ ($\mathcal{C}' \vdash^{r_n} \mathcal{C}$) iff clause $\mathcal{C}$ is the result of an one step application of rule $r_n \in R$ to premise clauses $\mathcal{C}'_i \in \Phi$ (to $\mathcal{C}'$ respectively). Multiple step derivations in calculus R are abbreviated by $\Phi_1 \vdash_R \Phi_k$ (or $\mathcal{C}_1 \vdash_R \mathcal{C}_k$).

A *standard model* for $\mathcal{HOL}$ provides a fixed set $\mathcal{D}_\iota$ of individuals, and a set $\mathcal{D}_o := \{\top, \bot\}$ of truth values. The domains for functional types are defined inductively: $\mathcal{D}_{\alpha\to\beta}$ is the set of all functions $f: \mathcal{D}_\alpha \to \mathcal{D}_\beta$. *Henkin models* only require that $\mathcal{D}_{\alpha\to\beta}$ has enough members that any well-formed formula can be evaluated. Thus, the generalisation to Henkin models restricts the set of valid formulae sufficiently, such that complete calculi are possible. In Henkin and standard semantics Leibniz equality ($\doteq^\alpha := \lambda X_\alpha.\ \lambda Y_\alpha.\ \forall P_{\alpha\to o}.\ PX \Rightarrow PY$) denotes the intuitive equality relation and the functional extensionality principles ($\forall M_{\alpha\to\beta}.\ \forall N_{\alpha\to\beta}.\ (\forall X.\ (MX) = (NX)) \Leftrightarrow (M = N)$) as well as the Boolean extensionality principle ($\forall P_o.\ \forall Q_o.\ (P = Q) \Leftrightarrow (P \Leftrightarrow Q)$) are valid (see [Ben99,BK97]). *Satisfiability* and *validity* ($\mathcal{M} \models \mathbf{F}$ or $\mathcal{M} \models \Phi$) of a formula $\mathbf{F}$ or set of formulas Φ in a model $\mathcal{M}$ is defined as usual.

The completeness proofs employ the abstract consistency method of [BK97]& [Ben99] which extends Andrews' HO adaptation [And71] of Smullyan's approach [Smu63] to Henkin semantics. Here we only mention the two main aspects:

Definition 1 (*Acc* for Henkin Models). *Let Σ be a signature and Γ_Σ a class of sets of Σ-sentences. If the following conditions (all but ∇_e^*) hold for all $\mathbf{A}, \mathbf{B} \in cwff_o$, $\mathbf{F}, \mathbf{G} \in cwff_{\alpha\to\beta}$, and $\Phi \in \Gamma_\Sigma$, then we call Γ_Σ an* **abstract consistency class for Henkin models with primitive equality***, abbreviated by $Acc^=$ (resp.* **abstract consistency class for Henkin models***, abbreviated by Acc).*

Saturated $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\neg\mathbf{A}\} \in \Gamma_\Sigma$.
∇_c *If* $\mathbf{A}$ *is atomic, then* $\mathbf{A} \notin \Phi$ *or* $\neg\mathbf{A} \notin \Phi$.
$\nabla_\neg$ *If* $\neg\neg\mathbf{A} \in \Phi$, *then* $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$.
∇_f *If* $\mathbf{A} \in \Phi$ *and* $\mathbf{B}$ *is the* $\beta\eta$*-normal form of* $\mathbf{A}$, *then* $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\vee$ *If* $\mathbf{A} \vee \mathbf{B} \in \Phi$, *then* $\Phi \cup \{\mathbf{A}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\wedge$ *If* $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, *then* $\Phi \cup \{\neg\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$.
$\nabla_\forall$ *If* $\Pi^\alpha\mathbf{F} \in \Phi$, *then* $\Phi \cup \{\mathbf{F}\ \mathbf{W}\} \in \Gamma_\Sigma$ *for each* $\mathbf{W} \in cwff_\alpha$.
$\nabla_\exists$ *If* $\neg\Pi^\alpha\mathbf{F} \in \Phi$, *then* $\Phi \cup \{\neg(\mathbf{F}\ w)\} \in \Gamma_\Sigma$ *for any new constant* $w \in \Sigma_\alpha$.
∇_b *If* $\neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Phi$, *then* $\Phi \cup \{\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$ *or* $\Phi \cup \{\neg\mathbf{A}, \mathbf{B}\} \in \Gamma_\Sigma$.
∇_q *If* $\neg(\mathbf{F} \doteq^{\alpha\to\beta} \mathbf{G}) \in \Phi$, *then* $\Phi \cup \{\neg(\mathbf{F}\ w \doteq^\beta \mathbf{G}\ w)\} \in \Gamma_\Sigma$ *for any new constant* $w \in \Sigma_\alpha$.
∇_e^r $\neg(\mathbf{A}_\alpha = \mathbf{A}) \notin \Phi$. ∇_e^s *If* $\mathbf{F}[\mathbf{A}]_p \in \Phi$ *and* $\mathbf{A} = \mathbf{B} \in \Phi$, *then* $\Phi \cup \{\mathbf{F}[\mathbf{B}]_p\} \in \Gamma_\Sigma$.[2]

Theorem 1 (Henkin Model Existence). *Let Φ be a set of closed Σ-formulas ($\Sigma^=$-formulas), Γ_Σ ($\Gamma_{\Sigma^=}$) be an Acc ($Acc^=$) and $\Phi \in \Gamma_\Sigma$. There exists a Henkin model $\mathcal{M}$, such that $\mathcal{M} \models \Phi$.*

$$\frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^T}{\mathbf{C} \vee [\mathbf{A}]^T \vee [\mathbf{B}]^T} \vee^T \quad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{A}]^F} \vee_l^F \quad \frac{\mathbf{C} \vee [\mathbf{A} \vee \mathbf{B}]^F}{\mathbf{C} \vee [\mathbf{B}]^F} \vee_r^F$$

$$\frac{\mathbf{C} \vee [\neg \mathbf{A}]^T}{\mathbf{C} \vee [\mathbf{A}]^F} \neg^T \quad \frac{\mathbf{C} \vee [\neg \mathbf{A}]^F}{\mathbf{C} \vee [\mathbf{A}]^T} \neg^F \quad \frac{\mathbf{C} \vee [\Pi^\gamma \mathbf{A}]^T \quad X_\gamma \text{ new variable}}{\mathbf{C} \vee [\mathbf{A}\ X]^T} \Pi^T$$

$$\frac{\mathbf{C} \vee [\Pi^\gamma \mathbf{A}]^F \quad \mathrm{sk}_\gamma \text{ is a Skolem term for this clause}}{\mathbf{C} \vee [\mathbf{A}\ \mathrm{sk}_\gamma]^F} \Pi^F$$

Fig. 1. Clause Normalisation Calculus $\mathcal{CNF}$

Clause Normalisation:

$$\frac{\mathcal{D} \quad \mathcal{C} \in \mathcal{CNF}(\mathcal{D})}{\mathcal{C}} \mathit{Cnf}$$

Resolution: (defined on proper clauses only)

$$\frac{[\mathbf{A}]^\alpha \vee \mathbf{C} \quad [\mathbf{B}]^\beta \vee \mathbf{D} \quad \alpha \neq \beta}{\mathbf{C} \vee \mathbf{D} \vee [\mathbf{A} = \mathbf{B}]^F} \mathit{Res} \qquad \frac{[\mathbf{A}]^\alpha \vee [\mathbf{B}]^\alpha \vee \mathbf{C} \quad \alpha \in \{T, F\}}{[\mathbf{A}]^\alpha \vee \mathbf{C} \vee [\mathbf{A}\ = \mathbf{B}]^F} \mathit{Fac}$$

$$\frac{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \quad \mathbf{P} \in \mathcal{AB}_\gamma^{\{\neg,\vee\} \cup \{\Pi^\beta \mid \beta \in \mathcal{T}^k\}},\ \alpha \in \{T, F\}}{[Q_\gamma\ \overline{\mathbf{U}^k}]^\alpha \vee \mathbf{C} \vee [Q = \mathbf{P}]^F} \mathit{Prim}$$

Extensional (Pre-)Unification:

$$\frac{\mathbf{C} \vee [\mathbf{M}_{\gamma \to \beta} = \mathbf{N}_{\gamma \to \beta}]^F \quad s_\gamma \text{ Skolem term for this clause}}{\mathbf{C} \vee [\mathbf{M}\ s = \mathbf{N}\ s]^F} \mathit{Func}$$

$$\frac{\mathbf{C} \vee [\mathbf{A}_{\gamma \to \beta}\ \mathbf{C}_\gamma = \mathbf{B}_{\gamma \to \beta}\ \mathbf{D}_\gamma]^F}{\mathbf{C} \vee [\mathbf{A} = \mathbf{B}]^F \vee [\mathbf{C} = \mathbf{D}]^F} \mathit{Dec}$$

$$\frac{\mathbf{C} \vee [\mathbf{A} = \mathbf{A}]^F}{\mathbf{C}} \mathit{Triv} \qquad \frac{\mathbf{C} \vee [X = \mathbf{A}]^F \quad X \text{ does not occur in } \mathbf{A}}{\mathbf{C}_{\{\mathbf{A}/X\}}} \mathit{Subst}$$

$$\frac{\mathbf{C} \vee [F_\gamma\ \overline{\mathbf{U}^n} = h\ \overline{\mathbf{V}^m}]^F \quad \mathbf{G} \in \mathcal{AB}_\gamma^h}{\mathbf{C} \vee [F\ \overline{\mathbf{U}^n} = h\ \overline{\mathbf{V}^m} \vee [F = \mathbf{G}]^F]^F} \mathit{FlexRigid}$$

$$\frac{\mathbf{C} \vee [\mathbf{M}_o = \mathbf{N}_o]^F}{\mathbf{C} \vee [\mathbf{M}_o \Leftrightarrow \mathbf{N}_o]^F} \mathit{Equiv} \qquad \frac{\mathbf{C} \vee [\mathbf{M}_\gamma = \mathbf{N}_\gamma]^F}{\mathbf{C} \vee [\forall P_{\gamma \to o}.\ P\ \mathbf{M} \Rightarrow P\ \mathbf{N}]^F} \mathit{Leib}$$

$$\left(\frac{\mathbf{C} \vee [F_{\overline{\gamma^n} \to \gamma}\ \overline{\mathbf{U}^n} = H_{\overline{\delta^m} \to \gamma}\ \overline{\mathbf{V}^m}]^F \quad \mathbf{G} \in \mathcal{AB}^h_{\overline{\gamma^n} \to \gamma} \text{ for a constant } h_\tau}{\mathbf{C} \vee [F\ \overline{\mathbf{U}^n} = H\ \overline{\mathbf{V}^m}]^F \vee [F = \mathbf{G}]^F} \mathit{FlexFlex} \right)$$

$\mathcal{AB}_\gamma^h$ specifies the set of partial bindings of type γ for head h as defined in [SG89]

Fig. 2. Extensional HO Resolution Calculus $\mathcal{ER}$

3 $\mathcal{ER}$: Extensional HO Resolution

Figure 1 presents calculus $\mathcal{CNF} := \{\vee^T, \vee^F_l, \vee^F_r, \neg^T, \neg^F, \Pi^T, \Pi^F\}$ for clause normalisation. These rules are defined on (pre-)clauses and are known to preserve validity or satisfiability with respect to standard semantics.[3]

The syntactical unification rules (cf. Fig. 2) provided by $\mathcal{ER}$ which operate on unification constraints are $\mathcal{UNI} := \{Func, Dec, Triv, Subst, FlexRigid\}$. These rules realise a sound and complete approach to HO pre-unification. Note the double role of extensionality rule *Func*: on the one hand this rule works as a syntactical unification rule and subsumes the α- and η-rule as, e.g. presented in [BK98a]; on the other hand *Func* applies the functional extensionality principle if none of the two terms is a λ-abstraction. Apart from rule *Func*, $\mathcal{ER}$ provides the extensionality rules *Equiv* and *Leib* (cf. Fig. 2). The former applies the Boolean extensionality principle and the latter simply replaces a negative unification constraint (encoded as a negative equation) by a negative Leibniz equation. The extensionality rules operate on unification constraints only and do in contrast to the respective axioms not introduce flexible heads into the search space.

The main proof search is performed by the resolution rule *Res* and the factorisation rule *Fac*. It is furthermore well known for HO resolution, that the primitive substitution rule *Prim* is needed to ensure Henkin completeness.

For the calculi presented in this paper we assume that the result of each rule application is transformed into hnf[4], where the hnf of unification constraints is defined special and requires both unification terms to be reduced to hnf. A set of formulas Φ is refutable in calculus R, iff there is a derivation $\Delta : \Phi_{cl} \vdash_R \square$, where $\Phi_{cl} := \{[\mathbf{F}']^T | \mathbf{F}' \text{ hnf of } \mathbf{F} \in \Phi\}$ is the clause-set obtained from Φ by simple pre-clausification. More details on $\mathcal{ER}$ are provided by [BK98a,Ben99].

Whereas completeness of $\mathcal{ER}$ has already been analysed in [BK98a] this paper (and [Ben99]) presents an alternative completeness proof for a slightly extended version of $\mathcal{ER}$ (this version, e.g. employs the instantiation guessing *FlexFlex*-rule). The new proof is motivated as follows: (i) it eases the proof of the lifting lemma and avoids the quite complicated notion of clause isomorphisms as used in [BK98a,Koh94], (ii) it can be reused to show the completeness for calculi $\mathcal{EP}$ and $\mathcal{ERUE}$ as well, (iii) it prepares the analysis of non-normal form resolution calculi, and (iv) it emphasises interesting aspects on rule *FlexFlex*, unification, and clause normalisation wrt. $\mathcal{ER}$, $\mathcal{EP}$, and $\mathcal{ERUE}$.

One such interesting aspect is that different to Huet [Hue72] eager unification is essential within our approach. This is illustrated by the argumentations for ∇_b and ∇_q in the completeness proofs (cf. [Ben99,BK98a]) as well as the examples presented in Sec. 7 or [Ben99]. However, we claim that rule *FlexFlex* can still be delayed until the end of a refutation, i.e., *FlexFlex* can be completely avoided.

The author has not been able to prove the latter claim yet. And thus the completeness proofs for $\mathcal{ER}$ (and $\mathcal{EP}$, $\mathcal{ERUE}$) still depends on the *FlexFlex*-rule.

[2] **A** does not contain free variables.
[3] For Skolemisation we employ Miller's sound HO correction [Mil83].
[4] One may also $\beta\eta$-normal form here.

We now sketch the main results on $\mathcal{ER}$ as discussed in detail in [Ben99].

Definition 2 (Extensional HO Resolution). *We define three calculi:*
$\mathcal{ER} := \{Cnf, Res, Fac, Prim\} \cup \mathcal{UNI} \cup \{Equiv, Leib\}$ *employs all rules (except* FlexFlex*) displayed in Fig. 2.*
$\mathcal{ER}_f := \mathcal{ER} \cup \{FlexFlex\}$ *uses full HO unification instead of pre-unification.*
$\mathcal{ER}_{fc} := (\mathcal{ER}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$ *employs unfolded and stepwise clause normalisation instead of exhaustive normalisations with rule* Cnf.
These calculi treat equality as a defined notion only (e.g. by Leibniz equality) and primitive equations are not allowed in problem formulations. Although unification constraints are encoded as negative equation literals, no rule but the unification rules are allowed to operate on them.

Theorem 2 (Soundness). *The calculi $\mathcal{ER}$, $\mathcal{ER}_f$, and $\mathcal{ER}_{fc}$ are Henkin-sound (H-sound).*

Proof. Preservation of validity or satisfiability with respect to Henkin semantics is proven analogously to the standard FO argumentation. For Skolemisation (employed in rule Π^F and *Func*) we use Miller's sound HO version [Mil83]. Soundness of the extensionality rules *Equiv, Func*, and *Leib* is obvious as they simply apply the valid extensionality principles.

Lemma 1 (Lifting of $\mathcal{ER}_{fc}$). *Let Φ be clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. One can easily show that each instantiated derivation can be reused on the uninstantiated level as well. In blocking situations caused by free variables at literal head position or at unification term head position, either rule *Prim* or rule *FlexFlex* can be employed in connection with rule *Subst* to introduce the missing term structure. The rather unintuitive clause isomorphisms of [BK98a] or [Koh94] are thereby avoided.

Theorem 3 (Completeness). *Calculus $\mathcal{ER}_{fc}$ is Henkin complete.*

Proof. Analogously to the proof in [BK98a] we show that the set of closed formulas that are not refutable in $\mathcal{ER}_{fc}$ (i.e., $I^{-}_{\Sigma} := \{\Phi \subseteq cwff_o | \Phi_{cl} \nvdash_{\mathcal{ER}_{fc}} \Box\}$) is a saturated abstract consistency class for Henkin models (cf. Def. 1). This entails Henkin completeness for $\mathcal{ER}_{fc}$ by Thm. 1.

Lemma 2 (Theorem Equivalence). *The calculi $\mathcal{ER}_{fc}$ and $\mathcal{ER}_f$ are theorem equivalent, i.e., for each clause set Φ holds that $\Phi \vdash_{\mathcal{ER}_{fc}} \Box$ iff $\Phi \vdash_{\mathcal{ER}_f} \Box$.*

Proof. We can even prove a more general property: For each proper clause $\mathcal{C}$ holds $\Phi \vdash_{\mathcal{ER}_{fc}} \mathcal{C}$ implies $\Phi \vdash_{\mathcal{ER}_f} \mathcal{C}$. The proof idea is to show that the unfolded clause normalisations can be grouped together and then replaced by rule *Cnf*.

Question 1 (Theorem Equivalence). The author claims that the calculi $\mathcal{ER}$ and $\mathcal{ER}_{fc}$ (or $\mathcal{ER}_f$) are theorem equivalent. A formal proof has not been carried out yet. Some evidence is given by the case studies carried out with the LEO-prover [BK98b] and the direct completeness proof for $\mathcal{ER}$ in [BK98a].

4 Primitive Equality

Treating equality as a defined notion in HO logic (e.g. by the Leibniz principle) is convenient in theory, but often inefficient and unintuitive in practical applications as many free literal heads are introduced into the search space, which increases the degree of blind search with primitive substitution rule *Prim*.[5] This is the main motivation for the two approaches to primitive equality presented in the next sections. Before we discuss these approaches in detail we point to the following interesting aspects of defined equality in HO logic:

- There are infinitely many different valid definitions of equality in HO logic.[6] For instance: Leibniz equality ($\doteq^\alpha := \lambda X_\alpha.\ \lambda Y_\alpha.\ \forall P_{\alpha\to o}.\ PX \Rightarrow PY$), Reflexivity definition[7] ($\ddot{=}^\alpha := \lambda X_\alpha.\ \lambda Y_\alpha.\ \forall Q_{\alpha\to\alpha\to o}.\ (\forall Z_\alpha.\ (Q\ Z\ Z)) \Rightarrow (Q\ X\ Y)$), and infinitely many artificial modifications to all valid definitions (e.g., $\dddot{=}^\alpha := \lambda X_\alpha.\ \lambda Y_\alpha.\ \forall P_{\alpha\to o}.\ ((a_o \vee \neg\ a_o) \wedge P\ X) \Rightarrow ((b_o \vee \neg\ b_o) \wedge P\ Y)$). The latter definition is obviously equivalent to Leibniz definition as it just adds some tautologies to the embedded formulas.
- The artificially modified definitions demonstrate, that it is generally not decidable whether a formula is a valid definition of equality (as the set of tautologies is not decidable). Hence, it is not decidable whether an input problem to one of our proof procedures contains a valid definition of equality, and we cannot simply replace all valid definitions embedded in a problem formulation by primitive equations as one might wish to.

If we are interested in Henkin completeness, we therefore have to ensure that the paramodulation and RUE-resolution approaches presented in the next sections can handle all forms of defined equality (like the underlying calculus $\mathcal{ER}$) and can additionally handle primitive equality.[8]

5 $\mathcal{EP}$: Extensional HO Paramodulation

In this section we adapt the well known FO paramodulation approach [RW69] to our HO setting and examine Henkin completeness. A straightforward adaptation of the traditional FO paramodulation rule is given by rule *Para* in Fig. 3. Analogous to the $\mathcal{ER}$ rules *Res* and *Fac*, (pre-)unification is delayed by encoding the respective unification problem (its solvability justifies the rewriting step) as

[5] This is illustrated by the examples that employ defined equality in [BK98a] and the examples that employ primitive equality in Sect. 7.

[6] For this statement we assume Henkin or standard semantics as underlying semantical notion. In weaker semantics things get even more complicated as, e.g., Leibniz equality does not necessary denote the intended equality relation. For a detailed discussion see [Ben99,BK97].

[7] As presented in Andrews textbook [And86], p. 155.

[8] The author admits, that in practice one is mainly interested in finding proofs rather than in the theoretical notion of Henkin completeness. Anyhow, our motivation in this paper is to clarify the theoretical properties of our approaches.

$$\frac{[\mathbf{A}[\mathbf{T}_\gamma]]^\alpha \vee C \quad [\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T \vee D}{[\mathbf{A}[\mathbf{R}]]^\alpha \vee C \vee D \vee [\mathbf{T} = \mathbf{L}]^F}\ Para \qquad \frac{[\mathbf{A}]^\alpha \vee C \quad [\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T \vee D}{[P_{\gamma\to o}\ \mathbf{R}]^\alpha \vee C \vee D \vee [\mathbf{A} = P\ \mathbf{L}]^F}\ Para'$$

We implicitly assume the symmetric application of $[\mathbf{L}_\gamma = \mathbf{R}_\gamma]^T$.
$\mathbf{T}$ (in *Para*) does not contain free variables which are bound outside of $\mathbf{T}$.

Fig. 3. Adapted Paramodulation Rule and a HO specific reformulation

a unification constraint. Rule *Para'* is an elegant HO specific reformulation[9] of paramodulation that has a very simple motivation: It describes the resolution step with the clause $[P\ \mathbf{L}]^F \vee [P\ \mathbf{R}]^T \vee D$, i.e., the clause obtained when replacing the primitive equation $[\mathbf{L} = \mathbf{R}]^T$ by its Leibniz definition. Note that the paramodulant of *Para'* encodes all possible single rewrite steps, all simultaneous rewrite-steps with rule *Para*, and in some sense even the left premise clause itself. This is nicely illustrated by the following example: $\mathcal{C}_1 : [p\ (f\ (f\ a))]^T$ and $\mathcal{C}_2 : [f = h]^T$, where $p_{\iota\to o}, f_{\iota\to\iota}, h_{\iota\to\iota}$ are constants. Applying rule *Para'* to $\mathcal{C}_1$ and $\mathcal{C}_2$ from left to right leads to $\mathcal{C}_3 : [P_{(\iota\to\iota)\to\iota}\ h]^T \vee [p\ (f\ (f\ a)) = P_{(\iota\to\iota)\to\iota}\ f]^F$. Eager unification computes the following four solutions for P, which can be back-propagated to literal $[P\ h]^T$ with rule *Subst*:

$[\lambda Z_{\iota\to\iota}.\ p\ (f\ (f\ a))/P]$ the pure imitation solution encodes $\mathcal{C}_1$ itself.
$[\lambda Z_{\iota\to\iota}.\ p\ (Z\ (f\ a))/P]$ encodes the rewriting of the first f ($[p\ (h\ (f\ a))]^T$).
$[\lambda Z_{\iota\to\iota}.\ p\ (f\ (Z\ a))/P]$ encodes the rewriting of the second f ($[p\ (f\ (h\ a))]^T$).
$[\lambda Z_{\iota\to\iota}.\ p\ (Z\ (Z\ a))/P]$ encodes the simult. rewr. of both f ($[p\ (h\ (h\ a))]^T$).

Rule *Para'* introduces flexible literal heads into the search space such that rule *Prim* becomes applicable. Thus, a probably suitable heuristics in practice is to avoid all primitive substitution steps on flexible heads generated by rule *Para'*.

Note that reflexivity resolution[10] and paramodulation into unification constraints[11] are derivable in our approach and can thus be avoided.

[9] This rule was first suggested by Michael Kohlhase.

[10] In FO a reflexivity resolution rule is needed to refute negative equation literals $[\mathbf{T}_1 = \mathbf{T}_2]^F$ if $\mathbf{T}_1$ and $\mathbf{T}_2$ are unifiable. As such literals are automatically treated as unification constraints reflexivity resolution is not needed in our approach.

[11] Let $\mathcal{C}_1 : C \vee [\mathbf{A}[\mathbf{T}] = \mathbf{B}]^F$ and $\mathcal{C}_2 : [\mathbf{L} = \mathbf{R}]^T \vee D$. The rewriting step $Para(\mathcal{C}_1, \mathcal{C}_2) : \mathcal{C}_3 : C \vee D \vee [\mathbf{A}[\mathbf{R}] = \mathbf{B}]^F \vee [\mathbf{L} = \mathbf{T}]^F$ can be replaced by derivation $Leib(\mathcal{C}_1) : \mathcal{C}_4 : [p\ \mathbf{A}[\mathbf{T}]]^T \vee C,\ \mathcal{C}_5 : [p\ \mathbf{B}]^F \vee C;\ Para(\mathcal{C}_4, \mathcal{C}_2) : \mathcal{C}_6 : [p\ \mathbf{A}[\mathbf{R}]]^T \vee C \vee D \vee [\mathbf{L} = \mathbf{T}]^F;\ Res(\mathcal{C}_6, \mathcal{C}_5), Fac, Triv : \mathcal{C}_7 : C \vee D \vee [p\ \mathbf{A}[\mathbf{R}] = p\ \mathbf{B}]^F \vee [\mathbf{L} = \mathbf{T}]^F;\ Dec(\mathcal{C}_7) : \mathcal{C}_3$. Notational remark: $Res(\mathcal{C}_6, \mathcal{C}_5), Fac, Triv$ describes the application of rule *Res* to $\mathcal{C}_6$ and $\mathcal{C}_5$, followed by applications of *Fac* and *Triv* to the subsequent results.

In the following discussion we will use the traditional paramodulation rule *Para* only.[12] As *Para'* is obviously more general than *Para* we obtain analogous completeness results if we employ *Para'* instead.

Definition 3 (Simple HO Paramodulation). *$\mathcal{EP}_{naive} := \mathcal{ER} \cup \{Para\}$ extends the extensional HO resolution approach by rule* Para. *Primitive equations in input problems are no longer expanded by Leibniz definition.* Para *operates on proper clause only and omits paramodulation into unification constraints.*

Whereas soundness of rule *Para* can be shown analogously to the FO case, it turns out that our simple HO paramodulation approach is incomplete:

Theorem 4 (Incompleteness). *Calculus $\mathcal{EP}_{naive}$ is Henkin incomplete.*

Proof. Consider the following counterexamples: $\mathbf{E}_1^{Para}$**:** $\neg\exists X_o.\ (X = \neg X)$, i.e., the negation operator is fix-point free, which is obviously the case in Henkin semantics. Negation and clause normalisation leads to clause $\mathcal{C}_1 :\ [a = \neg a]^T$, where a_o is a new Skolem constant. The only rule that is applicable is self-paramodulation at positions $\langle 1\rangle$, $\langle 2\rangle$, and $\langle\rangle$, leading to the following clauses (including the symmetric rewrite steps):

$Para(\mathcal{C}_1, \mathcal{C}_1)$ *at* $\langle 1\rangle : \mathcal{C}_2 :\ [a = \neg a]^T \vee [\neg a = a]^F, \quad \mathcal{C}_3 :\ [\neg a = \neg a]^T \vee [a = a]^F$
$Para(\mathcal{C}_1, \mathcal{C}_1)$ *at* $\langle 2\rangle : \mathcal{C}_4 :\ [a = \neg a]^T \vee [a = \neg a]^F, \quad \mathcal{C}_5 :\ [a = a]^T \vee [\neg a = \neg a]^F$
$Para(\mathcal{C}_1, \mathcal{C}_1)$ *at* $\langle\rangle :\ \mathcal{C}_6 :\ [a]^T \vee [\neg a = (a = \neg a)]^F, \mathcal{C}_7 :\ [a]^F \vee [a = (a = \neg a)]^F$

A case distinction on the possible denotations $\{\top, \bot\}$ for a shows that all clauses are tautologous, such that no refutation is possible in $\mathcal{EP}_{naive}$. Additional examples are discussed in [Ben99], e.g. $\mathbf{E}_2^{Para}$**:** $[G\ X = p]^T$, which stems from a simple version of cantor's theorem $\neg\exists G_{\iota\to\iota\to o}.\ \forall P_{\iota\to o}.\ \exists X_\iota.\ G\ X = P$, or example $\mathbf{E}_3^{Para}$**:** $[M = \lambda X_o.\ \bot]^T$, which stems from $\exists M_{o\to o}.\ M \neq \emptyset$.

The problem is that in HO logic even single positive equation literals can be contradictory. And the incompleteness is caused as the extensionality principles are now also needed to refute such positive equation literals.[13] Hence, we add the positive counterparts *Func'* and *Equiv'* (cf. Fig. 4) to the already present negative extensionality rules *Func* and *Equiv*. The completeness proof and the examples show that a positive counterpart for rule *Leib* can be avoided.

Definition 4 (Extensional HO Paramodulation). *Analogously to the extensional HO resolution case we define the calculi $\mathcal{EP} := \mathcal{ER} \cup \{Para, Equiv', Func'\}$, $\mathcal{EP}_f := \mathcal{EP} \cup \{FlexFlex\}$, and $\mathcal{EP}_{fc} := (\mathcal{EP}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$.*

Theorem 5 (Soundness). *The calculi $\mathcal{EP}$, $\mathcal{EP}_f$, and $\mathcal{EP}_{fc}$ are H-sound.*

[12] It has been pointed out by a unknown referee of this paper that rule *Para'* already captures full functional extensionality and should therefore be preferred over *Para*. Example $\mathbf{E}_1^{func}$ discussed in Sec. 10.6 of [Ben99] illustrates that this is generally not true.

[13] In contrast to $\mathcal{EP}$, the underlying calculus $\mathcal{ER}$ does not allow positive equation literals as the equality symbol is only used to encode unification constraints. Therefore the pure extensional HO resolution approach $\mathcal{ER}$ does not require a positive extensionality treatment.

$$\frac{\mathbf{C} \vee [\mathbf{M}_o = \mathbf{N}_o]^T}{\mathbf{C} \vee [\mathbf{M}_o \Leftrightarrow \mathbf{N}_o]^T}\ \mathit{Equiv}' \qquad \frac{\mathbf{C} \vee [\mathbf{M}_{\gamma\to\beta} = \mathbf{N}_{\gamma\to\beta}]^T \quad X \text{ new variable}}{\mathbf{C} \vee [\mathbf{M}\ X_\gamma = \mathbf{N}\ X_\gamma]^T}\ \mathit{Func}'$$

Fig. 4. Positive Extensionality Rules

Proof. Soundness of rule *Para* with respect to Henkin semantics can be proven analogously to the FO case and soundness of *Equiv'* and *Func'* is obvious, as they simply apply the extensionality principles, which are valid in Henkin semantics.

Lemma 3 (Lifting of $\mathcal{EP}_{fc}$). *Let Φ be a clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{EP}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. Analogous to Lemma 1. The additional rules do not cause any problems.

The main completeness theorem 6 for $\mathcal{EP}_{fc}$ below is proven analogously to Thm. 3, i.e., we employ the model existence theorem for Henkin models with primitive equality (cf. Thm. 1). As primitive equality is involved, we additionally have to ensure the abstract consistency properties ∇_e^r and ∇_e^s (cf. Def. 1), i.e., the reflexivity and substitutivity property of primitive equality. Whereas the reflexivity property is trivially met, we employ the following admissible[14] — and moreover even weakly derivable (i.e., modulo clause normalisation and lifting) — paramodulation rule to verify the substitutivity property.

Definition 5 (Generalised Paramodulation). *The generalised paramodulation rule* GPara *is defined as follows:*

$$\frac{[\mathbf{T}[\mathbf{A}_\beta]]^\alpha \vee C \quad [\mathbf{A}_\beta = \mathbf{B}_\beta]^T}{[\mathbf{T}[\mathbf{B}]]^\alpha \vee C}\ \mathit{GPara}$$

This rule extends Para *as it can be applied to non-proper clauses and it restricts* Para *as it can only be applied in special clause contexts, e.g. the second clause has to be a unit clause.* GPara *is especially designed to verify the substitutivity property of primitive equality ∇_e^s in the main completeness theorem 6.*

Weak derivability (which obviously implies admissibility) of *GPara* is shown with the help of the following weakly derivable generalised resolution rules.

Definition 6 (Generalised Resolution). *The generalised resolution rules $GRes_1$, $GRes_2$, and $GRes_3$ are defined as follows (for all rules we assume $\alpha, \beta \in \{T, F\}$ with $\alpha \neq \beta$, and for $GRes_2$ we assume that $\overline{Y^n} \notin \mathbf{free}(\mathbf{A})$):*

[14] Rule r is called admissible (derivable) in R, iff adding rule r to calculus R does not increase the set of refutable formulas (iff each application of rule r can be replaced by an alternative derivation in calculus R).

$$\frac{[\mathbf{A}_{\overline{\gamma}\to o}\ \overline{\mathbf{T}^n_\gamma}]^\alpha \vee C \quad [\mathbf{A}_{\overline{\gamma}\to o}\ \overline{X^n_\gamma}]^\beta \vee D}{(C \vee D)_{[\overline{T^n}/\overline{X^n}]}}\ GRes_1 \qquad \frac{[\mathbf{A}_\gamma\ \overline{Y^n}]^\alpha \vee C \quad [X_\gamma\ \overline{\mathbf{T}^n}]^\beta \vee D}{(C \vee D)_{[\mathbf{A}/X,\overline{\mathbf{T}^n}/\overline{Y^n}]}}\ GRes_2$$

$$\frac{[\mathbf{A}_\gamma\ \overline{\mathbf{T}^n}]^\alpha \vee C \quad [X_\gamma\ \overline{Y^n}]^\beta \vee D}{(C \vee D)_{[\mathbf{A}/X,\overline{\mathbf{T}^n}/\overline{Y^n}]}}\ GRes_3$$

These rules extend Res *as they can be applied to non-proper clauses, and they restrict* Res *as they are only defined for special clause contexts. The rules are designed just strong enough to prove weak derivability of* GPara*.*

Lemma 4 (Weak Derivability of $GRes_{1,2,3}$). *Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ be clauses and $r \in \{GRes_1, GRes_2, GRes_3\}$. If $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash^r \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ for a proper clause $\mathcal{C}_4$, then $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EP}_{fc}} \mathcal{C}_5$ for a clause $\mathcal{C}_5$ which generalises $\mathcal{C}_4$.*

Proof. The proof is by induction on the number of logical connectives in the resolution literals. It employs generalised (and weakly derivable) versions of the factorisation rule *Fac* and primitive substitution rule *Prim* (see [Ben99]), which are not presented here because lack of space. $GRes_2$ and $GRes_3$ are needed to prove weak derivability for $GRes_1$. As the rules *Para, Equiv', Func'* are not employed in the proof, this lemma analogously holds for calculus $\mathcal{ER}_{fc}$.

Lemma 5 (Weak Derivability of $GPara$). *Let $\mathcal{C}_1 : [\mathbf{T}[\mathbf{A}]_p]^\alpha \vee D_1, \mathcal{C}_2 : [\mathbf{A} = \mathbf{B}]^T, \mathcal{C}_3 : [\mathbf{T}[\mathbf{B}]_p]^\alpha \vee D_1$ be clauses. If $\Delta : \{\mathcal{C}_1, \mathcal{C}_2\} \vdash^{GPara} \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ for a proper clause $\mathcal{C}_4$, then $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{EP}_{fc}} \mathcal{C}_5$ for a clause $\mathcal{C}_5$ generalising $\mathcal{C}_4$.*

Proof. The proof is by induction on the length of Δ and employs the (weakly derivable) generalised resolution rule $GRes_1$ and the standard paramodulation rule *Para* in the quite complicated base case.

Theorem 6 (Completeness). *Calculus $\mathcal{EP}_{fc}$ is Henkin complete.*

Proof. Let $\mathit{\Gamma}_\Sigma$ be the set of closed Σ-formulas that cannot be refuted with calculus $\mathcal{EP}_{fc}$ (i.e., $\mathit{\Gamma}_\Sigma := \{\Phi \subseteq \mathit{cwff}_o | \Phi_{cl} \not\vdash_{\mathcal{EP}_{fc}} \Box\}$). We show that $\mathit{\Gamma}_\Sigma$ is a saturated abstract consistency class for Henkin models with primitive equality (cf. Def. 1). This entails completeness by the model existence theorem for Henkin models with primitive equality (cf. Thm. 1).

First we have to verify that $\mathit{\Gamma}_\Sigma$ validates the abstract consistency properties ∇_c, $\nabla_\neg$, ∇_β, $\nabla_\vee$, $\nabla_\wedge$, $\nabla_\forall$, $\nabla_\exists$, ∇_b, ∇_q and that $\mathit{\Gamma}_\Sigma$ is saturated. In all of these cases the proofs are identical to the corresponding argumentations in Thm. 3.

Thus, all we need to ensure is the validity of the additional abstract consistency properties ∇^r_e and ∇^s_e for primitive equality:
(∇^r_e) We have that $[\mathbf{A} =^\alpha \mathbf{A}]^F \vdash^{Triv} \Box$, and thus $\neg(\mathbf{A} =^\alpha \mathbf{A})$ cannot be in Φ.
(∇^s_e) Analogously to the cases in Sec. 3 we show the contrapositive of the assertion, and thus we assume that there is derivation $\Delta : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$. Now consider the following $\mathcal{EP}_{fc}$-derivation: $\Delta' : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T\} \vdash^{GPara} \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T, [\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$. By Lemma 5 *GPara* is weakly derivable (hence admissible) for calculus $\mathcal{EP}_{fc}$, such that there is a $\mathcal{EP}_{fc}$-derivation $\Delta'' : \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T\} \vdash_{\mathcal{EP}_{fc}} \Phi_{cl} \cup \{[\mathbf{F}[\mathbf{A}]]^T, [\mathbf{A} = \mathbf{B}]^T, [\mathbf{F}[\mathbf{B}]]^T\} \vdash_{\mathcal{EP}_{fc}} \Box$ which completes the proof.

Lemma 6 (Theorem Equivalence). *$\mathcal{EP}_{fc}$ and $\mathcal{EP}_f$ are theorem equivalent.*

Proof. Analogous to Lemma 2. The additional rules do not cause any problems.

Question 2 (Theorem Equivalence). The author claims that the calculi $\mathcal{EP}$ and $\mathcal{EP}_{fc}$ (or $\mathcal{EP}_f$) are theorem equivalent. The formal proof will most likely be analogous to the one for question 1.

6 $\mathcal{ERUE}$: Extensional HO RUE-Resolution

In this section we will adapt the **R**esolution by **U**nification and **E**quality approach [Dig79] to our higher-order setting. The key idea is to allow the resolution and factorisation rules also to operate on unification constraints (which is forbidden in $\mathcal{ER}$ and $\mathcal{EP}$). This implements the main ideas of FO RUE-resolution directly in our higher-order calculus. More precisely our approach allows to compute partial E-unifiers with respect to a specified theory E by resolution on unification constraints within the calculus itself (if we assume that E is specified in form of an available set of unitary or even conditional equations in clause form). 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 higher-order E-unification in case we also add the rule *FlexFlex*). Furthermore, each partial E-(pre-)unifier can be applied to a clause with rule *Subst*, and, like in the traditional FO RUE-resolution approach, the non-solved unification constraints are encoded as (still open) unification constraints, i.e., negative equations, within the particular clauses.

Definition 7 (Extensional HO RUE-Resolution). *We now allow the factorisation rule* Fac *and resolution rule* Res *to operate also on unification constraints and define the calculi $\mathcal{ERUE} := \mathcal{ER} \cup \{Equiv', Func'\}$, $\mathcal{ERUE}_f := \mathcal{ERUE} \cup \{FlexFlex\}$, and $\mathcal{ERUE}_{fc} := (\mathcal{ERUE}_f \backslash \{Cnf\}) \cup \mathcal{CNF}$.*

Theorem 7 (Soundness). *The calculi $\mathcal{ERUE}_{fc}$, $\mathcal{ERUE}_f$, and $\mathcal{ERUE}$ are H-sound.*

Proof. Unification constraints are encoded as negative literals, such that soundness of the extended resolution and factorisation rules with respect to Henkin semantics is obvious.

Lemma 7 (Lifting of $\mathcal{ERUE}_{fc}$). *Let Φ be a clause set, $\mathcal{D}_1$ a clause, and σ a substitution. If $\sigma(\Phi) \vdash_{\mathcal{ER}_{fc}} \mathcal{D}_1$, then $\Phi \vdash_{\mathcal{ERUE}_{fc}} \mathcal{D}_2$ for a clause $\mathcal{D}_2$ generalising $\mathcal{D}_1$.*

Proof. Analogous to Lemmata 1 and 3.

Within the main completeness proof we proceed analogously to previous section and employ the generalised paramodulation rule *GPara* to verify the crucial substitutivity property ∇_e^s. Thus, we need to show that *GPara* is admissible in calculus $\mathcal{ERUE}_{fc}$. Note that in Lemma 5 we were even able to show a weak derivability property of rule *GPara* for calculus $\mathcal{EP}_{fc}$. Whereas *GPara* is not weakly derivability for calculus $\mathcal{ERUE}_{fc}$, we can still prove admissibility of this rule here. As in Lemma 5, we employ the generalised resolution rules which are weakly derivable in $\mathcal{ERUE}_{fc}$ as well.

Lemma 8 (Weak Derivability of $GRes_{1,2,3}$). *Let $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ be clauses and $r \in \{GRes_1, GRes_2, GRes_3\}$. If $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash^r \mathcal{C}_3 \vdash_{\mathcal{CNF}} \mathcal{C}_4$ for a proper clause $\mathcal{C}_4$, then $\{\mathcal{C}_1, \mathcal{C}_2\} \vdash_{\mathcal{ERUE}_{fc}} \mathcal{C}_5$ for a clause $\mathcal{C}_5$ which generalises $\mathcal{C}_4$.*

Proof. Analogous to Lemma 4.

Lemma 9 (Admissibility of $GPara$). *Let Φ be a clause set, such that $\Delta : \Phi \vdash^{GPara} \Phi' \vdash_{\mathcal{ERUE}_{fc}} \Box$, then there exists a refutation $\Phi \vdash_{\mathcal{ERUE}_{fc}} \Box$.*

Proof. The proof is (analogous to Lemma 5) by induction on the length of Δ and employs the weakly derivable generalised resolution rule $GRes_1$. The applications of rule *Para* in the proof of Lemma 5 are replaced by corresponding derivations employing resolution and factorisation on unification constraints. The latter causes the loss of the weak derivability property.

Theorem 8 (Completeness). *Calculus $\mathcal{ERUE}_{fc}$ is Henkin complete.*

Proof. Analogously to Lemma 6 we show that the set of closed Σ-formulas which cannot be refuted by the calculus $\mathcal{ERUE}_{fc}$ (i.e., $I_\Sigma := \{\Phi \subseteq cwff_o | \Phi_{cl} \nvdash_{\mathcal{ERUE}_{fc}} \Box\}$) is a saturated abstract consistency class for Henkin models with primitive equality (cf. Def. 1). This entails the assertion by Thm. 1.

The proof is analogous to Lemma 6. Even the abstract consistency properties ∇_e^r and ∇_e^s are proven analogously by employing the generalised paramodulation rule *GPara*, which is by Lemma 9 admissible in $\mathcal{ERUE}_{fc}$.

Lemma 10 (Theorem Equiv.). *$\mathcal{ERUE}_{fc}$ and $\mathcal{ERUE}_f$ are theorem equivalent.*

Proof. Analogous to Lemma 2. The additional or modified rules do not cause any problems.

Question 3 (Theorem Equivalence). The author claims that the calculi $\mathcal{ERUE}$ and $\mathcal{ERUE}_{fc}$ (or $\mathcal{ERUE}_f$) are theorem equivalent. A formal proof will most likely be analogous to questions 1 and 2.

7 Examples

The first (trivial FO) example illustrates the main ideas of $\mathcal{EP}$ and $\mathcal{ERUE}$: $a_\iota \in m_{\iota\to o} \wedge a = b \Rightarrow b \in m$. Sets are encoded as characteristic functions and $\in := \lambda X_\alpha, M_{\alpha\to o}.\, M\ X$, such that the negated problem normalises to: $\mathcal{C}_1 : [m\ a]^T$, $\mathcal{C}_2 : [a = b]^T$, $\mathcal{C}_3 : [m\ b]^F$. An obvious term-rewriting refutation in $\mathcal{EP}$: $Para(\mathcal{C}_1, \mathcal{C}_2)$, $Triv : \mathcal{C}_4 : [m\ b]^T$; $Res(\mathcal{C}_3, \mathcal{C}_4), Triv : \Box$.[15] A difference-reducing refutation in $\mathcal{ERUE}$: $Res(\mathcal{C}_1, \mathcal{C}_3) : \mathcal{C}_4 : [m\ a = m\ b]^F$; $Dec(\mathcal{C}_4), Triv : \mathcal{C}_5 : [a = b]^F$; $Res(\mathcal{C}_2, \mathcal{C}_5), Triv : \Box$.

We now examine the examples mentioned in Thm. 4 in calculus $\mathcal{EP}$: $\mathbf{E}_2^{Para} : [(G\ X_\iota) = p_{\iota\to o}]^T$ (Cantor's theorem) $Func'(\mathbf{E}_2^{Para}), Equiv' : C_1 : [G\ X\ Y_\iota]^F \vee$

[15] Notation (as already used before): $Res(\mathcal{C}_6, \mathcal{C}_5), Fac$ describes a paramodulation step between $\mathcal{C}_6$ and $\mathcal{C}_5$ followed by factorisation of the resulting clause. $Prim(\mathcal{C}_1|\mathcal{C}_2)$ denotes the parallel application of rule $Prim$ to $\mathcal{C}_j$ and $\mathcal{C}_k$.

$[p\ Y_\iota]^T,\ \mathcal{C}_2 : [G\ X\ Y_\iota]^T \vee [p\ Y_\iota]^F;\ Prim(\mathcal{C}_1|\mathcal{C}_2), Subst : \mathcal{C}_3 : [G'\ X\ Y]^T \vee [p\ Y]^T,\ \mathcal{C}_4 : [G''\ X\ Y]^F \vee [p\ Y]^F;\ Fac(\mathcal{C}_3|\mathcal{C}_4), \mathcal{UNI} : \mathcal{C}_5 : [p\ Y]^T,\ \mathcal{C}_6 : [p\ Y]^F;\ Res(\mathcal{C}_5, \mathcal{C}_6), \mathcal{UNI} : \mathcal{C}_7 : \Box$. $\mathbf{E}_1^{Para}$ and $\mathbf{E}_3^{Para}$ can be proven analogously. The key idea is to employ the positive extensionality rules first. As paramodulation rule is not employed, these proofs are obviously also possible in $\mathcal{ERUE}$.

Example $\mathbf{E}_2^{set}$ focuses on reasoning about sets: $(\{X|\ odd\ X \wedge num\ X\} = \{X|\ \neg\ ev\ X \wedge num\ X\}) \Rightarrow (2^{\{X|\ odd\ X \wedge X > 100 \wedge num\ X\}} = 2^{\{X|\ \neg\ ev\ X \wedge X > 100 \wedge num\ X\}})$, where the powerset-operator is defined by $\lambda N_{\alpha\to o}.\ \lambda M_{\alpha\to o}.\ \forall X_\alpha.\ \mathbf{M}\ X \Rightarrow \mathbf{N}\ X$. $\mathcal{CNF}(\mathbf{E}_2^{set}), Func, Func' : \mathcal{C}_1 : [(odd\ X \wedge num\ X) = (\neg\ ev\ X \wedge num\ X)]^T$ and $\mathcal{C}_2 : [(\forall X.\ n\ X \Rightarrow ((odd\ X \wedge X > 100) \wedge num\ X)) = (\forall X.\ n\ X \Rightarrow ((\neg\ ev\ X \wedge X > 100) \wedge num\ X))]^F$ where n is a Skolem constant. The reader may check that an application of rule *Para* does not lead to a successful refutation here as the terms in the powerset description do unfortunately not have the *right structure*. Instead of following the term-rewriting idea we have to proceed with difference-reduction and a recursive call to the overall refutation search from within the unification process: $Dec(\mathcal{C}_2), Triv, Func, Dec, Triv : \mathcal{C}_3 : [((odd\ s \wedge s > 100) \wedge num\ s) = ((\neg\ ev\ s \wedge s > 100) \wedge num\ s)]^F;\ Equiv(\mathcal{C}_3), \mathcal{CNF}, Fac, \mathcal{UNI} : \mathcal{C}_4 : [odd\ s]^T \vee [ev\ s]^F,\ \mathcal{C}_5 : [s > 100]^T,\ \mathcal{C}_6 : [num\ s]^T,\ \mathcal{C}_7 : [odd\ s]^F \vee [s > 100]^F \vee [num\ s]^F \vee [ev\ s]^T;\ Equiv'(\mathcal{C}_1), \mathcal{CNF}, Fac, \mathcal{UNI} : \mathcal{C}_8 : [odd\ X]^F \vee [num\ X]^F \vee [ev\ X]^F,\ \mathcal{C}_9 : [odd\ X]^T \vee [num\ X]^F \vee [ev\ X]^T$. The rest of the refutation is a straightforward resolution proof on $\mathcal{C}_4 - \mathcal{C}_9$. It is easy to check that an elegant term-rewriting proof is only possible if we put the succedent of $\mathbf{E}_2^{set}$ in the *right order*: $2^{\{X|\ (odd\ X \wedge num\ X) \wedge X > 100\}} = 2^{\{X|\ (\neg\ ev\ X \wedge num\ X) \wedge X > 100\}}$. Thus this example nicely illustrates the unavoidable mixed term-reducing and difference-reducing character of extensional higher-order paramodulation.

On the other hand a very interesting goal directed proof is possible within the RUE-resolution approach $\mathcal{ERUE}$ by immediately resolving between $\mathcal{C}_1$ and the unification constraint $\mathcal{C}_2$ and subsequently employing syntactical unification in connection with recursive calls to the overall refutation process (with the extensionality rules) when syntactical unification is blocked.

[Ben99] provides a more detailed discussion of these and additional examples.

8 Conclusion

We presented the two approaches $\mathcal{EP}$ and $\mathcal{ERUE}$ for extensional higher-order paramodulation and RUE-resolution which extend the extensional higher-order resolution approach $\mathcal{ER}$ [BK98a] by a primitive equality treatment. All three approaches avoid the extensionality axioms and employ more goal directed extensionality rules instead. An interesting difference to Huet's original constraint resolution approach [Hue72] is that eager (pre-)unification becomes essential and cannot be generally delayed if an extensionality treatment is required.

Henkin completeness has been proven for the slightly extended (by the additional rule *FlexFlex*) approaches $\mathcal{ER}_f$, $\mathcal{EP}_f$ and $\mathcal{ERUE}_f$. The claim that rule *FlexFlex* is admissible in them has not been proven yet. All three approaches can be implemented in a higher-order set of support approach as presented in [Ben99]. [Ben99] also presents some first ideas how the enormous search space

of the introduced approaches can be further restricted in practice, e.g. by introducing redundancy methods.

It has been motivated that some problems cannot be solved in the paramodulation approach $\mathcal{EP}$ by following the term-rewriting idea only, as they unavoidably require the application of the difference-reducing extensionality rules. In contrast to $\mathcal{EP}$ the difference-reducing calculus $\mathcal{ERUE}$ seems to harmonise quite well with the difference-reducing extensionality rules (or axioms), and thus this paper concludes with the question: Can HO adaptations of term-rewriting approaches be as successful as in FO, if one is interested in Henkin completeness and extensionality, e.g., when reasoning about sets, where sets are encoded as characteristic functions? Further work will be to examine this aspect with the help of the LEO-system [BK98b] and to investigate the open questions of this paper.

Acknowledgements. The work reported here was funded by the Deutsche Forschungsgemeinschaft under grant HOTEL. The author is grateful to M. Kohlhase, F. Pfenning, P. Andrews, V. Sorge and S. Autexier for stimulating discussions and support.

References

And71. P. B. Andrews. Resolution in type theory. *JSL*, 36(3):414–432, 1971.
And86. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.
Bar84. H. P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*. North-Holland, 2nd edition 1984.
Ben99. C. Benzmüller. Equality and Extensionality in Automated Higher-Order Theorem Proving. PhD thesis, FB 14, Universität des Saarlandes, 1999.
BK97. C. Benzmüller and M. Kohlhase. Model existence for higher-order logic. Seki-Report SR-97-09, FB 14, Universität des Saarlandes, 1997, submitted to JSL.
BK98a. C. Benzmüller and M. Kohlhase. Extensional higher-order resolution. In Kirchner and Kirchner [KK98], pages 56–72.
BK98b. C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [KK98], pages 139–144.
Chu40. A. Church. A formulation of the simple theory of types. *JSL*, 5:56–68, 1940.
Dar68. J. L. Darlington. Automatic theorem proving with equality substitutions and mathematical induction. *Machine Intelligence*, 3:113–130, 1968.
Dig79. V. J. Digricoli. Resolution by unification and equality. In W. H. Joyner, editor, *Proc. of the 4th Workshop on Automated Deduction*, Austin, 1979.
Hen50. L. Henkin. Completeness in the theory of types. *JSL*, 15(2):81–91, 1950.
Hue72. G. P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic*. PhD thesis, Case Western Reserve University, 1972.
KK98. C. Kirchner and H. Kirchner, editors. *Proc. of the 15th Conference on Automated Deduction*, number 1421 in LNAI, Springer, 1998.
Koh94. M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Universität des Saarlandes, 1994.
Mil83. D. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, 1983.
RW69. G. A. Robinson and L. Wos. Paramodulation and TP in first order theories with equality. *Machine Intelligence*, 4:135–150, 1969.
SG89. W. Snyder and J. H. Gallier. Higher-Order Unification Revisited: Complete Sets of Transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.
Smu63. Raymond M. Smullyan. A unifying principle for quantification theory. Proc. Nat. Acad Sciences, 49:828–832, 1963.
Wol93. D. Wolfram. *The Clausal Theory of Types*. Cambridge University Press, 1993.

Extensional Higher-Order Resolution

Christoph Benzmüller and Michael Kohlhase

Fachbereich Informatik, Universität des Saarlandes, Germany
chris|kohlhase@cs.uni-sb.de

Abstract. In this paper we present an extensional higher-order resolution calculus that is complete relative to Henkin model semantics. The treatment of the extensionality principles – necessary for the completeness result – by specialized (goal-directed) inference rules is of practical applicability, as an implentation of the calculus in the LEO-System shows. Furthermore, we prove the long-standing conjecture, that it is sufficient to restrict the order of primitive substitutions to the order of input formulae.

1 Introduction

The history of building automated theorem provers for higher-order logic is almost as old as the field of deduction systems itself. The first successful attempts to mechanize and implement higher-order logic were those of Huet [Hue73] and Jensen and Pietrzykowski [JP76]. They combine the resolution principle for higher-order logic (first studied in [And71]) with higher-order unification. The unification problem in typed λ-calculi is much more complex than that for first-order terms, since it has to take the theory of $\alpha\beta\eta$-equality into account. In particular the higher-order unification problem is undecidable and sets of solutions need not to have most general elements that represent them. Thus the calculi for higher-order logic have to take special measures to circumvent the problems posed by the theoretical complexity of higher-order unification.

Experiments like the TPS system [And89,ABI+96] (which uses a higher-order matings calculus) or our own LEO system [BK98,Ben97] (which uses a variant of Huet's resolution calculus [Hue73]) have shown the practical feasibility of higher-order automated theorem proving based on these ideas. Establishing completeness for higher-order calculi is more problematic than in first-order logic. The intuitive set-theoretic *standard semantics* cannot give a sensible notion of completeness, since it does not admit complete calculi [Göd31]. But there is a more general notion of semantics due to Henkin [Hen50] that allows complete calculi and therefore sets the standard for the deductive power of calculi.

The core of higher-order resolution ($\mathcal{HORES}$, see [Hue73,Koh94] for details) is a simple extension of the first-order resolution method to the higher-order language: the only significant difference is that $\beta\eta$-equality has to be build in by keeping formulae in normal form and that first-order unification has to be replaced by higher-order unification (i.e. unification with respect to the theory

C. Kirchner and H. Kirchner (Eds.): Automated Deduction, CADE-15
LNAI 1421, pp. 56–71, 1998.

of $\beta\eta$-equality). Since this is a semi-decidable search process itself, it cannot simply be used as a sub-procedure that is invoked during the application of the resolution or factoring rules. Rather resolution and factorization rules are modified, so that they record the induced unification problem in a unification constraint instead of trying to compute a complete set of unifiers. Furthermore, the calculus is augmented with the inference rules of higher-order unification that are lifted to act on the unification constraints of clauses. With this trick the search for empty clauses and that for higher-order unifiers are interleaved, which alleviates the undecidability problem.

Unfortunately, neither $\mathcal{HORES}$ nor the TPS procedure are complete with respect to Henkin semantics, since they fail to capture substitutivity of equivalence. In [Koh95], the first author has presented a higher-order tableau calculus that addresses the problem with a new inference rule that uses substitutivity of equivalence in a goal-oriented way, but still fails to capture functional extensionality of Leibniz equality.

For our extensional higher-order resolution calculus $\mathcal{ER}$ we extend higher-order resolution by ideas from [Koh95] and a suitable treatment of Leibniz equality and prove the resulting calculus sound and complete with respect to Henkin's general model semantics [Hen50]. Furthermore, we show that we can restrict the set of primitive substitutions that are necessary for flexible literals to a finite set.

Before we begin with the exposition, let us specify what we mean by "higher-order logic": any simply typed logical system that allows quantification over function variables. In this paper, we will employ a system $\mathcal{HOL}$, which is based on the simply typed λ-calculus; for an introduction see for instance [And86,Bar84].

2 Higher-Order Logic ($\mathcal{HOL}$)

The set $wff_\alpha(\Sigma)$ of well-formed formulae of type α is build up from the set $\mathcal{V}$ of variables, and the signature Σ (a set of typed constants) as applications and λ-abstractions. We will denote variables with upper-case letters ($X_\alpha, Y, Z, X^1_\beta, X^2_\gamma \ldots$), constants with lower-case letters ($c_\alpha, f_{\alpha\to\beta}, \ldots$), and well-formed formulae with upper-case bold letters ($\mathbf{A}_\alpha, \mathbf{B}, \mathbf{C}^i, \ldots$)[1]. Furthermore, we abbreviate multiple applications and abstractions in a kind of vector notation, so that $\mathbf{A}\overline{\mathbf{U}^k}$ denotes k-fold application (associating to the left) and $\lambda\overline{X^k}.\mathbf{A}$ denotes k-fold λ-abstraction (associating to the right) and use the square dot . as an abbreviation for a pair of brackets, where . stands for the left one with its partner as far to the right as is consistent with the bracketing already present in the formula.

We will use the terms like free and bound variables in their standard meaning and we use $\mathbf{Free}(\mathbf{A})$ for the set of free variables of a formula $\mathbf{A}$. In particular alphabetic change of names of bound variables is build into our $\mathcal{HOL}$: we consider alphabetic variants to be identical (viewing the actual representation as a representative of an alphabetic equivalence class) and use a notion of substitution that avoids variable capture, systematically renaming bound variables. We

[1] We will denote the types of formulae as indices, if it is not clear from the context.

could also have used de Bruijn's indices [dB72] as a concrete implementation of this approach at the syntax level.

By $wff_\alpha^{cl}(\Sigma) \subseteq wff_\alpha(\Sigma)$ we denote the set of all closed well-formed formulae, i.e. which contain no free variables and we call the members of $wff_o(\Sigma)$ sentences.

We denote a substitution that instantiates a variable X with a formula $\mathbf{A}$ with $[\mathbf{A}/X]$ and write $\sigma, [\mathbf{A}/X]$ for the substitution that is identical with σ but instantiates X with $\mathbf{A}$.

The structural equality relation of $\mathcal{HOL}$ is induced by $\beta\eta$-reduction

$$(\lambda X_\alpha.\mathbf{A})\mathbf{B} \longrightarrow_\beta [\mathbf{B}/X]\mathbf{A} \qquad\qquad (\lambda X_\alpha.\mathbf{C}X) \longrightarrow_\eta \mathbf{C}$$

where X is not free in $\mathbf{C}$. It is well-known, that the reduction relations β, η, and $\beta\eta$ are terminating and confluent, so that there are unique normal forms.

In $\mathcal{HOL}$, the set of base types is $\{o, \iota\}$ for truth values and individuals, and the signature Σ contains logical constants for negation $\neg_{o\to o}$, conjunction $\wedge_{o\to o\to o}$, and quantification[2] $\Pi^\alpha_{(\alpha\to o)\to o}$. All other constants are called parameters, since the argumentation in this paper is parametric in their choice[3].

It is matter of folklore that equality can directly be expressed in $\mathcal{HOL}$ e.g. by the *Leibniz definition*, so that a primitive notion of equality (expressed by a primitive constant $=$ in Σ) is not strictly needed; we will use this observation in this paper to treat equality as a defined notion. Leibniz equality defines two terms to be equal, iff they have the same properties. Hence equality can be defined as

$$\doteq^\alpha := \lambda X_\alpha.\lambda Y_\alpha.\forall P_{\alpha\to o}.PX \Rightarrow PY$$

A **standard model** for $\mathcal{HOL}$ provides a fixed set $\mathcal{D}_\iota$ of individuals, and a set $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ of truth values. All the domains for the complex types are defined inductively: $\mathcal{D}_{\alpha\to\beta}$ is the set of functions $f\colon \mathcal{D}_\alpha \to \mathcal{D}_\beta$. The evaluation $\mathcal{I}_\varphi$ with respect to an interpretation $\mathcal{I}\colon \Sigma \to \mathcal{D}$ of constants and an assignment φ of variables is obtained by the standard homomorphic construction that evaluates a λ-abstraction with a function, whose operational semantics is specified by β-reduction.

Henkin models only require that $\mathcal{D}_{\alpha\to\beta}$ has enough members that any well-formed formula can be evaluated[4]. Note that with this generalized notion of a model, there are less formulae that are valid in all models (intuitively, for any given formulae there are more possibilities for counter-models). Thus the generalization to Henkin models restricts the set of valid formulae sufficiently, so that all of them can be proven by the resolution calculus presented in this paper. For our completeness proofs, we will use the abstract consistency method first introduced by Raymond Smullyan in [Smu63] for first-order logic and later

[2] With this quantification constant, standard quantification of the form $\forall X_\alpha.\mathbf{A}$ can be regained as an abbreviation for $\Pi^\alpha(\lambda X_\alpha.\mathbf{A})$.

[3] 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 [And72].

[4] In other words: the functional universes are rich enough to satisfy the comprehension axioms.

extended to higher-order logic by Peter Andrews [And71]. The model existence theorem below is a variant of the latter for Henkin models. For the proof we refer to [BK97].

Theorem 1 (Henkin Model Existence). *Let Γ_Σ be a saturated abstract consistency class for Henkin models (see the definition below), and $\Phi \in \Gamma_\Sigma$, then there is a Henkin model $\mathcal{M}$ such that $\mathcal{M} \models \Phi$.*

Definition 1 (Abstract Consistency Class for Henkin Models). *We call a class Γ_Σ of sets of sentences an* **abstract consistency class for Henkin Models**, *iff Γ_Σ is closed under subsets and such that for all sets $\Phi \in \Gamma_\Sigma$ (we use $\Phi * \mathbf{A}$ as an appreviation for $\Phi \cup \{\mathbf{A}\}$):*

∇_c *If $\mathbf{A}$ is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.*
$\nabla_\neg$ *If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.*
$\nabla_{\beta\eta}$ *If $\mathbf{A} \in \Phi$ and $\mathbf{B}$ is the $\beta\eta$-normal form of $\mathbf{A}$, then $\mathbf{B} * \Phi \in \Gamma_\Sigma$.*
$\nabla_\vee$ *If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.*
$\nabla_\wedge$ *If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.*
$\nabla_\forall$ *If $\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi * \mathbf{FG} \in \Gamma_\Sigma$ for each $\mathbf{G} \in wff^{cl}_\alpha(\Sigma)$.*
$\nabla_\exists$ *If $\neg\Pi^\alpha\mathbf{F} \in \Phi$, then $\Phi * \neg(\mathbf{F}w) \in \Gamma_\Sigma$ for a fresh parameter $w_\alpha \in \Omega_\alpha$.*
∇_b *If $\neg(\mathbf{A} \doteq^o \mathbf{B}) \in \Phi$, then $\Phi \cup \{\mathbf{A}, \neg\mathbf{B}\} \in \Gamma_\Sigma$ or $\Phi \cup \{\neg\mathbf{A}, \mathbf{B}\} \in \Gamma_\Sigma$.*
∇_q *If $\neg(\mathbf{F} \doteq^{\alpha\to\beta} \mathbf{G}) \in \Phi$, then $\Phi * \neg(\mathbf{F}w \doteq^\beta \mathbf{G}w) \in \Gamma_\Sigma$ for a fresh parameter $w_\alpha \in \Omega_\alpha$.*

We will call Γ_Σ **saturated**, *iff for all sentences $\mathbf{A} \in wff_o(\Sigma)$ we have $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \neg\mathbf{A} \in \Gamma_\Sigma$.*

Remark 1 (Counterparts for ∇_b,∇_q). In Definition 1 positive counterparts for the two conditions ∇_b,∇_q are not needed, since these conditions are automatically met (note that $\doteq$ is a defined construct). For details see [BK97].

In this paper the *extensionality principles* will play a major role. These formalize fundamental mathematical intuitions about functions and truth values. The **functional extensionality principle** says, that two functions are equal, iff they are equal on all arguments. This principle can be formulated by the following schematic λ-term:

$$\forall M_{\alpha\to\beta}.\forall N_{\alpha\to\beta}.(\forall X_\alpha.(MX) \doteq (NX)) \equiv (M \doteq N)$$

The **extensionality principle for truth values** states that on the set of truth values equality and equivalence relation coincide: $\forall P_o.\forall Q_o.(P \doteq Q) \equiv (P \equiv Q)$. Note that in Henkin models both extensionality principles are valid and that Leibniz equality indeed denotes equality relation (see [BK97] for details).

3 The Calculus $\mathcal{ER}$

Now we introduce the higher-order resolution calculus $\mathcal{ER}$. Therefore we will review standard higher-order resolution $\mathcal{HORES}$ and use the extensionality principles to discuss why it is not complete. From the deficiencies we will develop the necessary extensions and give an intuition by exhibiting refutations that become possible.

$\mathcal{HORES}$ is a refutation calculus that manipulates sets of *clauses*, i.e. sets (which we will represent as disjunctions) of literals (e.g. $\mathcal{C} := [q_{\alpha\to o}X_\alpha]^T \vee [p_{\alpha\to o}X_\alpha]^F \vee [c_\alpha = X_\alpha]^F$).

Definition 2 (Literal). *Literals are atomic propositions labeled with an intended truth value. We call a literal a* unification constraint*, iff it is negative (i.e. annotated by the truth value F) and the head is =, all the others we call* **proper literals**. *Clauses existing entirely of unification constraints are called* almost empty*. Since instantiation of a head variable will convert a literal into a general labeled propositions, we will sometimes call these* **pre-literals**.

Clause normalization is very similar to the first-order case, except for the treatment of existential quantification. Therefore, we will not present the transformation rules here, but simply discuss the differences and assume that each given higher-order proof problem $\mathcal{P}$ can be transformed into a set of clauses CNF($\mathcal{P}$). A naive treatment with Skolemization results in a calculus that is not sound with respect to Henkin models, since Skolem functions are special choice functions[5], 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. Skolemization becomes sound, if any Skolem function f^n only occurs in a *Skolem term*, i.e. a formula $\mathbf{S} = f^n\overline{\mathbf{A}^n}$, where none of the $\mathbf{A}^i$ contains a bound variable. Thus the Skolem terms only serve as descriptions of the existential witnesses and never appear as functions proper. When we speak of a **Skolem term $\mathbf{S}_\alpha$ for a clause** C, where $\{X^1_{\alpha^1}\cdots X^n_{\alpha^n}\}$ is the set of free variables occurring in C, then $\mathbf{S}_\alpha$ is an abbreviation for the term $(f^n_{\alpha^1\to\cdots\to\alpha^n\to\alpha}X^1\cdots X^n)$, where f is a new constant from $\mathcal{C}_{\alpha^1\to\cdots\to\alpha^n\to\alpha}$ and n specifies the number of necessary arguments for f.

Remark 2 (Leibniz Equality). We assume that before applying clause normalization each primitive equality symbol is replaced by its corresponding Leibniz definition. Hence after normalizing a given input problem, the resulting clause set does not contain any equality symbol. However, during the refutation process, equality symbols may be introduced again as we code unification constraints by negated equation literals.

3.1 Higher-Order Unification in $\mathcal{ER}$

Higher-order unification is a process of recursive deterministic simplification (rules α, η, *Dec*, *Triv*, and *Subst* in figure 1) and non-deterministic variable binding (rule *Flex/Rigid*). The rules α and η are licensed by the functional

[5] They choose an existential witness from the set of possible witnesses for an existential formula.

$$\frac{C \vee [(\lambda X_\alpha . \mathbf{A}) = (\lambda Y_\alpha . \mathbf{B})]^F \qquad s_\alpha \text{ Skolem term for this clause}}{C \vee [[s/X]\mathbf{A} = [s/Y]\mathbf{B}]^F}\ \alpha$$

$$\frac{C \vee [(\lambda X_\alpha . \mathbf{A}) = \mathbf{B}]^F \qquad s_\alpha \text{ Skolem term for this clause}}{C \vee [[s/X]\mathbf{A} = (\mathbf{B}s)]^F}\ \eta$$

$$\frac{C \vee [h\overline{\mathbf{U}^n} = h\overline{\mathbf{V}^n}]^F}{C \vee [\mathbf{U}^1 = \mathbf{V}^1]^F \vee \ldots \vee [\mathbf{U}^n = \mathbf{V}^n]^F}\ Dec \qquad \frac{C \vee [\mathbf{A} = \mathbf{A}]^F}{C}\ Triv$$

$$\frac{C \vee E \quad E \text{ solved for } C}{\mathrm{CNF}(\mathrm{subst}_E(C))}\ Subst$$

$$\frac{C \vee [F_\gamma \overline{\mathbf{U}^n} = h\overline{\mathbf{V}}]^F \quad \mathbf{G} \in \mathcal{GB}^h_\gamma}{C \vee [F = \mathbf{G}]^F \vee [F\overline{\mathbf{U}} = h\overline{\mathbf{V}}]^F}\ Flex/Rigid$$

Fig. 1. Lifted Higher-Order (pre-)unification rules

extensionality principle and eliminate the top λ-binder in unification constraints of functional type. The Skolem term s_α is an existential witness for the fact that the functions are different. Since clauses are implicitly universally quantified, this witness may depend on the values of all free variables occurring in the clauses, so it must be a Skolem term for this clause. Decomposition (rule *Dec*) is analogous to the first-order case and the rule *Triv* allows to remove reflexivity pairs. Rule *Dec* will be discussed again in connection with the extensionality rules in section 3.3.

The rule *Subst* eliminates variables that are solved in a clause: we call a unification constraint $U := [X_\alpha = \mathbf{N}_\alpha]^F$ or $U := [\mathbf{N}_\alpha = X_\alpha]^F$ **solved** iff X_α is not free in $\mathbf{N}_\alpha$. In this case X is called the **solved variable** of U. Let $C := L^1 \vee \cdots \vee L^n \vee U^1 \vee \cdots \vee U^m$ be a clause with unification constraints $U^1 \vee \cdots \vee U^m$ $(1 \leq m)$. Then a disjunction $U^{i_1} \vee \cdots \vee U^{i_k}$ $(i_j \in \{1, \cdots, m\}; 1 \leq j \leq k)$ of solved unification constraints occurring in C is called **solved for** C iff for every $U^{i_j} (1 \leq j \leq k)$ holds: the solved variable of U^{i_j} does not occur free in any of the U^{i_l} for $l \neq j; 1 \leq l \leq k$. Note that each solved set of unification constraints E for a clause C can be associated with a substitution subst_E which is the most general unifier of E. Thus the rule *Subst* essentially propagates the information from the unification constraints to the proper clause parts. Since the instantiation of flexible literals (i.e. literals, where the head is a free variable) may result in pre-literals, the result of this propagation may cease to be a clause, therefore it needs to be reduced to clause normal form.

Remark 3 (Eager Unification). The set of rules described up to now is terminating and confluent, so that higher-order unification applies it eagerly to filter

out all clauses with an unsolvable unification constraint[6]. It leads to unification constraints, where both sides are applications and where at least one side is flexible, i.e. where the head is a variable. In this case, the higher-order unification problem can be reduced to the problem of finding most general formulae of a given type and a given head symbol.

Definition 3 (General Binding). *Let $\alpha = (\overline{\beta^l} \to \gamma)$, and h be a constant or variable of type $(\overline{\delta_m} \to \gamma)$ in Γ, then $\mathbf{G} := \lambda\overline{X^l_{\beta^l}}.h\overline{\mathbf{V}^m}$ is called a* **general binding of type** α **and head** h*, if $\mathbf{V}^i = H^i\overline{X^l_{\beta_l}}$. The H^i are new variables of types $\overline{\beta^l} \to \delta^i]$. It is easy to show that general bindings indeed have the type and head claimed in the name and are most general in the class of all such terms.*

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

Since there are only finitely many general bindings (one imitation binding and at most l projection bindings) the *Flex/Rigid* rule is finitely branching. We never have to consider the so-called *Flex/Flex* literals[7], since *Flex/Flex* equations can always be solved by instantiating the head variables with suitable constant functions that absorb their arguments. This observation is due to Gérard Huet [Hue73] and defines higher-order pre-unification, a computationally more feasible (but still undecidable) variant of higher-order unification. However, even if *Flex/Flex* pairs are solvable, we cannot simply delete them like trivial pairs, since one or both of the heads may be instantiated making the term rigid, so that the pair has to be subject to pre-unification again.

3.2 Higher-Order Resolution

Definition 4 (Higher-Order Resolution). *The* **higher-order resolution calculus** $\mathcal{HORES}$ *consists of the inference rules in figure 2 together with the unification rules in figure 1. We call a clause* **empty***, iff it consists entirely of Flex/Flex unification constraints and say hat a $\mathcal{HORES}$-derivation of an empty clause from a set Φ of clauses is a* **refutation** *of Φ. For a sentence $\mathbf{A}_o$ we call a refutation of CNF($\neg\mathbf{A}$) a refutation for $\mathbf{A}$.*

As in first-order we have resolution and factorization rules *Res* and *Fac*. But instead of solving the unification problems immediately within a rule application we delay their solution and incorporate them explicitly as unification constraints in the resulting clauses. Note that the resolution rule as well as the factorization rule are allowed to operate on unification constraints.

[6] As we will see later this solution is too strong if we want to be complete in Henkin models since an unsolvable unification constraint might be solvable by using the extensionality rules.

[7] For a refutation, we do not need to enumerate all unifiers for a given unification problem but to seek for one possible instantiation of a given problem which leads to the contradiction.

$$\frac{[\mathbf{N}]^\alpha \vee C \quad [\mathbf{M}]^\beta \vee D \quad \alpha \neq \beta}{C \vee D \vee [\mathbf{N} = \mathbf{M}]^F}\ Res \qquad \frac{[\mathbf{N}]^\alpha \vee [\mathbf{M}]^\alpha \vee C \quad \alpha \in \{T, F\}}{[\mathbf{N}]^\alpha \vee C \vee [\mathbf{N} \ = \mathbf{M}]^F}\ Fac$$

$$\frac{[Q_\gamma \overline{\mathbf{U}^k}]^\alpha \vee C \quad \mathbf{P} \in \mathcal{GB}_\gamma^{\{\neg,\vee\} \cup \{\Pi^\beta | \beta \in \mathcal{T}^k\}}}{[Q_\gamma \overline{\mathbf{U}^k}]^\alpha \vee C \vee [Q = \mathbf{P}]^F}\ Prim^k$$

Fig. 2. Higher-order resolution rules

To find a refutation for a given problem we may have to instantiate the head variables of flexible literals by material that contains logical constants. Unfortunately these instantiations cannot be generated by the unification rules, since all logical constants have been eliminated from the clause set by normalization, thus they enter the refutation by unification. Therefore the rule *Prim* allows to instantiate head variables Q_γ by general bindings $\mathbf{P}$ of type γ and head in $\{\neg, \vee\} \cup \{\Pi^\beta | \beta \in \mathcal{T}\}$. Thus the necessary logical constants are introduced into the refutation one by one, hence the name *primitive substitutions*.

For instance the sentence $\mathbf{A} := \exists X_{o\bullet} X$ is valid in all Henkin models, but $\mathrm{CNF}(\neg\mathbf{A}) = \{[X]^F\}$ cannot be refuted without some kind of a primitive substitution rule, since none of the other rules apply. With *Prim*, we can deduce $[X]^F \vee [X = \neg H]^F$ and then $[Y]^T$ by *Subst*. These two unit literals can be resolved to $[X = Y]^F$, which is an empty clauses, since $[X = Y]^F$ is a *Flex/Flex* unification constraint.

The primitive substitution rules have originally been introduced by Peter Andrews in [And89] (Gérard Huet uses a set of so-called "splitting rules" for the same purpose in [Hue73]). Note that the set of general bindings is infinite, since we need one for every quantifier Π^α and the set of types is infinite. Thus in contrast to the goal-directed search for instantiations in unification, the rule *Prim* performs blind search and even worse, is infinitely branching. Therefore, the problem of finding instantiations for predicate variables is conceived as the limiting factor to higher-order automated theorem proving.

It has been a long-standing conjecture that in machine-oriented calculi it is sufficient to restrict the order of primitive quantifier substitutions to the order of the input formulae. In [BK97], we have established a finer-grained variant of theorem 1 that we can use as a basis to prove this conjecture. Let us now introduce the necessary definitions.

Definition 5 (Order). *For a type $\alpha \in \mathcal{T}$, we define the* **order** $\mathbf{ord}(\alpha)$ *of α as* $\mathbf{ord}(\iota) = \mathbf{ord}(o) = 0$, *and* $\mathbf{ord}(\alpha \to \beta) = \max\{\mathbf{ord}(\alpha), \mathbf{ord}(\beta)\} + 1$. *Note that the set $\mathcal{T}^k = \{\alpha \in \mathcal{T} \mid \mathbf{ord}(\alpha) \leq k\}$ is finite for any order k. We will take the order of a formula to be the highest order of any type of any of its subterms, and the order of a set of formulae to be the maximum of the orders of its members.*

Theorem 2 (Model Existence with Order). *The model existence theorem holds even if we weaken the condition $\nabla_\forall$ of an abstract consistency class to*

$\nabla_\forall^k$ *If* $\Pi^\alpha \mathbf{F} \in \Phi$, *then* $\Phi * \mathbf{FG} \in \Gamma_\Sigma$ *for each* $\mathbf{G} \in wff_\alpha^{cl}(\Sigma)$ *with* $\mathbf{ord}(\mathbf{G}) \leq \mathbf{ord}(\Phi)$.

In [BK97] we establish this theorem for arbitrary well-founded orderings on types such that $\mathbf{ord}(\alpha), \mathbf{ord}(\beta) \leq \mathbf{ord}(\alpha \to \beta)$. This allows us to restrict instantiation in $\mathcal{ER}$ to formulae of the order of the input formulae. Note that this only effects the primitive substitution rule, since all other instantiations are performed by unification, which is order-restricted by construction. In particular, the non-standard definition of order above ensures finite branching of the primitive substitution rule. This ordering, that takes the lengths of argument lists into account leads to an increased order of the input set compared to the standard definition of order ($\mathbf{ord}(\overline{\alpha_n} \to \beta) = \max_n\{\alpha_i\} + 1$) and effectively restricts the number of necessary instantiations.

Our result justifies the practice of higher-order theorem provers to restrict the search for primitive substitutions and gives a road-map towards complete procedures. Of course there is still a lot of room for experimentation with the respective orderings.

3.3 Extensionality

The higher-order resolution calculus $\mathcal{HORES}$ defined above is not complete with respect to Henkin models, as the following example will show.

Example 1. The following formulae E1-E5[8] are not provable in $\mathcal{HORES}$ without using additional axioms for functional extensionality and/or extensionality on truth values.

E1 $a_o \equiv b_o \Rightarrow (\forall P_{o\to o}. Pa \Rightarrow Pb)$
This is the non-trivial direction of the extensionality property for truth values: if a_o is equivalent to b_o then a_o is equal to b_o ($a_o \equiv b_o \Rightarrow a = b$).

E2 $\forall P_{o\to o}. P(a_o \wedge b_o) \Rightarrow P(b \wedge a)$.
Any property which holds for $a \wedge b$ also holds for $b \wedge a$ (or simply that $a \wedge b = b \wedge a$).

E3 $(p_{o\to o} a_o \wedge p b_o) \Rightarrow p(b \wedge a)$
In other words, an arbitrary property $p_{o\to o}$ which coincidently holds for a_o and b_o also holds for their conjunction.

E4 $(\forall X_\iota. \forall P_{\iota\to o}. (P(m_{\iota\to\iota} X) \Rightarrow P(n_{\iota\to\iota} X))) \Rightarrow (\forall Q_{(\iota\to\iota)\to o}. Q(\lambda X_\iota. mX) \Rightarrow Q(\lambda X_\iota. nX))$
This formula can be interpreted as an instance of the ξ-rule $(\forall X_\iota. m_{\iota\to\iota} X = n_{\iota\to\iota} X) \Rightarrow (\lambda X_\iota. mX) = (\lambda X_\iota. nX)$ (See for instance [Bar84]).

E5 $(\forall X_\iota. \forall P_{\iota\to o}. P(m_{\iota\to\iota} X) \Rightarrow P(n_{\iota\to\iota} X)) \Rightarrow (\forall Q_{(\iota\to\iota)\to o}. Qm \Rightarrow Qn)$
This is an instance of the non-trivial direction of the functional extensionality axiom for type $\iota \to \iota$: $(\forall X_\iota. (m_{\iota\to\iota} X) = (n_{\iota\to\iota} X) \Rightarrow m = n)$.

[8] In Problems E1, E2, E4, and E5 we have used Leibniz definition of equality to remove the intuitive equality symbols.

$$\frac{C \vee [\mathbf{M}_o = \mathbf{N}_o]^F}{\mathrm{CNF}(C \vee [\mathbf{M}_o \equiv \mathbf{N}_o]^F)}\ \mathit{Equiv} \qquad \frac{C \vee [\mathbf{M}_\alpha = \mathbf{N}_\alpha]^F \quad \alpha \in \{o, \iota\}}{\mathrm{CNF}(C \vee [\forall P_{\alpha\to o}.PM \Rightarrow PN]^F)}\ \mathit{Leib}$$

$$\frac{C \vee [\mathbf{M}_{\alpha\to\beta} = \mathbf{N}_{\alpha\to\beta}]^F \quad s_\alpha \text{ Skolem term for this clause}}{C \vee [\mathbf{M}s = \mathbf{N}s]^F}\ \mathit{Func}$$

Fig. 3. Extensionality rules

For a proof of **E1** note that the clause normal form of the succedent consists of the two unit clauses $[p^0 a]^F$ and $[p^0 b]^T$, where p^0 is the Skolem constant for the variable P. These can be resolved upon to obtain the clause $[p^0 a = p^0 b]^F$, which can be decomposed to $[a_o = b_o]^F$. Obviously, this unification constraint cannot be solved by higher-order unification, and hence the refutation fails. In this situation, we need the principle of extensionality on truth values, which allows to replace each negated equality on type o by an equivalence. This leads to the clause normal form of $[a_o \equiv b_o]^F$, which contradicts the antecedent of **E1** and finally gives us the refutation.

Similar investigations show that the other examples cannot be proven by $\mathcal{HORES}$ too.

Our aim is to find an extension of $\mathcal{HORES}$, which is both Henkin-complete and adequate for an implementation. Surely, the introduction of axioms for the extensionality principles can solve the completeness problem in theory, but this will lead to an explosion of the search space which has to be avoided in practice. In particular, we do not change the purely negative spirit of the resolution calculus by introducing axioms but introduce special inference rules.

Definition 6 (Extensional Higher-Order Resolution). *The* **extensional higher-order resolution** *calculus $\mathcal{ER}$ is $\mathcal{HORES}$ extended with the inference rules in figure 3.*

The Rule *Leib* instantiates the equality symbol by its Leibniz definition and applies clause normalization. Rule *Equiv* is directly motivated by the proof attempt of **E1** discussed in example 1. Thus rule *Equiv* reflects the extensionality property for truth values but in a negative way: if two formulas are not equal then they are also not equivalent. Rule *Func* does the same for functional extensionality: if two functions are not equal then there exists an argument s_α on which these functions differ. To ensure soundness s_α has to be a new Skolem term which contains all the free variables occurring in the given clause.

The new rules strongly connect the unification part of our calculus with the resolution part. In some sense, they make the unification part extensional, since they allow to modify unification problems, which are not solvable by preunification alone in an extensional appropriate way and to translate them back into usual literals, such that we can try to find the right argumentation for

the solvability of the unification constraints in the general refutation process by possibly respecting the additionally given clauses in the search space.

Remark 4 (Rule Func). Note that we have already introduced two rules – α and η in unification (see figure 1) – which are very similar to this one. In fact we can restrict rule *Func* to the case were **N** and **M** are non-abstractions or vice-versa, we can remove the α and η rules from simplification as they are subsumed by the rule *Func* as purely type-based and apply β-reduction to both sides of the modified unification constraint.

Remark 5 (Unification Constraints). We have lifted the unification constraints to clause level by coding them into negated equation literals. Hence the question arises whether or not resolution and factorization rules are allowed to be applied on these unification constraints. In order to obtain a Henkin complete calculus this is not necessary – as our completeness proof shows – if we add the three extensionality rules discussed in the next subsection. Consequently the unification constraints do not necessarily have to be coded as negative equation literals, any other form will work as well.

The coding of unification constraints as negated equation literals becomes important if one considers an alternative version of extensional higher order resolution – which we will also motivate below –, where the rule *Leib* is avoided.

Note 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 experiments show with the LEO theorem prover [BK98,Ben97] – than the extensionality axioms, which introduce lots of flexible literals in the refutation process.

3.4 Examples

We now demonstrate the idea of the extensional resolution calculus on examples **E3** and **E5**:

E3 $\forall P_{o\to o}.(Pa_o \wedge Pb_o) \Rightarrow P(a \wedge b)$

CNF($\neg$**E3**) ($p_{o\to o}$ is a new Skolem constant):

	c1: $[pa]^T$	*c2:* $[pb]^F$	*c3:* $[p(a \wedge b)]^F$
Res(*c3*,*c1*):	*c4:* $[p(a \wedge b) = pa]^F$		
Res(*c3*,*c2*):	*c5:* $[p(a \wedge b) = pb]^F$		
Dec(*c4*):	*c6:* $[(a \wedge b) = a]^F$		
Dec(*c5*):	*c7:* $[(a \wedge b) = b]^F$		
Equiv(*c6*):	*c8:* $[a]^F \vee [b]^F$	*c9:* $[a]^T \vee [b]^T$	*c10:* $[a]^T$
Equiv(*c7*):	*c11:* $[a]^F \vee [b]^F$	*c12:* $[a]^T \vee [b]^T$	*c13:* $[b]^T$

The rest is obvious: Resolve *c10* and *c13* against *c8* (or *c11*). □

E5 $(\forall X_\iota . \forall P_{\iota\to o} . P(m_{\iota\to\iota} X) \Rightarrow P(n_{\iota\to\iota} X)) \Rightarrow (\forall Q_{(\iota\to\iota)\to o} . Qm \Rightarrow Qn)$

CNF($\neg$**E5**) (q is a new Skolem constant):

c1: $[P(mX)]^F \vee [P(nX)]^T$ *c2:* $[qm]^T$ *c3:* $[qn]^F$

Res(*c2*,*c3*): *c4*: $[qm = qn]^F$

Dec(*c4*): *c5*: $[m = n]^F$

Func(*c5*) (s_ι is a new Skolem constant): *c6*: $[ms = ns]^F$

Leib(*c6*) ($p_{\iota\to o}$ is a new Skolem constant): *c7*: $[p(ms)]^T$ *c8*: $[p(ns)]^F$

Note that resolving *c2* and *c3* immediately against *c1* does not lead to a solvable unification constraint. Instead we made a detour to the pre-unification part of the calculus and modified the clauses *c2* and *c3* in an extensionally appropriate way. Now *c2* and *c3* have their counterparts in *c7* and *c8*, but in contrast to *c2* and *c3* the new clauses can successfully be resolved against *c1*. □

The proofs of the other examples are discussed in [Ben97].

Remark 6 (Optimization of Extensionality). Note the order in which the extensionality rules were applied in the examples above. For a practical implementation these examples suggest the following **extensionality treatment** of unification constraints: First decompose the unification constraint as much as possible. Then use rule *Func* to add as many arguments as possible to both hand sides of the resulting unification constraints. And last use rule *Leib* and/or *Equiv* to finish the extensionality treatment. In this sense the above rules can be combined to form only one rule *Ext-Treat*.

Remark 7 (Rule Leib). Due to an idea of Frank Pfenning every refutation which uses rule *Leib* can possibly be done without this rule by resolving against the extensional modified unification constraint instead, and hence rule *Leib* may be superfluous. For example the application of rule *Leib* in the proof of example **E5** can be replaced by an immediate resolution step between clause *c1* and *c6*: *c7*: $[P(mX)]^F \vee [P(nX) = (ms = ns)]^F$. And by pre-unification ($P \leftarrow \lambda Y_\iota . (ms = Y)$ and $X \leftarrow s$) we immediately get the empty clause. Note that in this case it is essential that unification constraints are encoded as negative equality literals (see Remark 5).

However, there are two reasons why rule *Leib* seems to be very appropriate. First the completeness proof with respect to Henkin models seems to be more complicated without rule *Leib* and isn't done yet. Additionally the experience from the implementation work of the system LEO is, that rule *Func* eases the implementation and the integration of heuristics. See [Ben97] for a more detailed discussion.

4 Soundness and Completeness

Theorem 3 (Soundness of $\mathcal{ER}$). *The calculus $\mathcal{ER}$ is sound with respect to Henkin semantics.*

Proof. The soundness of $\mathcal{HORES}$ is discussed in detail in [Koh94], the only major difference to the first-order case is the treatment of Skolemization, which has been discussed in [Mil83].

The soundness of the three new extensionality rules are obvious, as they do only apply the two extensionality principles and the Leibniz definition, which are valid in Henkin models.

For the completeness result, we will need a series of disjunction Lemmata, which are well-known for first-order logic, and which can be proven with the same techniques, only considering the extra inference rules of $\mathcal{ER}$ in the inductions.

Lemma 1. *Let $\Phi, \Delta, \Gamma_1, \Gamma_2 \subseteq \mathit{wff}^{cl}(\Sigma)$ and $\mathbf{A}, \mathbf{B} \in \mathit{wff}^{cl}(\Sigma)$. We have*

1. *If $\mathit{CNF}(\Phi * \mathbf{A}) \vdash_{\mathcal{ER}} \square$ and $\mathit{CNF}(\Phi * \mathbf{B}) \vdash_{\mathcal{ER}} \square$, then $\mathit{CNF}(\Phi * \mathbf{A} \vee \mathbf{B}) \vdash_{\mathcal{ER}} \square$*
2. *If $\mathit{CNF}(\Phi * \neg\mathbf{A} * \mathbf{B}) \vdash_{\mathcal{ER}} \square$ and $\mathit{CNF}(\Phi * \mathbf{A} * \neg\mathbf{B}) \vdash_{\mathcal{ER}} \square$, then $\mathit{CNF}(\Phi * \neg(\mathbf{A} \equiv \mathbf{B})) \vdash_{\mathcal{ER}} \square$*

Proof. For the proof of the first assertion we first verify that $\mathrm{CNF}(\Phi * \mathbf{A} \vee \mathbf{B}) = \mathrm{CNF}(\Phi) \cup \mathrm{CNF}(\mathbf{A}) \sqcup \mathrm{CNF}(\mathbf{B})$, where $\Gamma \sqcup \Delta = := \{\mathbf{C} \vee \mathbf{D} | \mathbf{C} \in \mathrm{CNF}(A)\}, \mathbf{D} \in \mathrm{CNF}(B)\}$. Then we use that $\Phi \cup \Gamma_1 \sqcup \Gamma_2 \vdash_{\mathcal{ER}} \square$, provided that $\Phi \cup \Gamma_1 \vdash_{\mathcal{ER}} \square$ and $\Phi \cup \Gamma_2 \vdash_{\mathcal{ER}} \square$. The second involves a tedious but straightforward calculation.

Lemma 2 (Lifting Lemma). *Let Φ be a set of clauses and σ a substitution, then Φ is refutable by $\mathcal{ER}$, provided that $\theta(\Phi)$ is.*

Proof. The claim is proven by an induction on the structure of the refutation $\mathcal{D}_\theta : \theta(\Phi) \vdash_{\mathcal{ER}} \square$ be a refutation of $\theta(\Phi)$ constructing a refutation $\mathcal{D}$ for Φ that is isomorphic to $\mathcal{D}_\theta$.

For this task it is crucial to maintain a tight correspondence $\omega : \Phi \longrightarrow \theta(\Phi)$ between the respective clause sets. This is formalized by a **clause set isomorphism**, i.e. a bijection of clause sets, that corresponding clauses are isomorphic, i.e. for a ω respects literal polarities and is compatible with θ, i.e. for any literal $\mathbf{N}^\alpha$ we have $\omega(\mathbf{N}) = \theta(\mathbf{N})$. The main difficulty with lifting properties in higher-order logic is the fact that due to the existence of predicate variables at the head of formulae, the propositional structure of formulae can change during instantiation. For instance if $\theta(F) = \lambda X_\alpha . GX \vee p$, and $\mathbf{A}^\mathtt{T} = Fa^\mathtt{T}$, then the pre-literal $\theta(F)$ is split $\mathcal{D}_\theta$ but not in the $\mathcal{ER}$-derivation already constructed. The solution of this problem is to apply the rule *Prim* with a suitable general binding $\mathcal{G}^\vee_{\alpha\to o} = \lambda X_\alpha . (H^1 X) \vee (H^2 X)$ and obtain a pre-literal $(H^1 a \vee H^2 a)^\mathtt{T}$, to which can be split in order to regain a clause set isomorphism. Since $\mathcal{G}^\vee_{\alpha\to o}$ is more general than $\theta(F)$ there is a substitution ρ, such that $\theta(F) = \rho(\mathcal{G}^\vee_{\alpha\to o})$, therefore $\omega((H^1 a \vee H^2 a)^\mathtt{T}) = \theta'((H^1 a \vee H^2 a)^\mathtt{T})$ where $\theta' = \theta \cup \rho$.

Theorem 4 (Completeness of $\mathcal{ER}$). *The calculus $\mathcal{ER}$ is complete with respect to Henkin semantics.*

Proof. Let Γ_Σ be the set of Σ-sentences which cannot be refuted by calculus $\mathcal{ER}$ ($\Gamma_\Sigma := \{\Phi \subseteq wff_o^{cl}(\Sigma) | \mathrm{CNF}(\Phi) \nvdash_{\mathcal{ER}} \square\}$), then we show that Γ_Σ is a saturated abstract consistency class for Henkin models. This entails completeness of $\mathcal{ER}$ by theorem 1.

Let $\Phi \in \Gamma_\Sigma$. We show that Φ mets the conditions required in definition 1:

∇_c Suppose that $\mathbf{A}, \neg\mathbf{A} \in \Phi$. Since $\mathbf{A}$ is atomic we have $\mathrm{CNF}(\Phi * \mathbf{A} * \neg\mathbf{A}) = \mathrm{CNF}(\Phi) * [\mathbf{A}]^T * [\mathbf{A}]^F$ and hence we can derive $\square$ with *Res* and $Triv$. This contradicts our assumption.

In all of the remaining cases, we show the contrapositive, e.g. in the next case we prove, that for all $\Phi \in \Gamma_\Sigma$, if $\Phi * \neg\neg\mathbf{A} * \mathbf{A} \notin \Gamma_\Sigma$, then $\Phi * \neg\neg\mathbf{A} \notin \Gamma_\Sigma$, which entails the assertion.

$\nabla_\neg$ If $\mathrm{CNF}(\Phi * \neg\neg\mathbf{A} * \mathbf{A}) \vdash_{\mathcal{ER}} \square$, then also $\mathrm{CNF}(\Phi * \neg\neg\mathbf{A}) \vdash_{\mathcal{ER}} \square$, since $\mathrm{CNF}(\Phi * \neg\neg\mathbf{A} * \mathbf{A}) = \mathrm{CNF}(\Phi * \neg\neg\mathbf{A})$.

$\nabla_{\beta\eta}$ Analog to $\nabla_\neg$, since $\mathrm{CNF}(\Phi * \mathbf{A} * \mathbf{A}_{\downarrow_{\beta\eta}}) = \mathrm{CNF}(\Phi * \mathbf{A})$.

$\nabla_\vee$ If $\mathrm{CNF}(\Phi * \mathbf{A} \vee \mathbf{B} * \mathbf{A}) \vdash_{\mathcal{ER}} \square$ and $\mathrm{CNF}(\Phi * \mathbf{A} \vee \mathbf{B} * \mathbf{B}) \vdash_{\mathcal{ER}} \square$, then $\mathrm{CNF}(\Phi * \mathbf{A} \vee \mathbf{B}) \vdash_{\mathcal{ER}} \square$ by lemma 1(3).

$\nabla_\wedge$ If $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \vee \mathbf{B}) * \neg\mathbf{A} * \neg\mathbf{B}) \vdash_{\mathcal{ER}} \square$, then $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \vee \mathbf{B})) \vdash_{\mathcal{ER}} \square$, since $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \vee \mathbf{B}) * \neg\mathbf{A} * \neg\mathbf{B}) = \mathrm{CNF}(\Phi * \neg(\mathbf{A} \vee \mathbf{B}))$.

$\nabla_\forall$ By the lifting lemma 2.

$\nabla_\exists$ Let $\mathrm{CNF}(\Phi * \neg\Pi\mathbf{F} * \neg\mathbf{F}w) \vdash^D_{\mathcal{ER}} \square$ and note that $\mathrm{CNF}(\Phi * \neg\Pi\mathbf{F} * \neg\mathbf{F}w) = \mathrm{CNF}(\Phi * \neg\mathbf{F}w' * \neg\mathbf{F}w)$. Now let w'' be any new constant symbol which does not occur in Φ or $\mathbf{F}$. Since also w and w' do not occur in Φ or $\mathbf{F}$ it is easy to verify that their is a derivation $\mathrm{CNF}(\Phi * \neg\mathbf{F}w'') \vdash^{D'}_{\mathcal{ER}} \square$, where each occurrence of $\neg\mathbf{F}w'$ or $\neg\mathbf{F}w$ is replaced by $\neg\mathbf{F}w''$. Hence $\mathrm{CNF}(\Phi * \neg\Pi\mathbf{F}) \vdash_{\mathcal{ER}} \square$.

∇_b We show that if $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq^o \mathbf{B}) * \neg\mathbf{A} * \mathbf{B}) \vdash_{\mathcal{ER}} \square$ and $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq^o \mathbf{B}) * \mathbf{A} * \neg\mathbf{B}) \vdash_{\mathcal{ER}} \square$, then $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq \mathbf{B}) \vdash_{\mathcal{ER}} \square$. Note that $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq \mathbf{B})) = \mathrm{CNF}(\Phi * \neg\Pi(\lambda P.\neg P\mathbf{A} \vee P\mathbf{B})) = \mathrm{CNF}(\Phi) * [r\mathbf{A}]^T * [r\mathbf{B}]^F$, with Skolem constant $r_{o\to o}$. Now consider the following derivation

$$\dfrac{\dfrac{\dfrac{[r\mathbf{A}]^T \quad [r\mathbf{B}]^F}{[r\mathbf{A} \doteq r\mathbf{B}]^F}\,Res}{[\mathbf{A} \doteq \mathbf{B}]^F}\,Dec}{\mathrm{CNF}(\neg(\mathbf{A} \equiv \mathbf{B}))}\,Equiv$$

Hence $\mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq \mathbf{B})) \vdash_{\mathcal{ER}} \mathrm{CNF}(\Phi * \neg(\mathbf{A} \doteq \mathbf{B})) \cup \mathrm{CNF}(\neg(\mathbf{A} \equiv \mathbf{B}))$ and we get the conclusion as a simple consequence of lemma 1(4).

∇_q We show that if $\mathrm{CNF}(\Phi * \neg(\mathbf{F} \doteq^{\alpha\to\beta} \mathbf{G}) * \neg(\mathbf{F}w \doteq^\beta \mathbf{G}w)) \vdash_{\mathcal{ER}} \square$, then $\mathrm{CNF}(\Phi * \neg(\mathbf{F} \doteq \mathbf{G})) \vdash_{\mathcal{ER}} \square$. Note that $\mathrm{CNF}(\Phi * \neg(\mathbf{F} \doteq \mathbf{G}) * \neg(\mathbf{F}w \doteq \mathbf{G}w)) = \mathrm{CNF}(\Phi * \neg\Pi(\lambda Q.\neg Q\mathbf{F} \vee Q\mathbf{G}) * \neg\Pi(\lambda P.\neg P(\mathbf{F}w) \vee P(\mathbf{G}w))) = \mathrm{CNF}(\Phi) * [q\mathbf{F}]^T * [q\mathbf{G}]^F * [p(\mathbf{F}w)]^T * [p(\mathbf{G}w)]^F$ and that $\mathrm{CNF}(\Phi * \neg(\mathbf{F} \doteq \mathbf{G})) = \mathrm{CNF}(\Phi) * [r\mathbf{F}]^T * [r\mathbf{G}]^F$, where $p_{\beta\to o}, q_{(\alpha\to\beta)\to o}$ and $r_{(\alpha\to\beta)\to o}$ are new Skolem constants. Now consider the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{[r\mathbf{F}]^T \quad [r\mathbf{G}]^F}{[r\mathbf{F} \doteq r\mathbf{G}]^F}\,Res}{[\mathbf{F} \doteq \mathbf{G}]^F}\,Dec}{[\mathbf{F}s \doteq \mathbf{G}s]^F}\,Func}{\begin{array}{c}[t(\mathbf{F}s)]^T\\ [t(\mathbf{G}s)]^F\end{array}}\,Leib$$

Here again s_α and $t_{\beta\to o}$ are new Skolem constants. Hence $\mathrm{CNF}(\Phi) * [r\mathbf{F}]^T * [r\mathbf{G}]^F \vdash_{\mathcal{ER}} \mathrm{CNF}(\Phi) * [r\mathbf{F}]^T * [r\mathbf{G}]^F * [t(\mathbf{F}s)]^T * [t(\mathbf{G}s)^F$.
Now the conclusion follows from the assumption since s, t and r are only renamings of the Skolem symbols w, p and q and all do not occur in Φ.

To see that Γ_Σ is saturated let $\mathbf{A} \in wff_o(\Sigma)$ and $\Phi \subseteq wff_o^{cl}(\Sigma)$ with $\Phi \nvdash_{\mathcal{ER}} \square$. We have to show that $\Phi * \mathbf{A} \nvdash_{\mathcal{ER}} \square$ or $\Phi * \neg\mathbf{A} \nvdash_{\mathcal{ER}} \square$. For that suppose $\Phi \nvdash_{\mathcal{ER}} \square$, but $\Phi * \mathbf{A} \vdash_{\mathcal{ER}} \square$ and $\Phi * \neg\mathbf{A} \vdash_{\mathcal{ER}} \square$. By lemma 1(3) we get that $\Phi * \mathbf{A} \vee \neg\mathbf{A} \vdash_{\mathcal{ER}} \square$, and hence, since $\mathbf{A} \vee \neg\mathbf{A}$ is a tautology, it must be the case that $\Phi \vdash_{\mathcal{ER}} \square$, which contradicts our assumption.

5 Conclusion

We have presented an extensional higher-order resolution calculus that is complete relative to Henkin model semantics. The treatment of the extensionality principles – necessary for the completeness result – by specialized (goal-directed) inference rules practical applicability, as an implentation of the calculus in the LEO-System [BK98] shows.

Acknowledgments The work reported here was funded by the Deutsche Forschungsgemeinschaft under grant HOTEL. The authors are grateful to Peter Andrews and Frank Pfenning for stimulating discussions.

References

ABI+96. Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

And71. Peter B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36(3):414–432, 1971.

And72. Peter B. Andrews. General models descriptions and choice in type theory. *Journal of Symbolic Logic*, 37(2):385–394, 1972.

And86. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.

And89. Peter B. Andrews. On Connections and Higher Order Logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

Bar84. H. P. Barendregt. *The Lambda Calculus*. North Holland, 1984.

Ben97. Christoph Benzmüller. A calculus and a system architecture for extensional higher-order resolution. Research Report 97-198, Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh,USA, June 1997.

BK97. Christoph Benzmüuller and Michael Kohlhase. Model existence for higher-order logic. SEKI-Report SR-97-09, Universität des Saarlandes, 1997.

BK98. Christoph Benzmüller and Michael Kohlhase. LEO, a higher-order theorem prover. to appear at CADE-15, 1998.

dB72. Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with an application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

Göd31. Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte der Mathematischen Physik*, 38:173–198, 1931.

Hen50. Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.

Hue73. Gérard P. Huet. A mechanization of type theory. In Donald E. Walker and Lewis Norton, editors, *Proc. IJCAI'73*, pages 139–146, 1973.

JP76. D. C. Jensen and T. Pietrzykowski. Mechanizing ω-order type theory through unification. *Theoretical Computer Science*, 3:123–171, 1976.

Koh94. Michael Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Universität des Saarlandes, 1994.

Koh95. Michael Kohlhase. Higher-Order Tableaux. In P. Baumgartner, et al. eds, *TABLEAUX'95*, volume 918 of *LNAI*, pages 294–309, 1995.

Mil83. Dale Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, 1983.

Smu63. Raymond M. Smullyan. A unifying principle for quantification theory. *Proc. Nat. Acad Sciences*, 49:828–832, 1963.

Can a Higher-Order and a First-Order Theorem Prover Cooperate?*

Christoph Benzmüller[1], Volker Sorge[2], Mateja Jamnik[3], and Manfred Kerber[2]

[1] Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany (www.ags.uni-sb.de/~chris)
[2] School of Computer Science, The University of Birmingham
Birmingham B15 2TT, England, UK (www.cs.bham.ac.uk/~{vxs|mmk})
[3] University of Cambridge Computer Laboratory
Cambridge CB3 0FD, England, UK (www.cl.cam.ac.uk/~mj201)

Abstract. State-of-the-art first-order automated theorem proving systems have reached considerable strength over recent years. However, in many areas of mathematics they are still a long way from reliably proving theorems that would be considered relatively simple by humans. For example, when reasoning about sets, relations, or functions, first-order systems still exhibit serious weaknesses. While it has been shown in the past that higher-order reasoning systems can solve problems of this kind automatically, the complexity inherent in their calculi and their inefficiency in dealing with large numbers of clauses prevent these systems from solving a whole range of problems.
We present a solution to this challenge by combining a higher-order and a first-order automated theorem prover, both based on the resolution principle, in a flexible and distributed environment. By this we can exploit concise problem formulations without forgoing efficient reasoning on first-order subproblems. We demonstrate the effectiveness of our approach on a set of problems still considered non-trivial for many first-order theorem provers.

1 Introduction

When dealing with problems containing higher-order concepts, such as sets, functions, or relations, today's state-of-the-art first-order automated theorem provers (ATPs) still exhibit weaknesses on problems considered relatively simple by humans (cf. [14]). One reason is that the problem formulations use an encoding in a first-order set theory, which makes it particularly challenging when trying to prove theorems from first principles, that is, basic axioms. Therefore, to aid ATPs in finding proofs, problems are often enriched by hand-picked additional lemmata, or axioms of the selected set theory are dropped leaving the theory incomplete. This has recently motivated extensions of state-of-the-art first-order

* This work was supported by EPSRC grant GR/M22031 and DFG-SFB 378 (first author), EU Marie-Curie-Fellowship HPMF-CT-2002-01701 (second author), and EPSRC Advanced Research Fellowship GR/R76783 (third author).

F. Baader and A. Voronkov (Eds.): LPAR 2004, LNAI 3452, pp. 415–431, 2005.

calculi and systems, as for example presented in [14] for the SATURATE system. The extended SATURATE system can solve some problems from the SET domain in the TPTP [24] which VAMPIRE [21] and E-SETHEO's [23] cannot solve.

While it has already been shown in [6,2] that many problems of this nature can be easily proved from first principles using a concise higher-order representation and the higher-order resolution ATP LEO, the combinatorial explosion inherent in LEO's calculus prevents the prover from solving a whole range of possible problems with one universal strategy. Often higher-order problems require only relatively few but essential steps of higher-order reasoning, while the overwhelming part of the reasoning is first-order or even propositional level. This suggests that LEO's performance could be improved when combining it with a first-order ATP to search efficiently for a possible refutation in the subset of those clauses that are essentially first-order.

The advantages of such a combination — further discussed in Sec. 2 — are not only that many problems can still be efficiently shown from first principles in a general purpose approach, but also that problems can be expressed in a very concise way. For instance, we present 45 problems from the SET domain of the TPTP-v3.0.1, together with their entire formalisation in less than two pages in this paper, which is difficult to achieve within a framework that does not provide λ-abstraction. We use this problem set, which is an extension of the problems considered in [14], in Sec. 4 to show the effectiveness of our approach. While many of the considered problems can be proved by LEO alone with some strategy, the combination of LEO with the first-order ATP BLIKSEM [11] is not only able to show more problems, but also needs only a single strategy to solve them. Several of our problems are considered very challenging by the first-order community and five of them (of which LEO can solve four) have a TPTP rating of 1.00, saying that they cannot be solved by any TPTP prover to date.

Technically, the combination — described in more detail in Sec. 3 — has been realised in the concurrent reasoning system OANTS [22,8] which enables the cooperation of hybrid reasoning systems to construct a common proof object. In our past experiments, OANTS has been successfully employed to check the validity of set equations using higher-order and first-order ATPs, model generation, and computer algebra [5]. While this already enabled a cooperation between LEO and a first-order ATP, the proposed solution could not be classified as a general purpose approach. A major shortcoming was that all communication of partial results had to be conducted via the common proof object, which was very inefficient for hard examples. Thus, the solved examples from set theory were considered too trivial, albeit they were often similar to those still considered challenging in the TPTP in the first-order context. In this paper we now present a novel approach to the cooperation between LEO and BLIKSEM inside OANTS by decentralising communication. This leads not only to a higher overall efficiency — Sec. 4 details our results — but also to a general purpose approach based on a single strategy in LEO.

2 Why Linking Higher-Order and First-Order?

Existing higher-order ATPs generally exhibit deficits in efficiently reasoning with first-order problems for several reasons. Unlike in the case of first-order provers, for which sophisticated calculi and strategies, as well as advanced implementation techniques, such as term indexing [19], have been developed, fully mechanisable higher-order calculi are still at a comparably early stage of development. Some problems are much harder in higher-order, for instance, unification is undecidable, strong constraining term- and literal-orderings are not available, extensionality reasoning and set variable instantiation has to be addressed. Nevertheless, for some mathematical problem domains, such as naive set theory, for instance, automated higher-order reasoning performs very well.

We motivate the need for linking higher-order and first-order ATPs with some examples from Table 1. It contains a range of challenging problems taken from the TPTP, against which we will evaluate our system in Sec. 4. The problems are given by the identifiers used in the SET domain of the TPTP, and are formalised in a variant of Church's simply typed λ-calculus with prefix polymorphism. In classical type theory terms and all their sub-terms are typed. Polymorphism allows the introduction of type variables such that statements can be made for all types. For instance, in problem SET014+4 the universally quantified variable $X_{o\alpha}$ denotes a mapping from objects of type α to objects of type o. We use Church's notation $o\alpha$, which stands for the functional type $\alpha \to o$. The reader is referred to [1] for a more detailed introduction. In the remainder, o will denote the type of truth values, and small Greek letters will denote arbitrary types. Thus, $X_{o\alpha}$ (resp. its η-longform $\lambda y_\alpha . Xy$) is actually a characteristic function denoting the set of elements of type α, for which the predicate associated with X holds. As further notational convention, we use capital letter variables to denote sets, functions, or relations, while lower case letters denote individuals. Types are usually only given in the first occurrence of a variable and omitted if inferable from the context.

The problems in Table 1 employ defined concepts that are specified in a knowledge base of hierarchical theories that Leo has access to. All concepts necessary for defining our problems in Table 1 are given in Table 2. Concepts are defined in terms of λ-expressions and they may contain other, already specified concepts. For presentation purposes, we use customary mathematical symbols $\cup, \cap$, etc., for some concepts like *union, intersection*, etc., and we also use infix notation. For instance, the definition of union on sets can be easily read in its more common mathematical representation $A \cup B := \{x | x \in A \vee x \in B\}$. Before proving a problem, Leo always expands — recursively, if necessary — all occurring concepts. This straightforward expansion to first principles is realised by an automated preprocess in our current approach.

SET171+3 We first discuss example SET171+3 to contrast our formalisation to a standard first-order one. After recursively expanding the input problem, that is, completely reducing it to first principles, Leo turns it into a negated unit clause. Since this initial clause is not in normal form, Leo first normalises it with explicit

Table 1. Problems from TPTP for the evaluation of Oants

SET	Problem Formalisation
014+4	$\forall X_{o\alpha}, Y_{o\alpha}, A_{o\alpha}.[[X \subseteq A \wedge Y \subseteq A] \Rightarrow (X \cup Y) \subseteq A]$
017+1	$\forall x_\alpha, y_\alpha, z_\alpha.[UnOrderedPair(x,y) = UnOrderedPair(x,z) \Rightarrow y = z]$
066+1	$\forall x_\alpha, y_\alpha.[UnOrderedPair(x,y) = UnOrderedPair(y,x)$
067+1	$\forall x_\alpha, y_\alpha.[UnOrderedPair(x,x) \subseteq UnOrderedPair(x,y)]$
076+1	$\forall x_\alpha, y_\alpha.\forall Z_{o\alpha}.x \in Z \wedge y \in Z \Rightarrow UnOrderedPair(x,y) \subseteq Z$
086+1	$\forall x_\alpha.\exists y_\alpha.[y \in Singleton(x)]$
096+1	$\forall X_{o\alpha}, y_\alpha.[X \subseteq Singleton(y) \Rightarrow [X = \emptyset \vee X = Singleton(y)]]$
143+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \cap Y) \cap Z = X \cap (Y \cap Z)]$
171+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)]$
580+3	$\forall X_{o\alpha}, Y_{o\alpha}, u_\alpha.[u \in ExclUnion(X,Y) \Leftrightarrow [u \in X \Leftrightarrow u \notin Y]]$
601+3	$\forall_. X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}[(X \cap Y) \cup ((Y \cap Z) \cup (Z \cap X)) = (X \cup Y) \cap ((Y \cup Z) \cap (Z \cup X))]$
606+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \backslash (X \cap Y) = X \backslash Y]$
607+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \cup (Y \backslash X) = X \cup Y]$
609+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \backslash (Y \backslash Z) = (X \backslash Y) \cup (X \cap Z)]$
611+3	$\forall X_{o\alpha}, Y_{o\alpha}.[X \cap Y = \emptyset \Leftrightarrow X \backslash Y = X]$
612+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[X \backslash (Y \cup Z) = (X \backslash Y) \cap (X \backslash Z)]$
614+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \backslash Y) \backslash Z = X \backslash (Y \cup Z)]$
615+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[(X \cup Y) \backslash Z = (X \backslash Z) \cup (Y \backslash Z)]$
623+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[ExclUnion(ExclUnion(X,Y),Z) = ExclUnion(X, ExclUnion(Y,Z))]$
624+3	$\forall X_{o\alpha}, Y_{o\alpha}, Z_{o\alpha}.[Meets(X,(Y \cup Z)) \Leftrightarrow [Meets(X,Y) \vee Meets(X,Z)]]$
630+3	$\forall X_{o\alpha}, Y_{o\alpha}.[Misses(X \cap Y, ExclUnion(X,Y))]$
640+3	$\forall R_{o\beta\alpha}, Q_{o\beta\alpha}.[Subrel(R,Q) \Rightarrow Subrel(R, (\lambda u_\alpha.\top) \times (\lambda v_\beta.\top))]$
646+3	$\forall x_\alpha, y_\beta.[Subrel(Pair(x,y), (\lambda u_\alpha.\top) \times (\lambda v_\beta.\top))]$
647+3	$\forall R_{o\beta\alpha}, X_{o\alpha}.[(RDom(R) \subseteq X) \Rightarrow Subrel(R, X \times RCodom(R))]$
648+3	$\forall R_{o\beta\alpha}, Y_{o\beta}.[(RCodom(R) \subseteq Y) \Rightarrow Subrel(R, RDom(R) \times Y)]$
649+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[RDom(R) \subseteq X \wedge RCodom(R) \subseteq Y] \Rightarrow Subrel(R, X \times Y)]$
651+3	$\forall R_{o\beta\alpha}.[RDom(R) \subseteq A_{o\alpha} \Rightarrow Subrel(R, A \times (\lambda u_\beta.\top))]$
657+3	$\forall R_{o\beta\alpha}.[Field(R) \subseteq ((\lambda u_\alpha.\top) \cup (\lambda v_\beta.\top))]$
669+3	$\forall R_{o\alpha\alpha}.[Subrel(Id(\lambda u_\alpha.\top), R) \Rightarrow [(\lambda u_\alpha.\top) \subseteq RDom(R) \wedge (\lambda u_\alpha.\top) = RCodom(R)]]$
670+3	$\forall Z_{o\alpha}, R_{o\beta\alpha}, X_{o\alpha} Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow IsRelOn(RestrictRDom(R,Z), Z, Y)]$
671+3	$\forall Z_{o\alpha}, R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[IsRelOn(R,X,Y) \wedge X \subseteq Z] \Rightarrow RestrictRDom(R,Z) = R]$
672+3	$\forall Z_{o\beta}, R_{o\beta\alpha}, X_{o\alpha} Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow IsRelOn(RestrictRCodom(R,Z), X, Z)]$
673+3	$\forall Z_{o\beta}, R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[[IsRelOn(R,X,Y) \wedge Y \subseteq Z] \Rightarrow RestrictRCodom(R,Z) = R]$
680+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow [\forall u_\alpha.u \in X \Rightarrow [u \in RDom(R) \Leftrightarrow \exists v_\beta.v \in Y \wedge R(u,v)]]]$
683+3	$\forall R_{o\beta\alpha}, X_{o\alpha}, Y_{o\beta}.[IsRelOn(R,X,Y) \Rightarrow [\forall v_\beta.v \in Y \Rightarrow [v \in RCodom(R) \Rightarrow \exists u_\alpha.u \in X \wedge u \in RDom(R)]]]$
684+3	$\forall P_{o\beta\alpha}, R_{o\gamma\beta}, x_\alpha, z_\gamma.[RelComp(P,R)xz \Leftrightarrow \exists y_\beta.Pxy \wedge Ryz]$
686+3	$\forall Z_{o\alpha}, R_{o\gamma\beta}, x_\alpha.[x \in InverseImageR(R,Z) \Leftrightarrow \exists y_\alpha.Rxy \wedge x \in Z]$
716+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}.[[Inj(F) \wedge Inj(G)] \Rightarrow Inj(G \circ F)]$
724+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, H_{\gamma\beta}.[[F \circ G = F \circ H \wedge Surj(F)] \Rightarrow G = H]$
741+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, H_{\alpha\gamma}.[[Inj((F \circ G) \circ H) \wedge Surj((G \circ H) \circ F) \wedge Surj((H \circ F) \circ G)] \Rightarrow Bij(H)]$
747+4	$\forall F_{\beta\alpha}, G_{\gamma\beta}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}, \lhd^3_{o\gamma\gamma}.[[IncreasingF(F,\lhd^1,\lhd^2) \wedge DecreasingF(G,\lhd^2,\lhd^3)] \Rightarrow DecreasingF(F \circ G, \lhd^1, \lhd^3)]$
752+4	$\forall X_{o\alpha}, Y_{o\alpha}, F_{\beta\alpha}.[ImageF(F, X \cup Y) = ImageF(F,X) \cup ImageF(F,Y)]$
753+4	$\forall X_{o\alpha}, Y_{o\alpha}, F_{\beta\alpha}.[ImageF(F, X \cap Y) \subseteq ImageF(F,X) \cap ImageF(F,Y)]$
764+4	$\forall F_{\beta\alpha}.[InverseImageF(F,\emptyset) = \emptyset]$
770+4	$\forall R_{o\beta\alpha}, Q_{o\beta\alpha}.[[EquivRel(R) \wedge EquivRel(Q)] \Rightarrow [EquivClasses(R) = EquivClasses(Q) \vee Disjoint(EquivClasses(R), EquivClasses(Q))]]$

clause normalisation rules to reach some proper initial clauses. In our concrete case, this normalisation process leads to the following unit clause consisting of a (syntactically not solvable) unification constraint (here $B_{o\alpha}, C_{o\alpha}, D_{o\alpha}$ are Skolem constants and Bx is obtained from expansion of $x \in B$):

$$[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) =^? (\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]$$

Note that negated primitive equations are generally automatically converted by Leo into unification constraints. This is why $[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) =^?$

Table 2. Defined concepts occurring in problems from Table 1

Defined Notions in Theory Typed Set		
$\_ \in \_$	$:=$	$\lambda x_\alpha, A_{o\alpha}.[Ax]$
$\emptyset$	$:=$	$[\lambda x_\alpha.\bot]$
$\_ \subseteq \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\forall x_\alpha.x \in A \Rightarrow x \in B]$
$\_ \cup \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \vee x \in B]$
$\_ \cap \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \wedge x \in B]$
$\overline{\_}$	$:=$	$\lambda A_{o\alpha}.[\lambda x_\alpha.x \notin A]$
$\_ \backslash \_$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\lambda x_\alpha.x \in A \wedge x \notin B]$
$ExclUnion(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[(A \backslash B) \cup (B \backslash A)]$
$Disjoint(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[A \cap B = \emptyset]$
$Meets(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\exists x_\alpha.x \in A \wedge x \in B]$
$Misses(\_,\_)$	$:=$	$\lambda A_{o\alpha}, B_{o\alpha}.[\neg\exists x_\alpha.x \in A \wedge x \in B]$
Defined Notions in Theory Relation		
$UnOrderedPair(\_,\_)$	$:=$	$\lambda x_\alpha, y_\alpha.[\lambda u_\alpha.u = x \vee u = y]$
$Singleton(\_)$	$:=$	$\lambda x_\alpha.[\lambda u_\alpha.u = x]$
$Pair(\_,\_)$	$:=$	$\lambda x_\alpha, y_\beta.[\lambda u_\alpha, v_\beta.u = x \wedge v = y]$
$\_ \times \_$	$:=$	$\lambda A_{o\alpha}, B_{o\beta}.[\lambda u_\alpha, v_\beta.u \in A \wedge v \in B]$
$RDom(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\lambda x_\alpha.\exists y_\beta.Rxy]$
$RCodom(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\lambda y_\beta.\exists x_\alpha.Rxy]$
$Subrel(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, Q_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.Rxy \Rightarrow Qxy]$
$Id(\_)$	$:=$	$\lambda A_{o\alpha}.[\lambda x_\alpha, y_\alpha.x \in A \wedge x = y]$
$Field(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[RDom(B) \cup RCodom(R)]$
$IsRelOn(\_,\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, A_{o\alpha}.\lambda B_{o\beta}.[\forall x_\alpha, y_\beta.Rxy \Rightarrow (x \in A \wedge x \in B)]$
$RestrictRCodom(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, A_{o\alpha}.[\lambda x_\alpha, y_\beta.x \in A \wedge Rxy]$
$RelComp(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, Q_{o\gamma\beta}.[\lambda x_\alpha, z_\gamma.\exists y_\beta.Rxy \wedge Ryz]$
$InverseImageR(\_,\_)$	$:=$	$\lambda R_{o\beta\alpha}, B_{o\beta}.[\lambda x_\alpha.\exists y_\beta.y \in B \wedge Rxy]$
$Reflexive(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.Rxx]$
$Symmetric(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.Rxy \Rightarrow Ryx]$
$Transitive(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[\forall x_\alpha.\forall y_\alpha.\forall z_\alpha.Rxy \wedge Ryz \Rightarrow Rxz]$
$EquivRel(\_)$	$:=$	$\lambda R_{o\beta\alpha}.[Reflexive(R) \wedge Symmetric(R) \wedge Transitive(R)]$
$EquivClasses(\_)$	$:=$	$\lambda R_{o\alpha\alpha}.[\lambda A_{o\alpha}.\exists u_\alpha.u \in A \wedge \forall v_\alpha.v \in A \Leftrightarrow Ruv]$
Defined Notions in Theory Function		
$Inj(\_)$	$:=$	$\lambda F_{\beta\alpha}.[\forall x_\alpha, y_\beta.F(x) = F(y) \Rightarrow x = y]$
$Surj(\_)$	$:=$	$\lambda F_{\beta\alpha}.[\forall y_\beta.\exists x_\alpha.y = F(x)]$
$Bij(\_)$	$:=$	$\lambda F_{\beta\alpha}.Surj(F) \wedge Inj(F)$
$ImageF(\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, A_{o\alpha}.[\lambda y_\beta.\exists x_\alpha.x \in A \wedge y = F(x)]$
$InverseImageF(\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, B_{o\beta}.[\lambda x_\alpha.\exists y_\beta.y \in B \wedge y = F(x)]$
$\_ \circ \_$	$:=$	$\lambda F_{\beta\alpha}, G_{\gamma\beta}.[\lambda x_\alpha.G(F(x))]$
$IncreasingF(\_,\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}.[\forall x_\alpha, y_\alpha.x \lhd^1 y \Rightarrow F(x) \lhd^2 F(y)]$
$DecreasingF(\_,\_,\_)$	$:=$	$\lambda F_{\beta\alpha}, \lhd^1_{o\alpha\alpha}, \lhd^2_{o\beta\beta}.[\forall x_\alpha, y_\alpha.x \lhd^1 y \Rightarrow F(y) \lhd^2 F(x)]$

$(\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]$ is generated, and not $[(\lambda x_\alpha.Bx \vee (Cx \wedge Dx)) = (\lambda x_\alpha.(Bx \vee Cx) \wedge (Bx \vee Dx))]^F$. Observe, that we write $[.]^T$ and $[.]^F$ for positive and negative literals, respectively. Leo then applies its goal directed functional and Boolean extensionality rules which replace this unification constraint by the negative literal (where x is a Skolem constant):

$$[(Bx \vee (Cx \wedge Dx)) \Leftrightarrow ((Bx \vee Cx) \wedge (Bx \vee Dx))]^F$$

This unit clause is again not normal; normalisation, factorisation and subsumption yield the following set of clauses:

$$[Bx]^F \qquad [Bx]^T \vee [Cx]^T \qquad [Bx]^T \vee [Dx]^T \qquad [Cx]^F \vee [Dx]^F$$

This set is essentially of propositional logic character and trivially refutable. Leo needs 0.56 seconds for solving the problem and generates a total of 36 clauses.

Let us consider now this same example SET171+3 in its first-order formulation from the TPTP (see Table 3). We can observe that the assumptions provide

Table 3. TPTP problem SET171+3 — distributivity of $\cup$ over $\cap$

Assumptions:	$\forall B, C, x_{\bullet}[x \in (B \cup C) \Leftrightarrow x \in B \vee x \in C]$	(1)
	$\forall B, C, x_{\bullet}[x \in (B \cap C) \Leftrightarrow x \in B \wedge x \in C]$	(2)
	$\forall B, C_{\bullet}[B = C \Leftrightarrow B \subseteq C \wedge C \subseteq B]$	(3)
	$\forall B, C_{\bullet}[B \cup C = C \cup B]$	(4)
	$\forall B, C_{\bullet}[B \cap C = C \cap B]$	(5)
	$\forall B, C_{\bullet}[B \subseteq C \Leftrightarrow \forall x_{\bullet} x \in B \Rightarrow x \in C]$	(6)
	$\forall B, C_{\bullet}[B = C \Leftrightarrow \forall x_{\bullet} x \in B \Leftrightarrow x \in C]$	(7)
Proof Goal:	$\forall B, C, D_{\bullet}[B \cup (C \cap D) = (B \cup C) \cap (B \cup D)]$	(8)

only a partial axiomatisation of naive set theory. On the other hand, the specification introduces lemmata that are useful for solving the problem. In particular, assumption (7) is trivially derivable from (3) with (6). Obviously, clausal normalisation of this first-order problem description yields a much larger and more difficult set of clauses. Furthermore, definitions of concepts are not directly expanded as in Leo. It is therefore not surprising that most first-order ATPs still fail to prove this problem. In fact, very few TPTP provers were successful in proving SET171+3. Amongst them are Muscadet 2.4. [20], Vampire 7.0, and Saturate. The natural deduction system Muscadet uses special inference rules for sets and needs 0.2 seconds to prove this problem. Vampire needs 108 seconds. The Saturate system [14] (which extends Vampire with Boolean extensionality rules that are a one-to-one correspondence to Leo's rules for Extensional Higher-Order Paramodulation [3]) can solve the problem in 2.9 seconds while generating 159 clauses. The significance of such comparisons is clearly limited since different systems are optimised to a different degree. One noted difference between the experiments with first-order provers listed above, and the experiments with Leo and Leo-Bliksem is that first-order systems often use a case tailored problem representation (e.g., by avoiding some base axioms of the addressed theory), while Leo and Leo-Bliksem have a harder task of dealing with a general (not specifically tailored) representation.

For the experiments with Leo and the cooperation of Leo with the first-order theorem prover Bliksem, λ-abstraction as well as the extensionality treatment inherent in Leo's calculus [4] is used. This enables a theoretically[4] Henkin-complete proof system for set theory. In the above example SET171+3, Leo generally uses the application of functional extensionality to push extensional unification constraints down to base type level, and then eventually applies Boolean extensionality to generate clauses from them. These are typically much simpler and often even *propositional-like* or *first-order-like* (FO-like, for short), that is, they do not contain any 'real' higher-order subterms (such as a λ-abstraction or

[4] For pragmatic reasons, such as efficiency, most of Leo's tactics are incomplete. Leo's philosophy is to rely on a theoretically complete calculus, but to practically provide a set of complimentary strategies so that these cover a broad range of theorems.

embedded equations), and are therefore suitable for treatment by a first-order ATP or even a propositional logic decision procedure.

SET624+3 Sometimes, extensionality treatment is not required and the originally higher-order problem is immediately reduced to only FO-like clauses. For example, after expanding the definitions, problem SET624+3 yields the following clause (where $B_{o\alpha}, C_{o\alpha}, D_{o\alpha}$ are again Skolem constants):

$$[(\exists x_\alpha . (Bx \wedge (Cx \vee Dx)) \Leftrightarrow ((\exists x_\alpha . Bx \wedge Cx) \vee (\exists x_\alpha . Bx \wedge Dx))]^F$$

Normalisation results in 26 FO-like clauses, which present a hard problem for Leo: it needs approx. 35 seconds (see Sec. 4) to find a refutation, whereas first-order ATPs only need a fraction of a second.

SET646+3 Sometimes, problems are immediately refuted after the initial clause normalisation. For example, after definition expansion in problem SET646+3 we get the following clause (where $B_{o\alpha}, C_{o\alpha}, x_\alpha$ are again Skolem constants):

$$[Ax \Rightarrow (\forall y_\beta . By \Rightarrow (\forall u_\alpha . \forall v_\beta . (u = x \wedge v = y) \Rightarrow ((\neg\bot) \wedge (\neg\bot))))]^F$$

Normalisation in Leo immediately generates a basic refutation (i.e., a clause $[\bot]^T \vee [\bot]^T$) without even starting proof search.

SET611+3 The examples discussed so far all essentially apply extensionality treatment and normalisation to the input problem in order to immediately generate a set of inconsistent FO-like clauses. Problem SET611+3 is more complicated as it requires several reasoning steps in Leo before the initially consistent set of available FO-like clauses grows into an inconsistent one. After definition expansion, Leo is first given the input clause:

$$[\forall A_{o\alpha}, B_{o\alpha} . (\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)) \Leftrightarrow (\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^F$$

which it normalises into:

$$[(\lambda x_\alpha . (Ax \wedge Bx)) =^? (\lambda x_\alpha . \bot)] \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) =^? (\lambda x_\alpha . Ax)] \quad (9)$$

$$[(\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)]^T \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^T \quad (10)$$

As mentioned before, the unification constraint (9) corresponds to:

$$[(\lambda x_\alpha . (Ax \wedge Bx)) = (\lambda x_\alpha . \bot)]^F \vee [(\lambda x_\alpha . (Ax \wedge \neg Bx)) = (\lambda x_\alpha . Ax)]^F \quad (11)$$

Leo has to apply to each of these clauses and to each of their literals appropriate extensionality rules. Thus, several rounds of Leo's set-of-support-based reasoning procedure are required, so that all necessary extensionality reasoning steps are performed, and sufficiently many FO-like clauses are generated which can be refuted by Bliksem.

In summary, each of the examples discussed in this section exposes a motivation for our higher-order/first-order cooperative approach to theorem proving. In particular, they show that:

- Higher-order formulations allow for a concise problem representation which often allows easier and faster proof search than first-order formulations.
- Higher-order problems can often be reduced to a set of first-order clauses that can be more efficiently handled by a first-order ATP.
- Some problems are trivially refutable after clause normalisation.
- Some problems require in-depth higher-order reasoning before a refutable first-order clause set can be extracted.

3 Higher-Order/First-Order Cooperation via OANTS

The cooperation between higher-oder and first-order reasoners, which we investigate in this paper, is realised in the concurrent hierarchical blackboard architecture OANTS [7]. We first describe in Sec. 3.1 the existing OANTS architecture. In order to overcome some of its problems, in particular efficiency problems, we devised within OANTS a new and improved cooperation method for the higher-order ATP LEO and first-order provers (in particular, BLIKSEM) – we describe this in Sec. 3.2. We address the question of how to generate the necessary clauses in Sec. 3.3, and discuss soundness and completeness of our implementation of the higher-order/first-order cooperation in Sec. 3.4.

3.1 OANTS

OANTS was originally conceived to support interactive theorem proving but was later extended to a fully automated proving system [22,8]. Its basic idea is to compose a *central proof object* by generating, in each proof situation, a ranked list of potentially applicable inference steps. In this process, all inference rules, such as calculus rules or tactics, are uniformly viewed with respect to three sets: premises, conclusions, and additional parameters. The elements of these three sets are called *arguments* of the inference rule and they usually depend on each other. An inference rule is applicable if at least some of its arguments can be instantiated with respect to the given proof context. The task of the OANTS architecture is now to determine the applicability of inference rules by computing instantiations for their arguments.

The architecture consists of two layers. On the lower layer, possible instantiations of the arguments of individual inference rules are computed. In particular, each inference rule is associated with its own blackboard and concurrent processes, one for each argument of the inference rule. The role of every process is to compute possible instantiations for its designated argument of the inference rule, and to record these on the blackboard. The computations are carried out with respect to the given proof context and by exploiting information already present on the blackboard, that is, argument instantiations computed by other processes. On the upper layer, the information from the lower layer is used for computing and heuristically ranking the inference rules that are applicable in the current proof state. The most promising rule is then applied to the central

proof object and the data on the blackboards is cleared for the next round of computations.

OANTS employs resource reasoning to guide search.[5] This enables the controlled integration (e.g., by specifying time-outs) of full-fledged external reasoning systems such as automated theorem provers, computer algebra systems, or model generators into the architecture. The use of the external systems is modelled by inference rules, usually one for each system. Their corresponding computations are encapsulated in one of the independent processes in the architecture. For example, an inference rule modelling the application of an ATP has its conclusion argument set to be an open goal. A process can then place an open goal on the blackboard, where it is picked up by a process that applies the prover to it. Any computed proof or partial-proof from the external system is again written to the blackboard from where it is subsequently inserted into the proof object when the inference rule is applied. While this setup enables proof construction by a collaborative effort of diverse reasoning systems, the cooperation can only be achieved via the central proof object. This means that all partial results have to be translated back and forth between the syntaxes of the integrated systems and the language of the proof object. Since there are many types of integrated systems, the language of the proof object — a higher-order language even richer than LEO's, together with a natural deduction calculus — is expressive but also cumbersome. This leads not only to a large communication overhead, but also means that complex proof objects have to be created (large clause sets need to be transformed into large single formulae to represent them in the proof object; the support for this in OANTS to date is inefficient), even if the reasoning of all systems involved is clause-based. Consequently, the cooperation between external systems is typically rather inefficient [5].

3.2 Cooperation via a Single Inference Rule

In order to overcome the problem of the communication bottleneck described above, we devised a new method for the cooperation between a higher-order and a first-order theorem prover within OANTS. Rather than modelling each theorem prover as a separate inference rule (and hence needing to translate the communication via the language of the central proof object), we model the cooperation between a higher-order (concretely, LEO) and a first-order theorem prover (in our case study BLIKSEM) in OANTS as a single inference rule. The cooperation between these two theorem provers is carried out directly and not via the central proof object. This avoids translating clause sets into single formulae and back. While in our previous approach the cooperation between LEO and an FO-ATP was modelled at the upper layer of the OANTS architecture, our new approach presented in this paper models their cooperation by exploiting the lower layer of the OANTS blackboard architecture. This is not an ad hoc solution,

[5] OANTS provides facilities to define and modify the processes at run-time. But notice that we do not use these advanced features in the case study presented in this paper.

but rather, it demonstrates OANTS's flexibility in modelling the integration of cooperative reasoning systems.

Concretely, the single inference rule modelling the cooperation between LEO and a first-order theorem prover needs four arguments to be applicable: (1) an open proof goal, (2) a partial LEO proof, (3) a set of FO-like clauses in the partial proof, (4) a first-order refutation proof for the set of FO-like clauses. Each of these arguments is computed, that is, its instantiation is found, by an independent process. The first process finds open goals in the central proof object and posts them on the blackboard associated with the new rule. The second process starts an instance of the LEO theorem prover for each new open goal on the blackboard. Each LEO instance maintains its own set of FO-like clauses. The third process monitors these clauses, and as soon as it detects a change in this set, that is, if new FO-like clauses are added by LEO, it writes the entire set of clauses to the blackboard. Once FO-like clauses are posted, the fourth process first translates each of the clauses directly into a corresponding one in the format of the first-order theorem prover, and then starts the first-order theorem prover on them. Note that writing FO-like clauses on the blackboard is by far not as time consuming as generating higher-order proof objects. As soon as either LEO or the first-order prover finds a refutation, the second process reports LEO's proof or partial proof to the blackboard, that is, it instantiates argument (2). Once all four arguments of our inference rule are instantiated, the rule can be applied and the open proof goal can be closed in the central proof object. That is, the open goal can be proved by the cooperation between LEO and a first-order theorem prover. When computing applicability of the inference rule, the second and the fourth process concurrently spawn processes running LEO or a first-order prover on a different set of FO-like clauses. Thus, when actually applying the inference rule, all these instances of provers working on the same open subgoal are stopped.

The cooperation can be carried out between any first-order theorem prover and LEO instantiated with any strategy, thus resulting in different instantiations of the inference rule discussed above. While several first-order provers are integrated in OANTS and could be used, BLIKSEM was sufficient for the case study reported in this paper (see Sec. 4). In most cases, more than one BLIKSEM process was necessary. But as the problems were always concerned with only one subgoal, only one LEO process had to be started.

Our approach to the cooperation between a higher-order and a first-order theorem prover has many advantages. The main one is that the communication is restricted to the transmission of clauses, and thus it avoids intermediate translation into the language of the central proof object. This significantly reduces the communication overhead and makes effective proving of more involved theorems feasible. A disadvantage of this approach is that we cannot easily translate and integrate the two proof objects produced by LEO and BLIKSEM into the central proof object maintained by OANTS, as is possible when applying only one prover per open subgoal. Providing such translation remains future work. The repercussions will be discussed in more detail in Sec. 3.4.

3.3 Extracting FO-Like Clauses from LEO

Crucial to a successful cooperation between LEO and a first-order ATP is obviously the generation of FO-like clauses. LEO always maintains a heap of FO-like clauses. In the current LEO system this heap remains rather small since LEO's standard calculus intrinsically avoids primitive equality and instead provides a rule that replaces occurrences of primitive equality with their corresponding Leibniz definitions which are higher-order. The Leibniz principle defines equality as follows $=_{o\alpha\alpha} := \lambda x_\alpha . \lambda y_\alpha . [\forall P_{o\alpha} . Px \Rightarrow Py]$. LEO also provides a rule which replaces syntactically non-unifiable unification constraints between terms of non-Boolean base type by their respective representations that use Leibniz equality. While the clauses resulting from these rules are still refutable in LEO, they are not refutable by BLIKSEM without adding set theory axioms. We illustrate the effect by the following simple example, where a_ι, b_ι, and $f_{\iota\iota}$ are constants:

$$a = b \Rightarrow f(a) = f(b)$$

Depending on whether we work with primitive equality or Leibniz equality this problem is reduced to the clause sets in either (12) or (13) respectively (in the latter $\mathbf{P}_{o\iota}$ is a new free variable, and $Q_{o\iota}$ is a new Skolem constant):

$$[a = b]^T \qquad [f(a) =^? f(b)] \tag{12}$$

$$[\mathbf{P}a]^F \vee [\mathbf{P}b]^T \qquad [Q(f(a))]^T \qquad [Q(f(b))]^F \tag{13}$$

While the former is obviously refutable in BLIKSEM, the latter is not. LEO, however, still finds a refutation for the latter and generates the crucial substitution $\mathbf{P} \leftarrow \lambda x_\alpha . Q(f(x))$ by higher-order pre-unification.

To circumvent this problem, we adapted the relevant rules in LEO. Instead of immediately constructing Leibniz representation of clauses, an intermediate representation containing primitive equality is generated and dumped on the heap of FO-like clauses. As a consequence, additional useful FO-like clauses are accumulated and the heap can become quite large, in particular, since we do not apply any subsumption to the set of FO-like clauses (this is generally done more efficiently by a first-order ATP anyway). Recent research has shown that Leibniz equality is generally very bad for automating higher-order proof search. Thus, future work in LEO includes providing support for full primitive equality and avoiding Leibniz equations.

3.4 Soundness and Completeness of the Cooperation

Clearly, soundness and completeness properties depend on the corresponding properties of the systems involved, in our case, of LEO and BLIKSEM.

Soundness: The general philosophy of OANTS is to ensure the correctness of proofs by the generation of explicit proof objects, which can be checked independently from the proof generation. In particular, reasoning steps of ATPs have to be translated into OANTS's natural deduction calculus via the TRAMP proof

transformation system [17] to be machine-checkable. Since the cooperative proof result of Leo-Bliksem cannot yet be directly inserted into the centralised proof object, the generation of a machine-checkable proof object is not yet supported. One possible solution is to insert Bliksem proofs into Leo proofs at the right places. Then, the modified Leo proofs can be inserted into the centralised proof object, and hence, explicit proof objects can be generated by Oants. In principle, there is no problem with this, however, it is not yet implemented.

While there are many advantages in guaranteeing correctness of proofs by checking them, it is worth noting that the combination of Leo and Bliksem is sound under the assumption that the two systems are sound. Namely, to prove a theorem it is sufficient to show that a subset of clauses generated in the proof is inconsistent. If Leo generates an inconsistent set of clauses, then it does so correctly by assumption, be it a FO-like set or not. Assuming that the translation from FO-like clauses to truly first-order clauses preserves consistency/inconsistency, then a set of clauses that is given to Bliksem is inconsistent only if Leo generated an inconsistent set of clauses in the first place. By the assumption that Bliksem is sound follows that Bliksem will only generate the empty clause when the original clause set was inconsistent.

Thus, soundness of our cooperative approach critically relies only on the soundness of the selected transformational mapping from FO-like clauses to proper first-order clauses. We use the mapping from Tramp, which has been previously shown to be sound and is based on [16]. Essentially, it injectively maps expressions such as $P(f(a))$ to expressions such as $@^1_{\text{pred}}(P, @^1_{\text{fun}}(f, a))$, where the @ are new first-order operators describing function and predicate application for particular types and arities. The injectivity of the mapping guarantees soundness, since it allows each proof step to be mapped back from first-order to higher-order. Hence, our higher-order/first-order cooperative approach between Leo and Bliksem is sound.

Completeness: Completeness (in the sense of Henkin completeness) can in principle be achieved in higher-order systems, but practically, the strategies used are typically not complete for efficiency reasons. Let us assume that we use a complete strategy in Leo. All that our procedure does is pass FO-like clauses to Bliksem. Hence, no proofs can be lost in this process. That is, completeness follows trivially from the completeness of Leo.

The more interesting question is whether particular cooperation strategies will be complete as well. For instance, in Leo we may want to give higher preference to real higher-order steps which guarantee the generation of first-order clauses.

4 Experiments and Results

We conducted several experiments to evaluate our hybrid reasoning approach. In particular, we concentrated on problems given in Table 1. We investigated several Leo strategies in order to compare Leo's individual performance with the performance of the Leo-Bliksem cooperation. Our example set differs from

the one in [14] in that it contains some additional problems, and it also omits an entry for problem SET108+1. This problem addresses the universal class and can therefore not be formalised in type theory in the same concise way as the other examples, but only in a way very similar to the one given in TPTP.

Table 4 presents the results of our experiments. All timings given in the table are in seconds. The first column contains the TPTP identifier of the problem. The second column relates some of the problems to their counterparts in the Journal of Formalized Mathematics (JFM; see `mizar.org/JFM`) where they originally stem from. This eases the comparison with the results in [6,2], where the problems from the JFM article *Boolean Properties of Sets* were already solved: the problems are named with prefix 'B:'. Prefix 'RS1:' stands for the JFM article *Relations Defined on Sets*. The third column lists the TPTP (v3.0.1 as of 20 January 2005, see `http://www.tptp.org`) difficulty rating of the problem, which indicates how hard the problem is for first-order ATPs (difficulty rating 1.00 indicates that no TPTP prover can solve the problem).

The fourth, fifth and sixth columns list whether SATURATE, MUSCADET (v2.4) and E-SETHEO (csp04), respectively, can (+) or cannot (–) solve a problem. The seventh column lists the timing results for VAMPIRE (v7). The results for SATURATE are taken from [14] (a '?' in Table 4 indicates that the result was not listed in [14] and is thus unavailable). The results for MUSCADET and E-SETHEO are taken from the on-line version of the solutions provided with the TPTP. Since the listed results were obtained from different experiments on different platforms, their run-time comparison would be unfair, and was thus not carried out. The timings for VAMPIRE, on the other hand, are based on private communication with A. Voronkov and they were obtained on a computer with a very similar specification as we used for the LEO-BLIKSEM timings. Note, that the results for VAMPIRE and E-SETHEO reported in [14] differ for some of the problems to the ones in TPTP. This is probably due to different versions of the systems tested, for instance, the TPTP uses VAMPIRE version 7, while the results reported in [14] are based on version 5. The results in columns four through to seven show that some problems are still very hard for first-order ATPs, as well as for the special purpose theorem prover MUSCADET. Column eight and nine in Table 4 list the results for LEO alone and LEO-BLIKSEM, respectively. Each of these two columns is further divided into sub-columns to allow for a detailed comparison.

All our experiments (for the values of LEO and LEO-BLIKSEM) were conducted on a 2.4 GHz Xenon machine with 1GB of memory and an overall time limit of 100 seconds. For our experiments with LEO alone in column eight in Table 4 we tested four different strategies. Mainly, they differ in their treatment of equality and extensionality. This ranges from immediate expansion of primitive equality with Leibniz equality and limited extensionality reasoning, STANDARD (ST), to immediate expansion of primitive equality and moderate extensionality reasoning, EXT, to delayed expansion of primitive equality and moderate extensionality reasoning, EXT-INPUT (EI), and finally to delayed expansion of primitive equality and advanced recursive extensionality reasoning,

Table 4. Experimental data for the benchmark problems given in Table 1

TPTP-Problem	Mizar Problem	Difficulty	Saturate	Mus cadet	E-Setheo	Vampire 7	LEO Strat.	LEO Cl.	LEO Time	LEO-Bliksem Cl.	LEO-Bliksem Time	LEO-Bliksem FOcl	LEO-Bliksem FOtm	LEO-Bliksem GnCl
SET014+4		.67	+	+	+	.01	ST	41	.16	34	6.76	19	.01	7
SET017+1		.56	–	–	+	.03	EXT	3906	57.52	25	8.54	16	.01	74
SET066+1		1.00	?	–	–	–	–	–	–	26	6.80	20	10	56
SET067+1		.56	+	+	+	.04	ST	6	.02	13	.32	16	.01	12
SET076+1		.67	+	–	+	.00	–	–	–	10	.47	18	.01	35
SET086+1		.22	+	–	+	.04	ST	4	.01	4	.01	N/A	N/A	N/A
SET096+1		.56	+	–	+	.03	–	–	–	27	7.99	14	.01	25
SET143+3	B:67	.67	+	+	+	68.71	EIR	37	.38	33	7.93	18	.01	19
SET171+3	B:71	.67	+	+	–	108.31	EIR	36	.56	25	4.75	19	.01	20
SET580+3	B:23	.44	+	+	+	14.71	EIR	25	.19	6	2.73	8	.01	13
SET601+3	B:72	.22	+	+	+	168.40	EIR	145	2.20	55	4.96	8	.01	13
SET606+3	B:77	.78	+	–	+	62.02	EIR	21	.33	17	10.8	15	.01	5
SET607+3	B:79	.67	+	+	+	65.57	EIR	22	.31	17	7.79	15	.01	6
SET609+3	B:81	.89	+	+	–	161.78	EIR	37	.60	26	6.50	19	10	17
SET611+3	B:84	.44	+	–	+	60.20	EIR	996	12.69	72	32.14	38	.01	101
SET612+3	B:85	.89	+	–	–	113.33	EIR	41	.54	18	3.95	6	.01	7
SET614+3	B:88	.67	+	+	–	157.88	EIR	38	.46	19	4.34	16	.01	17
SET615+3	B:89	.67	+	+	–	109.01	EIR	38	.57	17	3.59	6	.01	9
SET623+3	B:99	1.00	?	–	–	–	EXT	43	8.84	23	9.54	10	.01	14
SET624+3	B:100	.67	+	–	+	.04	ST	4942	34.71	54	9.61	46	.01	212
SET630+3	B:112	.44	+	–	+	60.39	EIR	11	.07	6	.08	8	10	4
SET640+3	RS1:2	.22	+	–	+	70.41	EIR	2	.01	2	.01	N/A	N/A	N/A
SET646+3	RS1:8	.56	+	–	+	59.63	EIR	2	.01	2	.01	N/A	N/A	N/A
SET647+3	RS1:9	.56	+	–	+	64.21	EIR	26	.15	13	.30	13	.01	15
SET648+3	RS1:10	.56	+	–	+	64.22	EIR	26	.15	14	.30	13	.01	16
SET649+3	RS1:11	.33	–	–	+	63.77	EIR	45	.30	29	5.49	12	.01	16
SET651+3	RS1:13	.44	–	–	+	63.88	EIR	20	.10	11	.16	10	10	11
SET657+3	RS1:19	.67	+	–	+	1.44	EIR	2	.01	2	.01	N/A	N/A	N/A
SET669+3	RS1:19	.22	–	–	+	.34	EI	35	.22	35	.23	N/A	N/A	N/A
SET670+3	RS1:33	1.00	?	–	–	–	EXT	15	.17	17	.36	16	.01	6
SET671+3	RS1:34	.78	–	–	+	218.02	EIR	78	.64	7	2.71	10	.01	14
SET672+3	RS1:35	1.00	?	–	–	–	EXT	27	.4	30	.70	21	.01	11
SET673+3	RS1:36	.78	–	–	+	47.86	EIR	78	.65	14	5.66	14	.01	16
SET680+3	RS1:47	.33	+	–	+	.07	ST	185	.88	29	4.61	18	.01	24
SET683+3	RS1:50	.22	+	–	+	.06	ST	46	.20	35	8.90	18	10	24
SET684+3	RS1:51	.78	–	–	+	.33	ST	275	2.45	46	5.95	26	.01	47
SET686+3	RS1:53	.56	–	–	+	.11	ST	274	2.36	46	5.37	26	.01	46
SET716+4		.89	+	+	–	–	ST	39	.45	18	3.81	18	.01	118
SET724+4		.89	+	+	–	–	EXT	154	2.75	18	7.21	15	10	23
SET741+4		1.00	?	–	–	–	–	–	–	–	–	–	–	–
SET747+4		.89	–	+	–	–	ST	34	.46	25	1.11	18	10	10
SET752+4		.89	?	+	–	–	–	–	–	50	6.60	48	.01	4363
SET753+4		.89	?	+	–	–	–	–	–	15	3.07	12	10	19
SET764+4		.56	+	+	+	.02	EI	9	.05	8	.04	N/A	N/A	N/A
SET770+4		.89	+	+	–	–	–	–	–	–	–	–	–	–

EXT-INPUT-RECURSIVE (EIR). Column eight in Table 4 presents the fastest strategy for a respective problem (Strat.), the number of clauses generated by Leo (Cl.), and the total runtime (Time). While occasionally there were more than one Leo strategy that could solve a problem, it should be noted that none of the strategies was successful for all the problems solved by Leo.

In contrast to the experiments with Leo alone, we used only the EXT-INPUT strategy for our experiments with the Leo-Bliksem cooperation. Column nine in Table 4 presents the number of clauses generated by Leo (Cl.) together with the time (Time), and in addition, the number of first-order clauses sent to Bliksem (FOcl), the time used by Bliksem (FOtm), and the number of clauses generated

by Bliksem (GnCl). Note, that we give the data only for the first instance that Bliksem actually succeeded in solving the problem. This time also includes the time needed to write and process input and output files over the network. While Leo and instances of Bliksem were running in separate threads (each run of Bliksem was given a 50 second time limit), the figures given in the 'Time' column reflect the overall time needed for a successful proof. That is, it contains the time needed by all concurrent processes: Leo's own process as well as those processes administering the various instances of Bliksem. Since all these processes ran on a single processor, there is potential to ameliorate the overall runtimes by using real multiprocessing.

Note also, that the number of clauses in Leo's search space is typically low since subsumption is enabled. Subsumption, however, was not enabled for the accumulation of FO-like clauses in Leo's bag of FO-like clauses. This is why there are usually more clauses in this bag (which is sent to Bliksem) than there are available in Leo's search space. Finally, observe that for some problems a refutation was found after Leo's clausal normalisation, and therefore Bliksem was not applicable (N/A).

While Leo itself can solve a majority of the considered problems with some strategy, the Leo-Bliksem cooperation can solve more problems and, moreover, needs only a single Leo strategy. We can also observe that for many problems that appear to be relatively hard for Leo alone (e.g., SET017+1, SET611+3, SET624+3), the Leo-Bliksem cooperation solves them not only more quickly, but also it sometimes reduces the problems to relatively small higher-order preprocessing steps with subsequent easy first-order proofs, as for instance, in the case of SET017+1.

From a mathematical viewpoint the investigated problems are trivial and, hence, they should ideally be reliably and very efficiently solvable within a proof assistant. This has been achieved for the examples in Table 4 (except for SET741+4 and SET770+4) by our hybrid approach. While some of the proof attempts now require slightly more time than when using Leo alone with a specialised strategy, they are, in most cases, still faster than when proving with a first-order system.

5 Related Work and Conclusion

Related to our approach is the Techs system [12], which realises a cooperation between a set of heterogeneous first-order theorem provers. Similarly to our approach, partial results in Techs are exchanged between the different theorem provers in form of clauses. The main difference to the work of Denzinger *et al.* (and other related architectures like [13]) is that our system bridges between higher-order and first-order automated theorem proving. Also, unlike in Techs, we provide a declarative specification framework for modelling external systems as cooperating, concurrent processes that can be (re-)configured at run-time. Related is also the work of Hurd [15] which realises a generic interface between HOL and first-order theorem provers. It is similar to the solution

previously achieved by TRAMP [17] in OMEGA, which serves as a basis for the sound integration of ATPs into OANTS. Both approaches pass essentially first-order clauses to first-order theorem provers and then translate their results back into HOL resp. OMEGA. Some further related work on the cooperation of Isabelle with VAMPIRE is presented in [18]. The main difference of our work to the related systems is that while our system calls first-order provers from within higher-order proof search, this is not the case for [15,17,18].

One of the motivations for our work is to show that the cooperation of higher-order and first-order automated theorem provers can be very successful and effective. The results of our case study provide evidence for this: our non-optimised system outperforms related work on state-of-the-art first-order theorem provers and their ad hoc extensions such as SATURATE [14] on 45 mathematical problems chosen from the TPTP SET category. Among them are four problems which cannot be solved by any TPTP system to date. In contrast to the first-order situation, these problems can in fact be proved in our approach reliably from first principles, that is, without avoiding relevant base axioms of the underlying set theory, and moreover, without the need to provide *relevant* lemmata and definitions by hand.

The results of our case study motivate further research in the automation of higher-order theorem proving and the experimentation with different higher-order to first-order transformation mappings (such as the ones used by Hurd) that support our hybrid reasoning approach. They also provide further evidence for the usefulness of the OANTS approach as described in [8,5] for flexibly modelling the cooperation of reasoning systems.

Our results also motivate the need for a higher-order extension of the TPTP library in which alternative higher-order problem formalisations are linked with their first-order counterparts so that first-order theorem provers could also be evaluated against higher-order systems (and vice versa).

Future work is to investigate how far our approach scales up to more complex problems and more advanced mathematical theories. In less trivial settings as discussed in this paper, we will face the problem of selecting and adding relevant lemmata to avoid immediate reduction to first principles and to appropriately instantiate set variables. Relevant related work for this setting is Bishop's approach to *selectively expand definitions* as presented in [9] and Brown's PhD thesis on *set comprehension in Church's type theory* [10].

Acknowledgements For advice and help we thank Chad Brown, Andreas Meier, Andrei Voronkov, and Claus-Peter Wirth.

References

1. P. Andrews. *An Introduction to mathematical logic and Type Theory: To Truth through Proof.* Number 27 in Applied Logic Series. Kluwer, 2002.
2. C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving.* PhD thesis, Universität des Saarlandes, Germany, 1999.
3. C. Benzmüller. Extensional higher-order paramodulation and RUE-resolution. *Proc. of CADE-16*, *LNAI* 1632, p. 399–413. Springer, 1999.

4. C. Benzmüller. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–235, 2002.
5. C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Experiments with an Agent-Oriented Reasoning System. *Proc. of KI 2001*, *LNAI* 2174, p.409--424. Springer, 2001.
6. C. Benzmüller and M. Kohlhase. LEO – a higher-order theorem prover. *Proc. of CADE-15*, *LNAI* 1421. Springer, 1998.
7. C. Benzmüller and V. Sorge. A Blackboard Architecture for Guiding Interactive Proofs. *Proc. of AIMSA'98*, *LNAI* 1480, p. 102–114. Springer, 1998.
8. C. Benzmüller and V. Sorge. OANTS – An open approach at combining Interactive and Automated Theorem Proving. *Proc. of Calculemus-2000*. AK Peters, 2001.
9. M. Bishop and P. Andrews. Selectively instantiating definitions. *Proc. of CADE-15*, *LNAI* 1421. Springer, 1998.
10. C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Dept. of Mathematical Sciences, Carnegie Mellon University, USA, 2004.
11. H. de Nivelle. *The Bliksem Theorem Prover, Version 1.12*. Max-Planck-Institut, Saarbrücken, Germany, 1999.
`http://www.mpi-sb.mpg.de/~bliksem/manual.ps`.
12. J. Denzinger and D. Fuchs. Cooperation of Heterogeneous Provers. *Proc. IJCAI-16*, p. 10–15. Morgan Kaufmann, 1999.
13. M. Fisher and A. Ireland. Multi-agent proof-planning. CADE-15 Workshop "Using AI methods in Deduction", 1998.
14. H. Ganzinger and J. Stuber. Superposition with equivalence reasoning and delayed clause normal form transformation. *Proc. of CADE-19*, *LNAI* 2741. Springer, 2003.
15. J. Hurd. An LCF-style interface between HOL and first-order logic. *Automated Deduction — CADE-18*, *LNAI* 2392, p. 134–138. Springer, 2002.
16. M. Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic*. PhD thesis, Universität Kaiserslautern, Germany, 1992.
17. A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. *Proc. of CADE-17*, *LNAI* 1831. Springer, 2000.
18. J. Meng and L. C. Paulson. Experiments on supporting interactive proof using resolution. *Proc. of IJCAR 2004*, *LNCS* 3097, p. 372–384. Springer, 2004.
19. R. Nieuwenhuis, Th. Hillenbrand, A. Riazanov, and A. Voronkov. On the evaluation of indexing techniques for theorem proving. *Proc. of IJCAR-01*, *LNAI* 2083, p. 257–271. Springer, 2001.
20. D. Pastre. Muscadet2.3 : A knowledge-based theorem prover based on natural deduction. *Proc. of IJCAR-01*, *LNAI* 2083, p. 685–689. Springer, 2001.
21. A. Riazanov and A. Voronkov. Vampire 1.1 (system description). *Proc. of IJCAR-01*, *LNAI* 2083, p. 376–380. Springer, 2001.
22. V. Sorge. *OANTS: A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*. PhD thesis, Universität des Saarlandes, Germany, 2001.
23. G. Stenz and A. Wolf. E-SETHEO: An Automated^3 Theorem Prover – System Abstract. *Proc. of the TABLEAUX'2000*, *LNAI* 1847, p. 436–440. Springer, 2000.
24. G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving

Christoph Benzmüller* and Volker Sorge†
(C.E.Benzmuller@cs.bham.ac.uk, sorge@ags.uni-sb.de)

School of Computer Science, The University of Birmingham
Edgbaston, Birmingham B15 2TT, England

Fachbereich Informatik, Universität des Saarlandes,
D-66041 Saarbrücken, Germany

Abstract. *We present the* Ω-ANTS *theorem prover that is built on top of an agent-based command suggestion mechanism. The theorem prover inherits beneficial properties from the underlying suggestion mechanism such as run-time extendibility and resource adaptability. Moreover, it supports the distributed integration of external reasoning systems. We also discuss how the implementation and modeling of a calculus in our agent-based approach can be investigated wrt. the inheritance of properties such as completeness and soundness.*

1 Introduction

We present the new Ω-ANTS automated theorem proving approach that is build on top of the Ω-ANTS agent-based command suggestion mechanism. This mechanism has been originally developed to support the user in interactive theorem proving by using available resources in-between user interactions to search for the next possible proof steps [4]. This is done via a hierarchical blackboard-architecture where agents concurrently check for applicable commands (i.e. commands that apply proof rules) and the most promising commands are dynamically presented to the user. The faster a command's applicability can be analysed the faster it will be reported immediately to the user. Further benefits of the distributed Ω-ANTS

*The author would like to thank EPSRC for its support by grant GR/M99644.
†The author's work was supported by the 'Studienstiftung des deutschen Volkes'.

command suggestion mechanism are the increased robustness (errors in the distributed computations do not harm the overall mechanism), its resource- and user-adaptability, and its run-time extendibility and modifiability [5].

In this paper we present how we achieve the automation of Ω-ANTS. On the one hand the concurrency enables the integration of external reasoning systems into the suggestion process. External reasoners can either be used to suggest whole subproofs or to compute particular arguments of commands. On the other hand Ω-ANTS can be automated directly by executing suggested commands automatically instead of just presenting them to the user. Thereby it is important to restrict the set of involved commands to those suitable for automation and to fix a certain clock speed determining the period of time the suggestion mechanism may maximally consume for its computations in-between the automated command executions. In any proof state where Ω-ANTS cannot find any new applicable commands it simply backtracks by retracting the lastly executed command.

The Ω-ANTS suggestion mechanism and the Ω-ANTS theorem prover have been developed and implemented within the ΩMEGA theorem proving environment [17]. However, the approach is not restricted to a particular logic, calculus, or theorem proving environment. It can be rather seen as an approach parameterised over the particular calculus it is working for. In this respect the question arises how the designer of the Ω-ANTS agents which have to be provided for each calculus rule can ensure that the modeling guarantees a complete proof search in Ω-ANTS. This question is discussed in the second half of the paper by informally defining properties of agent societies in Ω-ANTS which are necessary to ensure completeness and by giving some examples how these properties are checked in practice.

This paper is organised as follows: Sec. 2 sketches the Ω-ANTS command suggestion mechanism (for further details see [4, 5]), illustrates its declarative agent specification language, and sketches a formal semantics. Sec. 3 describes how external reasoners can be integrated at different layers. In Sec. 4 the Ω-ANTS theorem prover built on top of the suggestion mechanism is introduced and completeness aspects are discussed in Sec. 5. We conclude with discussing some related work and hinting at future work.

2 The Ω-ANTS suggestion mechanism

In this section we sketch the hierarchical, agent-based suggestion mechanism underlying the Ω-ANTS theorem prover. We also discuss the declarative agent specification language supporting the specification and modification of agents at run-time, and sketch how this language can be linked

to a formal semantics.

Agent-based architecture The suggestion mechanism originally aims at supporting a user in interactive tactical theorem proving to choose an appropriate proof rule from the generally large set of available ones. It computes and proposes commands that invoke proof rules that are applicable in a given proof state.[1] This is basically done in two steps: firstly, by computing whether there are any possible instantiations for single arguments of a command in the current proof state; and secondly, by gathering those commands for which at least some arguments could be instantiated and presenting them in some heuristically ordered fashion to the user.

An important notion for the Ω-ANTS mechanism is that of a *Partial Argument Instantiations (PAI)* for a command. Considering a command and its corresponding proof rule there is usually a strong connection between the formal arguments of both, i.e. the formal arguments of the command are generally a subset of the formal arguments of the proof rule. As an example we observe the proof rule ∧I and its corresponding command `AndI`:

$$\frac{A \quad B}{A \wedge B}\ \wedge\mathrm{I} \longrightarrow \frac{\texttt{LConj} \quad \texttt{RConj}}{\texttt{Conj}}\ \texttt{AndI}$$

Here the command's formal argument `Conj` needs to be instantiated with an open proof node containing a conjunction, `LConj` and `RConj` with nodes containing the appropriate left and right conjuncts, respectively. In general, a command's formal arguments need to be partially instantiated only, in order to be applicable. For instance, `AndI` is also applicable if only the `Conj` argument is provided, resulting in the introduction of two new open proof nodes containing the two conjuncts. Or additionally one of `LConj` or `RConj` or even both could be provided, resulting in the introduction of only one open node or in simply closing the given open conjunction. Thus, we can denote partial argument instantiations for a command as a set relating some of a command's formal argument to actual arguments for its execution. One possible PAI for `AndI` would be (`Conj` : x,`LConj` : y) where x and y are proof nodes that contain the appropriate formulas. PAIs can also be seen as functions, indexed by the different command names, with the set of argument-names as domain and the infinite set of possible proof lines and parameters as codomain. For instance, PAIs for `AndI` can be represented as particular functions

$$\mathrm{PAI}^{\texttt{AndI}} : \{\texttt{Conj, LConj, RConj}\} \longrightarrow \mathrm{Prooflines} \cup \mathrm{Parameters} \cup \{\epsilon\}$$

[1] For the remainder of the paper, if we talk about applicability of a command we always mean the applicability to the corresponding proof rule (e.g., a calculus rule, a tactic, a proof planning method, or an external system call) in the given proof state.

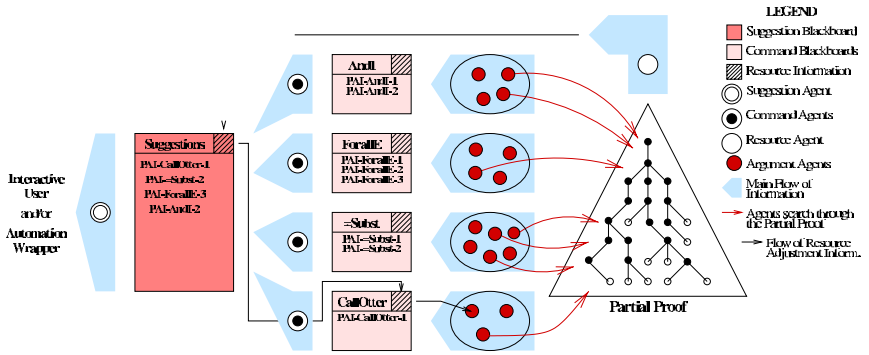


Figure 1. The hierarchical, agent-based Ω-ANTS-architecture.

where ϵ is a special symbol denoting the empty proofline. In these sense the PAI $(\texttt{Conj} : x, \texttt{LConj} : y)$ for `AndI` is realised by a respective function such that $\text{PAI}^{\texttt{AndI}}(\texttt{Conj}) = x$, $\text{PAI}^{\texttt{AndI}}(\texttt{LConj}) = y$, and $\text{PAI}^{\texttt{AndI}}(\texttt{RConj}) = \epsilon$.

The idea of the suggestion mechanism is to compute in each proof state for each command PAIs as complete as possible, to determine which commands are applicable, and then to give preference to those, e.g., with the most complete PAIs. The first task is done by societies of *Argument Agents* (rightmost circles in Fig. 1) where one society is associated with each of the commands. Each argument agent is associated with one or several of the command's formal arguments and has a specification for possible instantiations of these arguments. Its task is to search for proof nodes in the partial proof or to compute parameters according to its specification. Argument agents exchange results via *Command Blackboards* (for each command one command blackboard is provided) using PAIs as messages. Every argument agent commences its computations only when it finds a PAI on the command blackboard that contains instantiations of arguments that are relevant for its own computations.

For example, the `AndI` argument agent associated with Conj searches the partial proof for an open node containing a conjunction and, once it has found one, say in node x, it places a respective PAI $(\texttt{Conj} : x)$ on the command blackboard. Now the agents for `LConj` and `RConj` can use this result in order to look simultaneously for nodes in the given partial proof containing the appropriate left or right conjunct. Each argument agent only reads old suggestions and possibly adds expanded new suggestions, thus there is no need for conflict resolutions between the agents.

On top of the layer of argument agents are the *Command Agents* (dotted circles). Their task is to monitor the command blackboard associated with

the command and to heuristically order the PAIs from most promising (e.g., most complete) to least promising. Whenever their heuristics indicate that there is a new best PAI on the command blackboard they pass it to the *Suggestion Blackboard*. The suggestion blackboard itself is again monitored by the *Suggestion Agent* (leftmost double circle) which sorts the entries with respect to its heuristic criteria and presents them to the user.

When the Ω-ANTS mechanism is started all command blackboards are initialised with the empty PAI. The agents then autonomously search for applicable commands and the newest suggestions are successively presented to the user. At any point a command can be executed and when the proof state has actually been changed the Ω-ANTS mechanism is re-initialised in order to compute new suggestions for the modified proof state. Ω-ANTS can also be used to respond to particular user queries, i.e. the user can interactively specify certain argument instantiations and the mechanism tries to complete these.

The whole mechanism can be adjusted during run-time by changing sorting heuristics for the command blackboards and the suggestion blackboard or by removing, adding, or modifying argument agents. Moreover, Ω-ANTS employs a resource mechanism that automatically disables and enables argument agents with respect to their usefulness and performance in particular proof states. Although not depicted here, the mechanism also contains classification agents whose purpose is to classify the focused subproblem in terms of logic and mathematical theory is belongs to. This information is communicated within the blackboard architecture enabling agents to decide whether they are appropriate (i.e. should be active) in the current proof state or not. See [5] for further details.

A Declarative Agent Specification Language In Ω-ANTS only the argument agents need to be explicitly specified. All other agents are then generated automatically (certain heuristics may be adapted by the user, though). Argument agents are implemented with a Lisp-like declarative language such as the following two argument agents for the `AndI` command:

$\mathfrak{A}_1$:
```
(agent~defagent AndI c-predicate
   (for Conj) (uses )
   (exclude LConj RConj)
   (definition
     (logic~conjunction-p Conj)))
```

$\mathfrak{A}_4$:
```
(agent~defagent AndI s-predicate
   (for RConj) (uses Conj)
   (definition
     (logic~right-conjunct-p RConj Conj))))
```

The agent $\mathfrak{A}_1$ is defined as a `c-predicate` agent, indicating that it will always restrict its search to open proof nodes, i.e., possible conclusions. `s-predicate` agents like $\mathfrak{A}_4$ in contrast search the support nodes for possible premises. The proof nodes $\mathfrak{A}_1$ is looking for are instantiations of the argument `Conj`, given in the `for`-slot. The empty `uses`-slot indicates that

$$
\begin{array}{lll}
\mathfrak{A}_1\colon & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\},\{\mathrm{LConj},\mathrm{RConj}\}} & := \quad \lambda \mathit{Conj}_{\blacksquare}(\mathit{Conj} \equiv A \wedge B) \\
\mathfrak{A}_2\colon & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{LConj}\},\{\mathrm{RConj}\}} & := \quad \lambda \mathit{Conj}_{\blacksquare}(\mathit{Conj} \equiv A \wedge B) \,\&\, (\mathrm{LConj} \equiv A) \\
\mathfrak{A}_3\colon & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{RConj}\},\{\mathrm{LConj}\}} & := \quad \lambda \mathit{Conj}_{\blacksquare}(\mathit{Conj} \equiv A \wedge B) \,\&\, (\mathrm{RConj} \equiv B) \\
\mathfrak{A}_4\colon & \mathfrak{S}^{\{\mathrm{RConj}\}}_{\{\mathrm{Conj}\},\{\}} & := \quad \lambda \mathit{RConj}_{\blacksquare}(\mathrm{Conj} \equiv A \wedge B) \,\&\, (\mathit{RConj} \equiv B) \\
\mathfrak{A}_5\colon & \mathfrak{S}^{\{\mathrm{LConj}\}}_{\{\mathrm{Conj}\},\{\}} & := \quad \lambda \mathit{LConj}_{\blacksquare}(\mathrm{Conj} \equiv A \wedge B) \,\&\, (\mathit{LConj} \equiv A) \\
\mathfrak{A}_6\colon & \mathfrak{C}^{\{\mathrm{Conj}\}}_{\{\mathrm{LConj},\mathrm{RConj}\},\{\}} & := \quad \lambda \mathit{Conj}_{\blacksquare}(\mathit{Conj} \equiv A \wedge B) \,\&\, (\mathrm{LConj} \equiv A) \,\&\, (\mathrm{RConj} \equiv B)
\end{array}
$$

Figure 2. A society of argument agents for command `AndI`.

the agent does not require any already given argument suggestions in a PAI for its computations. The `exclude`-slot on the other hand determines that this agent must not complete any PAI that already contains an instantiation for arguments `LConj` or `RConj`. In the special case of $\mathfrak{A}_1$ this means the agent is exactly triggered by the empty PAI. The idea for this exclusion constraint is to suppress redundant or even false computations.

The full set of argument agents for the `AndI` command is given in Fig. 2 in a specification meta-language. `c-predicate` and `s-predicate` agents are denoted by $\mathfrak{C}$ and $\mathfrak{S}$ respectively, the superscript set corresponds to the `for`-list, and the `uses`- and `exclude`-list to the first and second index. The subset of the nodes in a partial proof that will be detected by each argument agent can be formally described by a λ-term (characteristic function). When running over the partial proof the agents use these characteristic functions to test each node before possibly returning an expanded PAI. A and B are free meta-variables. $\equiv$ and & are symbols of the meta-language with the meaning, for instance in agent $\mathfrak{A}_2$, that given an arbitrary formula A instantiating argument LConj then Conj has to be of form $A \wedge B$, i.e. the left hand side of Conj is determined by the already given suggestion LConj whereas is right hand side is still *free*.

This attempt at a formal semantics for our agent definitions by assigning characteristic functions to them does not yet address the agents functional behaviour (they pick up & return potentially modified PAIs) nor does it formally regard the uses and exclude-restrictions. This is the idea of the λ-expression for agent $\mathfrak{A}_2$ below. Assuming that PAIs are represented as functions this term denotes that $\mathfrak{A}_2$ picks up certain PAIs on the blackboard and returns possibly modified ones while using an (extended/modified) characteristic function in the previous sense as filter. Here the [.]-brackets denote a function which accesses the formula content of the proofline given as an argument to it (note that PAIs map argument names to prooflines, while here we want to talk about the formulas of the prooflines[2]).

[2]In other parts of this paper we do not take this so serious and assume that the user

$\lambda\, PAI \,.\, \lambda\, Conj_{\text{open}} \,.$
if $PAI(\text{Conj}) \equiv \epsilon$ & $PAI(\text{LConj}) \not\equiv \epsilon$ & $PAI(\text{RConj}) \equiv \epsilon$
then if $[Conj] \equiv$ A $\wedge$ B & $[PAI(\text{LConj})] \equiv$ A
then $PAI|_{\{\text{LConj},\ \text{RConj}\}} \cup \{\text{Conj} \mapsto Conj\}$ $\rightarrow$ new ext. PAI
else PAI $\rightarrow$ no new PAI
fi
else PAI $\rightarrow$ no new PAI
fi

3 Integration of External Reasoning Systems

The following four examples illustrate how external reasoners can be integrated into Ω-ANTS. The first row presents four inference rules and the second the corresponding commands which we want to model in Ω-ANTS.

$$\frac{P_1 \quad \ldots \quad P_n}{C}\ \begin{matrix}\textit{Otter}\\ \textit{Mace}\end{matrix} \qquad \frac{A \quad B \Rightarrow C}{C}\ \begin{matrix}\textit{mp-mod-Otter}(A \Rightarrow B)\\ \textit{mp-mod-CAS}(A \overset{\text{simpl}}{\Rightarrow} B)\end{matrix}$$

$$\frac{\texttt{Prem}_1 \quad \ldots \quad \texttt{Prem}_n}{\texttt{Conc}}\ \begin{matrix}\texttt{Otter}\\ \texttt{Mace}\end{matrix} \qquad \frac{\texttt{Left} \quad \texttt{Impl}}{\texttt{Conc}}\ \begin{matrix}\texttt{mp-mod-Otter(Impl-Prob)}\\ \texttt{mp-mod-CAS(Simpl-Prob)}\end{matrix}$$

The first two rules describe the integration of the first-order theorem prover OTTER and the propositional logic decision procedure MACE. These commands may be used in a given proof state in order to justify a goal from its premises by the application of one of these external systems. The next two rules describe a situation where external reasoners are used within an inference modulo, in our particular case modus ponens modulo the validity of an implication (to be checked by OTTER) and modus ponens modulo the simplifiability of a proposition (to be analyzed by a computer algebra system). For instance, sensible instances of these commands in a concrete proof situation would be: `Left` $\leftarrow \forall x.p(x) \wedge q(x)$, `Impl` $\leftarrow p(a) \Rightarrow r(a)$, `Conc` $\leftarrow r(a)$, and `Impl-Prob` $\leftarrow (\forall x.p(x) \wedge q(x)) \Rightarrow p(a)$ for `mp-mod-Otter`, and `Left` $\leftarrow$ continuous$(\lambda x.1 - \cos^2(x))$, `Impl` $\leftarrow$ continuous$(\lambda x.\sin^2(x)) \Rightarrow$ something$(\lambda x.\sin^2(x))$, `Conc` $\leftarrow$ something$(\lambda x.\sin^2(x))$, `Simpl-Prob`$\leftarrow$continuous $(\lambda x.1 - \cos^2(x)) \overset{\text{simpl}}{\Rightarrow}$ continuous$(\lambda x.\sin^2(x))$ for `mp-mod-CAS`. The idea is that the external systems are used to check the 'modulo'-side-conditions of these rules. Note, that in contrast to the theorem proving modulo approach described in [9] we explicitly facilitate and support the integration of non-decision procedures; a strict separation of deduction and computation is not needed due to the distribution and resource-guidance aspect of the Ω-ANTS mechanism.

We are here not concerned with correctness issues for the integration of the external systems. However, since we are working in the ΩMEGA

recognises whether we address a proof line or its formula content from the context.

environment we can make use of the work already done in this area that ensures the correctness by translating proofs or computations from external reasoners into primitive inference steps of ΩMEGA [16, 19].

If we, for example, consider the `Otter` and the `mp-mod-CAS` command we can observe two different ways of integrating the external reasoners into agents: For the `Otter` command one agent attacks the focused open goal in-between user interactions and as soon as OTTER finds a proof the application of this command is suggested to the user. Thus, the agent employs the external system to prove an open sub-problem. Similarly, other external reasoners can be integrated.

In case of the `mp-mod-CAS` command an agent will first look for appropriate implication proof nodes with respect to the open goal. This agent's results trigger another agent that which employs an integrated computer algebra system to look for appropriate proof nodes as instances for argument `Left`. More precisely, the latter agent checks whether a proof node can be matched with the antecedent of the implication with respect to algebraic simplification of sub-terms. Hence, the agent uses an external reasoner only to find possible instantiations of arguments.

4 Automation

The Ω-ANTS suggestion mechanism of Sec. 2 can be automated into a full-fledged proof search procedure by embedding the execution of suggested commands into a backtracking wrapper. The algorithm is given in Fig. 3.

The basic automation performing a depth first search is straight forward: The suggestion mechanism waits until all agents have performed all possible computations and no further suggestions will be produced and then executes the heuristically preferred suggestion (1a&2). When a proof step is executed and the proof is not yet finished, the remaining suggestions on each command blackboard are pushed on the backtracking stack (3). In case no best suggestion could be computed Ω-ANTS backtracks by popping the first element of the backtrack stack and re-instantiating its values on the blackboards (6). The proof is constructed as an explicit proof plan data structure of ΩMEGA [8]. It enables to store proofs in a generalised natural deduction format, i.e. proof steps cannot only be justified by basic natural deduction rules but by abstract tactics or computations of external reasoners as well. Moreover, the proof plan data structure supports the expansion of externally computed proofs into primitive inference steps and thus the check for correctness as well as the storage of information for the automation loop directly in the proof object.

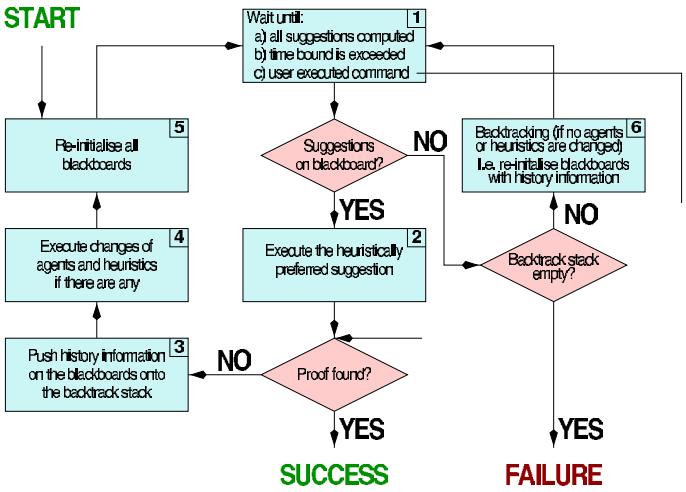


Figure 3. Main-loop of the Ω-ANTS theorem prover.

The simple automation loop is complicated by the distinct features of Ω-ANTS: (i) some agents can perform infinite or very costly computations, (ii) commands can be executed by the user parallel to the automation, and (iii) the components of Ω-ANTS can be changed at run-time. Furthermore, the automation can be suspended and revoked especially in order to perform the latter two interaction possibilities in a coordinated way.

We avoid that Ω-ANTS is paralysed by agents that get stuck in infinite computations by giving a time limit after which the best command, suggested so far, is executed (cf. step 1b). However, such a proof step is treated special when backtracking, since then the blackboards will be re-instantiated with all the values of the proof step, i.e. containing the executed command as well. This way there is a second chance for agents that could not contribute the first time to add information. The question how the Ω-ANTS theorem prover can avoid to get lost on an infinite branch in the search space without ever backtracking will be addressed in the completeness discussion in Sec. 5.

If a command has been executed by the user the loop proceeds immediately with saving the blackboards' history without executing another command (1c). When backtracking the whole history on the last step is re-instantiated onto the blackboards, possibly containing also the command executed by the user, in order not to loose possible proofs (1c&6).

One main feature of Ω-ANTS is its run-time adaptability by adding or deleting agents or changing the filter and sorting heuristics used by the suggestion and commands agents. These changes also take effect when

running the automation wrapper (4). The automation wrapper can be suspended by the user at any time, for instance, in order to analyzed the current proof state and to add, change or remove certain agents from the suggestion mechanism. It can then be resumed using all the information computed so far.

We briefly summarise the user interaction facilities inherited by the Ω-ANTS prover from the Ω-ANTS suggestion mechanism:

Pure user interaction/mixed initiative reasoning: In automation mode the entries on the suggestion blackboard are (theoretically[3]) steadily visible to the user, who can interfere with the automation wrapper by executing a command before the automation wrapper does.

Adjustment of resource bounds: The user may want to actively modify the resource bounds (time, memory, deactivation threshold) in order to adapt the system to particular needs.

Disable/resume single agents: Ω-ANTS allows to disable/resume single agents, agent societies, or the whole mechanism at run-time.

Modification/addition of argument agents: The user may want to specify and load new agents at run-time or modify the definition of already given agents. This is supported by the declarative agent-specification language.

Modification of command/suggestion agents: In order to influence the provers search through the search space the user may want to choose different heuristics and sorting criteria for these agents.

The Ω-ANTS system has been applied to automate the propositional logic fragment of the normal form natural deduction calculus NIC [7], see [2] for more details. We currently experiment with the full first-order fragment of NIC. The integration of external reasoners has been tested with the propositional logic prover MACE, the first-order provers OTTER and SPASS, the higher-order prover TPS, and the computer algebra system MAPLE. The theorems we are working with are still all relatively simple and nothing any of the involved systems is not able to solve on its own. The computations involved are mainly to solve equations and compute derivatives.

5 Ω-ANTS and Completeness

In this section we introduce and discuss some notions that are necessary to characterise and guarantee completeness and soundness of a theorem prover based on Ω-ANTS with respect to the underlying calculus. The discussion

[3] In our experiments with the NIC calculus, the theorem prover is unfortunately much faster than the graphical user interface to allow a synchronised displaying.

is rather informal since we have yet to define completely formal syntax and semantics for our agent specification language. However, the following shall both give an intuition for the properties that need to be considered and contributes to a better understanding of Ω-ANTS.

Given a theoretically complete calculus, how can it be modeled in Ω-ANTS such that completeness is still assured in the mechanism? Note, that we do not address the theoretical completeness of the underlying calculus itself, in fact we do not even need to specify here what particular logic and calculus we are interested in. We rather aim to ensure that each calculus rule application that is theoretically possible in a given proof state can indeed be determined and suggested by the Ω-ANTS mechanism. In particular we will discuss two different notions of completeness in this sense, namely *interaction completeness* and *automation completeness*. This is due to twofold bias of the Ω-ANTS system as a suggestion mechanism and as an automated theorem prover. The authors admit that naming these properties also 'completeness' might be slightly misleading. However, automation (interaction) completeness of the agent societies involved taken together with the 'theoretical (logical) completeness' of a calculus implies that a complete proof search is actually supported by Ω-ANTS.

Theoretical completeness investigations typically assume non-limited resources like computation time and space. In our case the resources available to the Ω-ANTS-system in-between the command executions are crucial wrt. completeness as well. However, for the time being we neglect points possibly interfering with this assumption, in particular cases 1(b) or 1(c) of the prover's main-loop in Fig. 3 and the existence of agents with calls to undecidable procedures such as the OTTER agent in Sec. 3.

Automation Completeness Automation completeness depends in the first place on the *suggestion completeness* of the argument agent societies associated with each rule: A society of suggestion agents working for a single command C is called *suggestion complete* wrt. a given calculus, if in any possible proof state all PAIs of a command necessary to ensure completeness of the calculus can be computed by the mechanism. Under the resource abstraction assumption from above suggestion completeness requires that each particular agent society consists of *sufficiently* many individual suggestion agents and that their particular definitions are *adequate* wrt. the structural dependencies and side-conditions of the respective calculus rule. *Adequacy* basically excludes wrong agent specifications, while *Sufficiency* refers to the ability of an agent society to cooperatively compute each applicable PAI in a given proof state.

We call a command agent *non-excluding* if it indeed always reports at least one selected entry from the associated command blackboard to the

suggestion blackboard as soon as the former contains some applicable PAIs. And the suggestion agent is non-excluding if it always reports the complete set of entries on the command blackboard to the automation wrapper. This ensures that computed PAIs are actually propagated in the mechanism.

We additionally have to ensure that the proof search is organised in a *fair* way by ensuring that the execution of an applicable PAI suggested within a particular proof step cannot be infinitely long delayed. The fairness problem of Ω-ANTS is exactly the same as in other theorem proving approaches performing depth first search. In our experiments with the propositional logic fragment of the NIC this problem did not occur as the considered fragment defines a decision procedure. However to ensure that the prover does not get lost on infinite search path when working with the full first-order fragment of NIC we chose iterative deepening search.

Our mechanism can then be called automation complete wrt. to a given calculus C if (i) the agent societies specified are suggestion complete wrt. C, and (ii) the command agents for C and the suggestion agent are non-excluding, (iii) the search procedure is fair and (iv) the resource bounds and deactivation threshold are chosen sufficiently high, such that each agents computation terminates within these bounds.

We illustrate the notions of adequacy and sufficency in more detail with the example of the `AndI` agents. We claim that the agents $\mathfrak{A}_1 \ldots \mathfrak{A}_6$ of Fig. 2 are both (a) adequate and (b) sufficient to apply `AndI` (whenever possible) in automated proof search.

(a) To show that all computable suggestions are indeed applicable we check that each agent produces an adequate predicate if all arguments of the uses slot are instantiated correctly. We observe this in the case, of agent $\mathfrak{A}_2$ when applying it to a PAI of the form `(LConj:`a`)`. Here a is an arbitrary but fixed term. The resulting predicate is $Conj \equiv a \wedge B$ which permits all conjunctions with left conjunct a and is therefore adequate.

After checking adequacy of all single agents we have to ensure adequacy of cooperation between agents. That is, to show that no incorrect PAIs can be assembled by cooperation of agents with correct predicates. Here we are only concerned with agents whose `for-`, `uses-`, and `exclude`-list does not contain all possible arguments of the command, thus in our case agents $\mathfrak{A}_4$ and $\mathfrak{A}_5$. It can be easily seen that even if, for instance, $\mathfrak{A}_4$ is applied to a PAI already containing an instantiation for `LConj`, adding an appropriate instantiations for `RConj` will maintain the PAI's applicability, provided it was correct to begin with.

(b) To ensure sufficiency we have to show that each PAI of `AndI` necessary for automation can (cooperatively) be computed. In automatic mode the NIC calculus is intended for pure backward search and thus the possi-

ble PAIs are of the form[4] i) (Conj:$a \wedge b$), ii) (Conj:$a \wedge b$, LConj:a), iii) (Conj:$a \wedge b$, RConj:b), or iv) (Conj:$a \wedge b$, LConj:a, RConj:b), where a and b are arbitrary but fixed formulas occurring in a partial proof P. We representatively discuss case ii) and verify that each PAI of form S = (Conj:$a \wedge b$, LConj:a) that is applicable in P will actually be computed. As S is applicable, P must contain an open node containing $a \wedge b$ together with a support node containing a. Initially the command blackboard contains the empty PAI () to which only $\mathfrak{A}_1$ can be applied. Provided the underlying implementation, i.e. the function logic~conjunction-p is correct, $\mathfrak{A}_1$'s predicate suffices to compute (Conj:$a \wedge b$). This PAI in turn triggers the computations of $\mathfrak{A}_4$ and $\mathfrak{A}_5$ with the respective instantiated predicates *RConj*$\equiv b$ and *LConj*$\equiv a$. Since the latter is true on the support node containing a, $\mathfrak{A}_5$ returns the PAI in question.

When checking all other cases we can observe that for the automation mode (where pure backward reasoning is assumed) the agents $\mathfrak{A}_1$,$\mathfrak{A}_4$, and $\mathfrak{A}_5$ are already sufficient. And indeed the other three agents are needed to support user interaction, only. For instance, the user can apply Ω-ANTS to complete a particular PAI like (LConj:a) which will trigger the computations of agent $\mathfrak{A}_2$.

Interaction Completeness Interaction completeness of a calculus implies that one never has to rely on another interaction mechanism besides Ω-ANTS in order to perform possible proof steps within a given calculus. Therefore, we have to show that all possible PAIs to apply a rule interactively can be computed. This is generally a stronger requirement than for automation completeness as can be easily observed with our AndI example. When automated the NIC calculus strictly performs backward search and only the PAIs (i)—(iv) given above are legitimate. However, when using the calculus interactively forward reasoning (i.e. a PAI of the form (LConj:a, RConj:b)) is a perfectly legal option. But it can be easily seen that this PAI cannot be computed with the given agent society and thus $\{\mathfrak{A}_1 \ldots \mathfrak{A}_6\}$ are not interaction complete.

When dealing with interaction completeness we have also to consider all possible initialisations of the command blackboards. While in automation mode the blackboards are always initialised with the empty PAI, the user can ask Ω-ANTS interactively to complete a particular PAI (such as (LConj:a)) which is then used as initial value on the blackboard. It is necessary to show sufficiency and adequacy for all possible initialisations.

Soundness Should not the *soundness* aspect be addressed here as well? Our answer is no, as we presuppose that the underlying theorem proving

[4] PAIs are essentially sets and thus the order of the particular entries is not important.

environment takes care of a sound application of its own proof rules. Furthermore, in systems such as ΩMEGA soundness is always only guaranteed on the level of primitive inferences and not necessarily for all proof methods etc. involved. Thus, soundness requirements when computing suggestions for methods that do not necessarily lead to a correct proof would not make sense. Thus, instead of *logical soundness* we are rather interested in the notion of *applicability*. This notion relates the PAIs computed by Ω-ANTS to the particular side-conditions of the underlying proof rules (whether they are logically sound or not).

The effect of non-applicable PAIs suggested to the user or the automation wrapper might lead to failure when applying the respective command. In the current implementation such a failure will simply be ignored and the responsible PAI is discarded. However, too many non-executable suggestions might negatively influence the mechanisms user-acceptance and especially the performance of the automation wrapper.

6 Related Work

There exist several theorem proving environments where a mixture of interactive and partial automated proving is supported. In systems such as PVS [18] and HOL [13] special tactics are available that can be used to automatically solve certain problems. These tactics are essentially proof procedures build on top of the primitive inferences of the respective systems but do not directly construct a proof in terms of primitive inferences, although the automated parts can be, at least in the case of HOL, expanded. Moreover, there is no possibility for a user of the system to change the behaviour of the automation tactic during its application. In the TPS system [1] interaction and automation can also be interleaved and any automatic proving attempt can be interrupted, its behaviour changed and restarted by the user. The automation is achieved by using a mating search technique that is substantially different from the natural deduction calculus that is used for interactive proving. Finally, an approach to achieve automation in an interactive environment is to enable the use of external reasoners which is, for instance, one of the features of the ΩMEGA system [17]. However, without the Ω-ANTS part, application of rules, tactics and external reasoners cannot be automated.

As an environment that is especially designed to support the combination of interactive and automated theorem proving together with the use of already existing reasoning systems, is the Open Mechanised Reasoning System [12, 11] that has been extended to facilitate computer algebra sys-

tems [6]. While the concept of a reasoning structure to represent explicit proof states is similar to our concept of a proof object, external reasoners are connected as *plug-and-play* components which requires significant changes to their control components and therefore complicates the use of existing technology.

7 Conclusion

We presented the Ω-ANTS theorem prover build on top of the agent-based Ω-ANTS suggestion mechanism. This theorem prover inherits interesting features from the underlying suggestion mechanism and due to the distribution of computations down to a very fine-grained layer (e.g. reasoning about potential instances of single arguments of the considered inference rules) it especially supports the integration of external reasoning systems at various layers. We have illustrated that the Ω-ANTS architecture especially supports deduction modulo computation/deduction performed by external reasoners. As the same suggestion mechanism that supports user-interaction is now also used as the main part of the automated theorem prover's inference machine the architecture also supports a close integration of interactive and automated theorem proving. This is underlined by the various interaction facilities the Ω-ANTS prover already supports. The system can be seen as an open approach that is parameterised over the particular calculus it is working for (and note that it is only in a technical sense restricted to the ΩMEGA environment in which it has been developed). The calculus it is working for can even be modified/extended at run-time, making our system in the long-run also interesting for the integration of components aiming at learning new inference rules from past proof experience [15]. The learned rules could then be dynamically added to the running system.

Immediate further work is a more rigorous formalisation of the agent specification language as well as to formally model the connection between Ω-ANTS and underlying calculi. Other future work is to analyse whether our system could benefit from a dynamic agent grouping approach as described in [10] and whether it can fruitfully support the integration of proof critics as discussed in [14]. The Ω-ANTS system is also employed as the basis of the resource-guided and agent-based proof planning approach [3], currently under development. Extending the Ω-ANTS system this approach also focuses on the cooperation aspect between integrated external reasoners and addresses the question how an agent-based proof planner can be sensibly guided by a resource mechanism.

Acknowledgement The authors thank John Byrnes for his support in realizing the NIC calculus in Ω-ANTS. We furthermore thank S. Autexier, M. Kerber, M. Jamnik, and M. Hübner for fruitful discussions.

References

[1] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[2] C. Benzmüller, J. Byrnes, and V. Sorge. Ω-ANTS for interactive ATP. Unpublished draft: `www.ags.uni-sb.de/~chris/oants-nic00.ps.gz`.

[3] C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge Towards Concurrent Resource Managed Deduction. Cognitive Science Research Paper CSRP-99-17, University of Birmingham, 1999.

[4] C. Benzmüller and V. Sorge. A Blackboard Architecture for Guiding Interactive Proofs. *Proc. of AIMSA'98*, LNAI 1480, Springer, 1998.

[5] C. Benzmüller and V. Sorge. Critical Agents Supporting Interactive Theorem Proving. *Proc. of EPIA-99*, LNAI 1695, Springer, 1999.

[6] P. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. *Fundamenta Informaticae*, 39(1–2), 1999.

[7] J. Byrnes. *Proof Search and Normal Forms in Natural Deduction*. PhD thesis, Dep. of Philosophy, CMU, Pittsburgh, PA, USA, 1999.

[8] L. Cheikhrouhou and V. Sorge. *PDS* — A Three-Dimensional Data Structure for Proof Plans. In *Proc. of ACIDCA'2000*, Monastir, Tunisia, 2000.

[9] G. Dowek, Th. Hardin, and C. Kirchner. Theorem proving modulo. Rapport de Recherche 3400, INRIA, France, 1998.

[10] M. Fisher and M. Wooldridge A Logical Approach to the Representation of Societies of Agents. In N. Gilbert and R. Conte, editors, Artificial Societies. UCL Press, 1995.

[11] F. Giunchiglia, P. Bertoli, and A. Coglio. The OMRS project: State of the Art. *Electronic Notes in Theoretical Computer Science*, 15, 1998.

[12] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories - Towards an Architecture for Open Mechanized Reasoning Systems. In *Proc. of Frontiers of Combining Systems*, pages 157–174, 1996.

[13] M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, Cambridge, United Kingdom, 1993.

[14] A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. Journal of Automated Reasoning, 16(1–2):79–111, 1996.

[15] M. Jamnik, M. Kerber, and C. Benzmüller. Towards Learning new Proof Methods in Proof Planning. In this volume.

[16] A. Meier. Übersetzung automatisch erzeugter Beweise auf Faktenebene. Master's thesis, Computer Science Department, Universität des Saarlandes, Germany, 1997.

[17] The ΩMEGA-group. ΩMega: Towards a Mathematical Assistant. *Proc. of CADE–14*, LNAI 1249, Springer, 1997.

[18] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In *Computer-Aided Verification, CAV '96*, LNCS 1102, pages 411–414. Springer, 1996.

[19] V. Sorge. Non-Trivial Computations in Proof Planning. In *Proc. of Frontiers of Combining Systems*, LNCS 1794. Springer, 2000.

Journal of Universal Computer Science, vol. 5, no. 3 (1999), 188-207
submitted: 1/11/98, accepted: 15/3/99, appeared: 28/3/99

Integrating TPS and ΩMEGA

Christoph Benzmüller
Fachbereich Informatik, Universität des Saarlandes, Germany
chris@cs.uni-sb.de

Matthew Bishop[1]
Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, USA
mbishop+@cs.cmu.edu

Volker Sorge
Fachbereich Informatik, Universität des Saarlandes, Germany
sorge@ags.uni-sb.de

Abstract: This paper reports on the integration of the higher-order theorem proving environment TPS [Andrews *et al.*, 1996] into the mathematical assistant ΩMEGA [Benzmüller *et al.*, 1997]. TPS can be called from ΩMEGA either as a black box or as an interactive system. In black box mode, the user has control over the parameters which control proof search in TPS; in interactive mode, all features of the TPS-system are available to the user. If the subproblem which is passed to TPS contains concepts defined in ΩMEGA's database of mathematical theories, these definitions are not instantiated but are also passed to TPS. Using a special theory which contains proof tactics that model the ND-calculus variant of TPS within ΩMEGA, any complete or partial proof generated in TPS can be translated one to one into an ΩMEGA proof plan. Proof transformation is realised by proof plan expansion in ΩMEGA's 3-dimensional proof data structure, and remains transparent to the user.

1 Introduction

Current theorem proving systems, whether automatic or interactive, are usually strong in some domains while lacking reasoning power in others. Furthermore, there are no standardised formats for databases of higher-order problems, as there are for first-order problems [Sutcliffe *et al.*, 1994], and so higher-order theorem provers are generally unable to share databases of problems. In recent years there have been several attempts to combine two or more systems and hence to allow various theorem provers with different proof strategies to cooperate on a problem [Giunchiglia *et al.*, 1996], to allow users of an interactive system to invoke an external automatic system on a subproblem [Slind *et al.*, 1998a; Slind *et al.*, 1998b; Meier, 1997; Dahn *et al.*, 1994] or to avoid duplication of work by sharing databases [Felty and Howe, 1997].

In this paper we describe the integration of the higher-order theorem proving system TPS into the mathematical assistant ΩMEGA, and discuss the benefits that this provides for both systems. For a preliminary report on our work we refer to [Benzmüller and Sorge, 1998b].

[1] Supported by the National Science Foundation under grant CCR-9624683.

1.1 The TPS system

TPS [Andrews *et al.*, 1996] is a higher-order theorem proving system for classical type theory (Church's simply-typed λ-calculus). Proofs in TPS may be constructed automatically using the matings method (connection method) [Andrews, 1981], or interactively using an extended variant of Gentzen's natural deduction calculus [Gentzen, 1935]. Automatic proofs may be translated into natural deduction format [Miller, 1984; Pfenning, 1987], and hence the user may interleave the automatic and interactive proof methods by, for example, invoking the automatic component on a subproblem of a partially-completed interactive proof. This translation between automatic and natural deduction proofs provides the basis for the integration of TPS and ΩMEGA.

There are several built-in automatic search procedures in TPS, each of which is governed by a set of parameters (known as *flags*) which may be adjusted by the user or even automatically by TPS itself. Furthermore TPS can expand definitions using the dual instantiation strategy described in [Bishop and Andrews, 1998]; this provides an effective way to decide which abbreviations to instantiate during a proof. TPS provides a library for storing objects such as theorems, definitions and modes (groups of flag settings), and can also store and retrieve files containing sequences of commands (*work files*) or natural deduction proofs (*proof files*). All of these facilities are also used in the integration of ΩMEGA and TPS.

A more complete description of the capabilities of TPS is provided in [Andrews *et al.*, 1997], or online at `http://www.cs.cmu.edu/~andrews/tps.html`.

1.2 The ΩMEGA system

The ΩMEGA-system [Benzmüller *et al.*, 1997] is designed as an interactive mathematical assistant system, aimed at supporting proof development in mainstream mathematics. It consists of a variety of tools including a proof planner [Huang *et al.*, 1994], a graphical user interface LΩUI [Siekmann *et al.*, 1998], the PROVERB system [Huang and Fiedler, 1997] for translating proofs into natural language, and a variety of external systems such as computer algebra systems [Kerber *et al.*, 1998], automated theorem provers [McCune, 1994; Baumgartner and Furbach, 1994; Weidenbach *et al.*, 1996] and constraint solvers. ΩMEGA also provides the built-in higher-order theorem prover LEO [Benzmüller and Kohlhase, 1998], which specialises in reasoning about higher-order equality and extensionality.

ΩMEGA is, like TPS, a theorem proving system for classical type theory (Church's simply-typed λ-calculus) which uses a ND calculus variant as its basic inference mechanism. However the set of basic ND rules in TPS is larger than that in ΩMEGA, in order to keep TPS proofs concise and readable. Therefore certain rules in TPS abstract over small subproofs (such as *RuleP*, which abstracts over proofs in propositional logic, cf. Section 3). In ΩMEGA, however, the set of basic ND-rules is just large enough to ensure completeness, and all extensions to the basic ND-calculus (e.g. equality substitution) are defined as tactics. Nevertheless, proofs can be both constructed and displayed on several abstract levels by using a 3-dimensional data structure (see Section 2) for representing (partial) proofs. The structure on the one hand enables the user to freely switch back and forth between different abstract levels and on the other hand provides a means

for directly integrating results of external reasoners while leaving the expansion to the calculus level to ΩMEGA's tactic mechanism.

Further information and an online version of ΩMEGA are available over the Internet at `http://www.ags.uni-sb.de/~omega/`.

1.3 Benefits of integrating TPS and ΩMEGA

Both TPS and ΩMEGA use a higher-order logic based on Church's simply-typed λ-calculus, and both use a Gentzen-style natural deduction calculus; this makes the integration somewhat easier and more natural than it might otherwise have been. However, the two systems are still different enough for each to benefit considerably from the other.

ΩMEGA is designed to be a mathematical assistant, and so contains a small basic set of natural deduction rules, plus many defined tactics. ΩMEGA provides facilities such as a database of mathematical theories, a proof planner, proof verbalisation, integration of computer algebra systems and first-order theorem provers, and a graphical display in which the level of detail provided may be varied by the user. Since many of the predefined theories contain higher-order concepts, problems formulated in these theories will naturally lie beyond the capabilities of the first-order theorem provers which have already been integrated into ΩMEGA, and so the principal benefit of the integration for ΩMEGA is the addition of a powerful higher-order automated theorem prover as an external reasoning component.

TPS, on the other hand, is designed to be a system for proving theorems in a specific logic (as well as a tool for research into automated theorem proving). TPS must keep its proofs as concise as possible, since it has a command-line interface rather than the graphical interface of ΩMEGA, and so it contains a larger range of natural deduction rules than ΩMEGA. TPS has comparatively few predefined theories, since all but the smallest such theories contain far too many axioms for any of its automatic search procedures to cope with. Furthermore, TPS cannot invoke any external reasoning components. For TPS, then, the principal benefits of integration with ΩMEGA are the addition of a graphical interface, proof verbalisation, and the ability to use external reasoning systems (although the present integration does not allow TPS to call such systems itself, it can in effect call them through ΩMEGA, since ΩMEGA can call both TPS and the other systems, and any proof known to ΩMEGA can be passed to TPS).

2 Natural Deduction Proofs in ΩMEGA

The essential prerequisite for a smooth integration of TPS proofs into ΩMEGA proofs is ΩMEGA's ability to expand abstract inference steps into inferences in its own calculus. This enables the definition of abstract inference methods that can incorporate both decision procedures and partial proofs from other systems. In this section we will elaborate further on this issue by giving an overview of the core of the ΩMEGA system.

The entire process of theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, plan execution, and verification, centred around the so-called *Proof Plan Data Structure* ($\mathcal{PDS}$). A $\mathcal{PDS}$ is a hierarchical data structure which represents a (partial) proof at different levels of abstraction

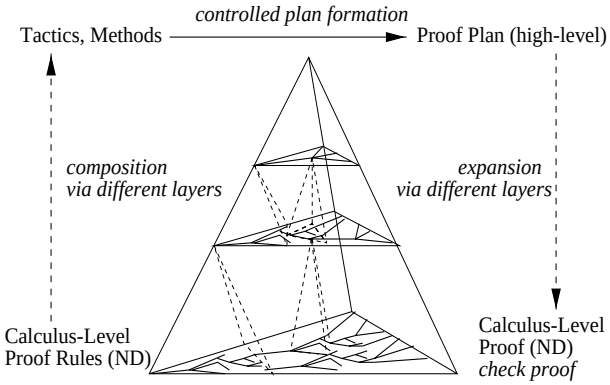


Figure 1: ΩMEGA's 3-dim. $\mathcal{PDS}$

(called *proof plans*). It is represented as a directed acyclic graph, where the nodes are justified by tactics or methods. Conceptually, each justification represents a proof plan (the *expansion* of the justification) at a lower level of abstraction that is computed when the justification is expanded. A proof plan can be recursively expanded until a fully explicit proof on the calculus level (ND) has been reached. In ΩMEGA, the original proof plan is kept in a 3-dimensional expansion hierarchy (cf. Figure 1). Thus the $\mathcal{PDS}$ makes explicit the hierarchical structure of proof plans and retains it for further applications such as proof explanation or analogical transfer of plans.

Once a proof plan is completed, its justifications can successively be expanded to verify the well-formedness of the resulting $\mathcal{PDS}$. When the expansion process is completed, the establishment of correctness of the ND proof relies solely on the correctness of the verifier and the calculus. This approach also provides a basis for a controlled integration of external reasoning components – such as an automated theorem prover or a computer algebra system – if each reasoner's results can (on demand) be transformed into a sub-$\mathcal{PDS}$.

A $\mathcal{PDS}$ can be constructed by automated or mixed-initiative planning, or by pure user interaction. In particular, new pieces of the $\mathcal{PDS}$ can be added by directly calling tactics, by inserting facts from a data base, or by calling some external reasoner.

In order to demonstrate the basic expansion mechanism we consider the ND-rule $\forall_E$ and the simple tactic $\forall_E^*$:

$$\frac{\forall x.A}{[t/x]A}\ \forall_E(t) \qquad \frac{\forall x_1,\ldots,x_n.A}{[t_1/x_1,\ldots,t_n/x_n]A}\ \forall_E^*(t_1,\ldots,t_n)$$

The application of the latter would be on an abstract level in the $\mathcal{PDS}$ and its expansion to ND-level would result in a sequence of applications of the $\forall_E$-rule. Besides providing a means for handling the application and expansion of these rather small abstractions, the $\mathcal{PDS}$ is also the foundation for integrating deductions from external reasoning components into ΩMEGA on a very abstract

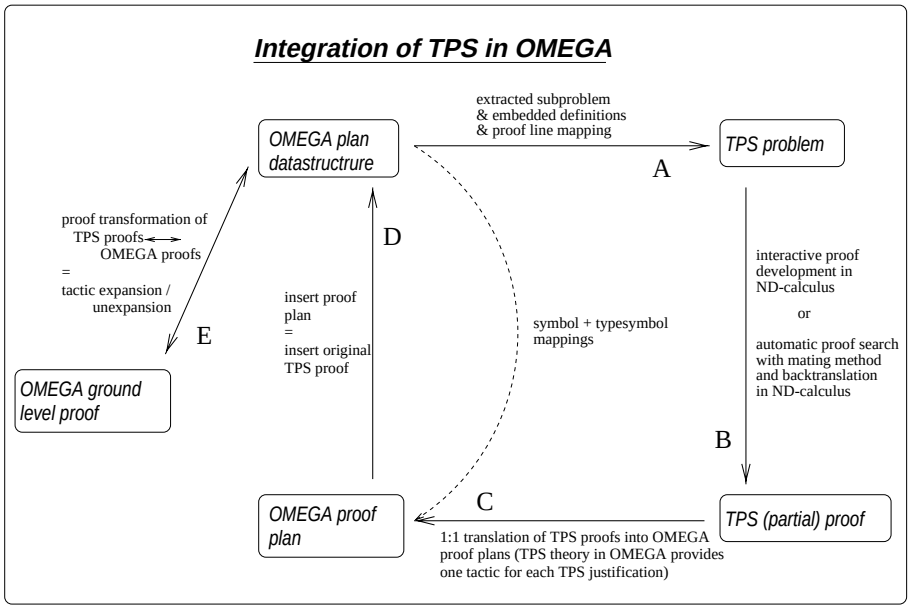


Figure 2: The integration architecture

level. We exploit this possibility for the integration of TPS by specifying three different abstraction levels for TPS's deductions:

1. A single justification expressing that a proof for a particular subproblem has been found by TPS.
2. A second expansion level incorporates the original TPS proof into ΩMEGA's $\mathcal{PDS}$. On this level the justifications for the respective proof lines contain the justifications of the original TPS proof.
3. A third level where the TPS justifications are mapped to corresponding ΩMEGA tactics. However, this level does not correspond to a proof on the calculus level, as some of the tactics might need to be expanded even further.

3 The Integration

The general integration approach, as illustrated in Figure 2, is divided into five steps A–E. Currently the integration is still one-directional; TPS can be used from within ΩMEGA, but ΩMEGA cannot be used from within TPS. We start with a partial proof plan, on an arbitrary abstraction level in ΩMEGA, that contains an open subproblem we want to prove with TPS. In step A the focused subproblem is extracted and, together with the relevant concepts from ΩMEGA's knowledge base, translated into TPS syntax. In step B TPS reads the translated problem and either tries to find a proof automatically (when called in automatic mode from ΩMEGA) or pops up its command interface for interactive proof development (when called in interactive mode from ΩMEGA; see the screen-shot in Example 3

in the appendix). The result is a complete or partial proof that is mirrored one-to-one as an ΩMEGA proof plan in step C. In step D, this proof plan is inserted into ΩMEGA's proof data structure ($\mathcal{PDS}$) in order to fill the given gap. Finally, a TPS proof which has been modelled on ΩMEGA's proof tactic level is transformed into a proper proof in ΩMEGA's basic ND-calculus by proof plan expansion in step E. This transformation may require support from the other external reasoners already integrated into ΩMEGA. Since all of the particular expansion steps in the proof transformation are stored in ΩMEGA's 3-dimensional $\mathcal{PDS}$, ΩMEGA's expansion/contraction mechanism for proof tactics allows the user to move freely between the TPS proof on an abstract level, the proof on ΩMEGA's basic ND-calculus level, and all of the intermediate levels of abstraction. This ensures that proof transformation is transparent to the user, and remains so even as the user examines the proof on different levels of abstraction.

In the following we will discuss the particular integration steps in more detail, using the following example as an illustration:

Example 1. (THM136) $\forall r_{o\alpha\alpha}.\mathit{transitive}\ (\mathit{transitive\text{-}closure}\ r)$
This example states that the transitive closure of a relation is transitive[2].

This problem is defined within ΩMEGA's theory *RELATION*, which also provides the recursively entailed defined concepts which are *transitive-closure*, *transitive* and *sub-relation*. These are defined as follows:
$\mathit{transitive\text{-}closure} := \lambda r_{o\alpha\alpha}.\lambda x_\alpha.\lambda y_\alpha.\forall q_{o\alpha\alpha}.$
$(\mathit{sub\text{-}relation}\ r\ q \wedge \mathit{transitive}\ q) \Rightarrow q\ x\ y$
$\mathit{transitive} := \lambda r_{o\alpha\alpha}.\forall x_\alpha, y_\alpha, z_\alpha.(r\ x\ y \wedge r\ y\ z) \Rightarrow r\ x\ z$
$\mathit{sub\text{-}relation} := \lambda r_{o\alpha\alpha}.\lambda q_{o\alpha\alpha}.\forall x_\alpha, y_\alpha.r\ x\ y \Rightarrow q\ x\ y$

3.1 A: Calling TPS from ΩMEGA

When calling TPS within ΩMEGA the user specifies the subgoal to be proved, some parameters which specify the proof heuristic to be used by TPS, and a time limit for this proof attempt. Furthermore the user may specify definitions that are entailed in the problem but which are not to be passed to TPS, in order to force TPS to treat them as uninterpreted constants.

Firstly, the focused subproblem is extracted from ΩMEGA's $\mathcal{PDS}$, by identifying the open subgoal explicitly mentioned as a parameter and determining its support nodes. Then ΩMEGA computes the set of all defined concepts that are recursively entailed in the extracted subproblem, and eliminates from this set all those concepts which the user has explicitly prohibited from being passed to TPS. Thus for THM136 we get exactly the three definitions shown above, assuming that the user has permitted all definitions to be passed to TPS. In the next step both this subproblem and the selected definitions are translated into TPS syntax. As both systems implement a logic based upon Church's simply-typed λ-calculus, and even their representations of types are very similar, this translation process is rather trivial, and we shall not discuss it in much detail. However, there are some minor considerations to be taken care of:

[2] Information on the syntax: In TPS the type $(\alpha \to \beta) \to \gamma$ is denoted $(\gamma(\beta\alpha))$. In particular, the type $o\alpha\alpha$ (i.e. $((o\alpha)\alpha)$) is the type $\alpha \to \alpha \to o$ of a binary relation on objects of type α.

1. TPS uses a small set of constant symbols with a fixed semantics (e.g. the logical connectives), and these symbols must not be redefined.
2. The polymorphic types which are allowed in ΩMEGA must usually be renamed in order for TPS to interpret them correctly.
3. It is important to maintain a mapping between the initial TPS proof lines in the translated subproblem and their counterparts in ΩMEGA's $\mathcal{PDS}$.

Problems 1 and 2 are solved by setting up hash-tables within ΩMEGA which store the necessary information about renamings of constant symbols (in 1) and the correspondence between the polymorphic type-symbols (in 2). As the line numbering in TPS steadily changes, we can not use another hash-table for solving 3. Fortunately, TPS allows the user to attach arbitrary additional information to each proof line; we use this feature to mark the TPS proof lines in the translated subproblem with the names of their counterparts on the ΩMEGA side.

Apart from the above-mentioned hash-tables, the most important results of phase A are two files containing all the necessary information for TPS. The first file — which we call the *problem-file* — contains the information on the subproblem in focus and the recursively embedded defined concepts. The second file — the *command-file* — contains a sequence of commands to be executed by TPS. These commands tell TPS to read the problem-file, to set the proof tactic as specified by the user and, in the case that TPS is called in automatic mode (see phase B), to invoke TPS's mating-search procedure. The problem-file created by ΩMEGA for our example THM136 is presented in the appendix of this paper (see Example 2).

3.2 B: Automatic or Interactive Proof Search in TPS

TPS can be called from ΩMEGA in either automatic or interactive mode. In the former case the TPS core image is started as a black box and the only information visible to the user is the time resource allocated to TPS's proof attempt. TPS executes only the commands which are specified in the command-file created by ΩMEGA.

When TPS is called in interactive mode, an xterm with TPS's command user interface pops up (see Example 3 in the appendix) and the interactive session is initialised by the commands stored in the command-file. The user can then interactively use all the available features of TPS in order to construct a complete or partial ND-style proof.

TPS's built-in proof transformation procedure [Miller, 1984; Pfenning, 1987] translates mating proofs into ND-calculus such that, in both interactive and automatic modes, the final result of the proof attempt is either a complete or partial proof in TPS's ND-calculus variant. This (partial) proof is then stored in a *tps-output* file[3] and passed back to ΩMEGA.

A very important feature of our approach is that TPS can use its mechanism for dual instantiation [Bishop and Andrews, 1998] within its mating-search procedure. This is possible because we do not expand all defined concepts before

[3] Actually there are two files produced by TPS, one containing the (partial) proof in ASCII format and one containing the same proof in a Lisp-like presentation. The former is only used to present the original TPS proof within ΩMEGA and the latter, which is the more important of the two, is used in phase C to translate the TPS proof to ΩMEGA.

passing the subproblem to TPS, but instead pass these concepts as additional information and leave the subproblem as it is. Thus TPS can decide on its own whether it is necessary to expand particular defined concepts or not. Example 1, above, is a good example of a theorem which cannot be proven by TPS if all the definitions are expanded before the mating-search procedure is called[4]. For a detailed discussion see [Bishop and Andrews, 1998]. The proof generated by TPS for THM136 is presented in Example 3 in the appendix.

3.3 C & D: Representing TPS Proofs as ΩMEGA-Proof Plans and Insertion of Proof Plans

One main idea of our approach is to provide as transparent a translation mechanism as possible, by modelling TPS's ND-calculus variant on ΩMEGA's proof tactic level. We implement this modelling by defining a special theory *TPS* in ΩMEGA's knowledge base. For each possible TPS ND justification, the theory *TPS* introduces a corresponding ΩMEGA-tactic; the expansion contents of some of these tactics are presented in Example 5 in the appendix. There is one additional black box tactic *tps*, which will be used to provide the most abstract view of subproblems proven by TPS. The concrete proof translation proceeds as follows:

1. A proof generated by TPS is mirrored one to one as a proof plan in ΩMEGA by mapping the particular proof justifications in the TPS proof to the corresponding proof tactics provided by the special theory *TPS* in ΩMEGA's knowledge base. In order to guarantee a correct mapping of the entailed constants and type symbols, the translation process uses the hash-tables constructed by ΩMEGA in phase A. Furthermore, the correspondence between the proof lines of the focused ΩMEGA-subproblem and the corresponding TPS proof lines is given as explicit information in the TPS proof. The proof plan we obtain for THM136 is presented as Example 4 in the appendix.
2. The resulting proof plan is then stored in ΩMEGA with a reference to the subproblem on which TPS has been called. Some additional information is also stored, such as the original TPS proof in ASCII format, the proof parameters and some proof statistics.

In phase D the open line itself is first closed and justified by using the special black box tactic *tps*, thereby providing the most abstract view of the proof for our subproblem in focus. By expanding this special tactic the corresponding proof plan is inserted in ΩMEGA's $\mathcal{PDS}$, and the structure of the original TPS proof can be visualised in ΩMEGA's graphical user interface LΩUI [Siekmann *et al.*, 1998]. Example 6 in the appendix presents the proof structure of the original TPS proof for THM136 (see Examples 3 and 4), graphically visualised in LΩUI.

3.4 E: Transparent Proof Transformation by Proof Plan Expansion

It remains to transform the abstract proof plan representing the TPS proof into ΩMEGA's own basic ND-calculus variant. Such a proof transformation is

[4] This theorem is still a challenging problem for current ATP's. Apart from a proof constructed by ΩMEGA's proof planner using very special control information [Sehn, 1995], TPS is the only system known to the authors that can automatically find a proof.

necessary, as ΩMEGA's philosophy on integrated systems is not to trust any externally-produced proof until it can be transformed and proof checked on ΩMEGA's basic ND-calculus level. The transformation problem for TPS proofs has a very simple solution since the ND-calculus variants of both systems are very similar, and the other external reasoners already integrated to ΩMEGA (e.g. OTTER [McCune, 1994]) can fruitfully support the transformation in non-trivial cases.

Proof transformation is realised via tactic expansion. Each proof tactic defined in ΩMEGA's special *TPS* theory contains specific expansion information that maps any concrete application of this particular tactic onto a proof on a lower, more detailed proof level in ΩMEGA's $\mathcal{PDS}$. Thus, by stepwise tactic expansion, the original TPS proof mirrored in ΩMEGA can finally be transformed into ΩMEGA's basic ND-calculus level. A nice side effect of this approach is that the original TPS proof, the corresponding ΩMEGA ND-proof and all intermediate levels of the proof transformation process are permanently stored in ΩMEGA's $\mathcal{PDS}$. Consequently the flexible tactic expansion/contraction mechanism in ΩMEGA allows users to analyse the proof on whatever level interests them. Example 6 in the appendix presents two different layers in ΩMEGA's $\mathcal{PDS}$.

We distinguish four categories of expansion tactics defined in the *TPS* theory, as follows:

I *Simple mapping:* Many rules of the ND-calculus variant of TPS have direct counterparts in ΩMEGA. Examples are presented in Figure 3. Here tactic tps*ForallE is mapped to ΩMEGA's basic ND-calculus rule $\forall_E$ and the tactic tps*Conj is mapped to the tactic $\wedge_E$, which itself expands into the basic ND-calculus rules $\wedge_{E_l}$ and $\wedge_{E_r}$. The expansion content of the tactic tps*ForallE is presented in Example 5 in the appendix.

II *Case Distinction:* Some tactics of the *TPS* theory need case distinctions in their expansion mapping. For example, the tactic tps*Neg justifies applications of the push negation as well as the pull negation principle; see Figure 3. ΩMEGA provides the corresponding tactics Pushneg and Pullneg, and thus the expansion of tps*Neg simply analyses the situation and maps to either Pushneg or Pullneg, as appropriate. Both Pushneg and Pullneg are tactics that expand with case distinction mappings to a lower level in ΩMEGA's $\mathcal{PDS}$. By subsequent tactic expansion we finally get a medium-sized derivation in ΩMEGA's basic ND-calculus. The definition of the tactic tps*Neg is presented in Example 5 in the appendix.

III *Restructuring:* Existential quantification elimination in TPS (the particular rule in TPS is called RuleC) structures a proof slightly differently from the corresponding rule $\exists_E$ in ΩMEGA; see Figure 3. Consequently the expansion of the tactic tps*RuleC into rule $\exists_E$ requires some simple restructuring of the proof with respect to the dependencies between some proof lines.

IV *External Reasoners:* TPS abbreviates pure propositional logic derivations in a complex ND proof with a single-step justification, called RuleP, and hides the boring details from the user. Thus both RuleP and the ΩMEGA-tactic tps*RuleP mean that a particular proof line follows from some premise lines by propositional logic. We need a way to expand this rather general justification, with so little detailed information available, into a concrete derivation in ΩMEGA's basic ND-calculus. An extravagant solution would be to imple-

Cat.	TPS tactic in ΩMEGA	Expansion Mapping	ΩMEGA's ND-calculus
I	$\dfrac{\forall x.A}{[x \leftarrow a]A}$ tps*Foralle(a)	$\dfrac{\forall x.A}{[x \leftarrow a]A}\ \forall_E(a)$	$\dfrac{\forall x.A}{[x \leftarrow a]A}\ \forall_E(a)$
I	$\dfrac{A \wedge B}{A \quad B}$ tps*Conj	$\dfrac{A \wedge B}{A \quad B}\ \wedge_E$	$\dfrac{A \wedge B}{A}\ \wedge_{E_l}\ \dfrac{A \wedge B}{B}\ \wedge_{E_r}$
II	$\dfrac{\neg A}{A'}$ tps*Neg $\dfrac{A'}{\neg A}$ tps*Neg	$\dfrac{\neg A}{A'}$ Pushneg $\dfrac{A'}{\neg A}$ Pullneg	$\neg A$ derivation D1 A' A' derivation D2 $\neg A$
III	$\dfrac{\exists x.A}{[[x \leftarrow a]A]^1}$ tps*Choose(a) $\dfrac{B}{B}$ tps*RuleC^1	$\dfrac{\exists x.A \qquad [[x \leftarrow a]A]^1 \quad B}{B}\ \exists_E{}^1$	$\dfrac{\exists x.A \qquad [[x \leftarrow a]A]^1 \quad B}{B}\ \exists_E{}^1$
IV	$\dfrac{A}{A'}$ tps*RuleP	$\dfrac{A}{A'}$ call-PL-ATP	A derivation D3 A'

Figure 3: Transparent transformation of TPS proofs into ΩMEGA proofs, as realised by ΩMEGA's tactic expansion mechanism.

ment a propositional logic prover in ΩMEGA and to employ this prover in the expansion of tps*RuleP. Fortunately there are already several systems integrated to ΩMEGA, such as the first-order provers OTTER [McCune, 1994], SPASS [Weidenbach *et al.*, 1996] or PROTEIN [Baumgartner and Furbach, 1994], which can be used instead. In fact, TPS itself also provides a special propositional logic mode that can be used to construct detailed propositional logic proofs. Hence no additional implementation effort with respect to the expansion of tps*RuleP is necessary; we simply map tps*RuleP to a recursive call of an arbitrary system, already integrated to ΩMEGA, that is able to construct propositional logic derivations (see Figure 3). In the first implementation we used OTTER in connection with a special mapping from higher-order to propositional logic. We can also map tps*RuleP back to a call of TPS in propositional logic mode. Then, by expanding tps*RuleP, ΩMEGA's tactic mechanism automatically performs a recursive call to TPS. The definition of the tactic tps*RuleP is presented in Example 5 in the appendix.

4 Examples

Our integration approach does not restrict the set of examples that can be proved by TPS. If one introduces the necessary definitions in ΩMEGA's knowledge base then generally all the theorems provable by TPS alone should be provable by calling TPS from ΩMEGA as well. Among the TPS examples that have already been proven by calling TPS from ΩMEGA (where they can be fully expanded and proof checked) are[5]:

Cantor's theorem: $\forall g_{o\alpha}.g <_{card} (\mathcal{P}\ g)$
The cardinality of the powerset of a set g is greater than the cardinality of g.

THM15b: $\forall f_{\iota\iota}.(\exists g_{\iota\iota}.(iteratep{+}\ f\ g) \wedge (\exists x_\iota.(g\ x) = x \wedge (\forall z_\iota.(g\ z) = z \Rightarrow z = x)) \Rightarrow (\exists y_\iota.((f\ y) = y))$
This theorem is discussed in detail in [Andrews *et al.*, 1996]. It states that if some positive iterate of f has a unique fixed point, then f has a fixed point.

THM48: $\forall f_{\iota\iota}.\forall g_{\iota\iota}.(injectivep\ f) \wedge (injectivep\ g) \Rightarrow (injectivep\ (f \circ g))$
The composition of injective functions is injective.

THM134: $\forall z_\iota.\forall g_{\iota\iota}.(iteratep{+}\ (\lambda x_\iota.z)\ g) \Rightarrow (\forall x_\iota.(g\ x) = z)$
The only positive iterate of a constant function is that function.

THM135: $\forall f_{\iota\iota}.\forall g^1_{\iota\iota}.\forall g^2_{\iota\iota}.(iteratep\ f\ g^1) \wedge (iteratep\ f\ g^2) \Rightarrow (iteratep\ f\ (g^1 \circ g^2))$
The composition of two iterates of a function f is an iterate of f.

---

[5] These examples from the TPS library are also discussed in [Andrews *et al.*, 1996].The definitions occurring in the above examples are defined in ΩMEGA's knowledgebase (analogously to TPS's library) as follows:
$<_{card} := \lambda g_{o\alpha}.\lambda h_{o\beta}.\neg\exists f_{\beta\alpha}.(surjective\ g\ h\ f)$
$surjective := \lambda f_{o\alpha}.\lambda g_{o\beta}.\lambda h_{\beta\alpha}.\forall x_\beta.(gx) \Rightarrow (\exists y_\alpha.(fy) \wedge (x = (hy)))$
$\mathcal{P} := superset,\ superset := \lambda u_{o\alpha}.\lambda v_{o\alpha}.\forall x_\alpha(u\ x) \Rightarrow (v\ x)$
$iteratep := \lambda f_{\alpha\alpha}.\lambda g_{\alpha\alpha}.\forall p_{o\alpha\alpha}.(p\ (\lambda u_\alpha.u) \wedge (\forall j_{\alpha\alpha}.(p\ j) \Rightarrow (p\ (f \circ j)))) \Rightarrow (p\ g)$
$iteratep{+} := \lambda f_{\alpha\alpha}.\lambda g_{\alpha\alpha}.\forall p_{o\alpha\alpha}.(p\ f) \wedge (\forall j_{\alpha\alpha}.(p\ j) \Rightarrow (p\ (f \circ j))) \Rightarrow (p\ g)$
$injectivep := \lambda f_{\gamma\beta}.\forall x_\beta.\forall y_\beta.((f\ x) = (f\ y)) \Rightarrow (x = y)$
$\circ := \lambda f_{\gamma\delta}.\lambda g_{\beta\gamma}.\lambda x_\delta.g\ (f\ x)$

$$\begin{aligned}\textit{THM270: } \forall f_{\beta\alpha}.\forall g_{\gamma\alpha}.\forall h_{\gamma\beta}.\ & (\forall x_\alpha.h\ (f\ x) = g\ x) \wedge (\forall y_\beta.\exists x_\alpha.f\ x = y)\\ & \wedge\ (\forall x_\alpha.\forall y_\alpha.f\ (x *^1 y) = (f\ x) *^2 (f\ y))\\ & \wedge\ (\forall x_\alpha.\forall y_\alpha.g\ (x *^1 y) = (g\ x) *^3 (g\ y))\\ & \Rightarrow (\forall x_\beta.\forall y_\beta.h\ (x *^2 y) = (h\ x) *^3 (h\ y))\end{aligned}$$

If f is a surjective homomorphism, g is a homomorphism, and h is any function such that for all x, $h\ (f\ x) = g\ x$, then h is a homomorphism.

In the following we present two examples, which are not automatically provable in either TPS or ΩMEGA alone, and which motivate a cooperation between the two systems.[6]

$$\begin{aligned}\textit{THM262: }\quad & \forall p_{o(o\iota)}.\textit{partition } p\\ & \Rightarrow \exists q_{(o\iota\iota)}.\textit{equivalence-rel } q \wedge (\textit{equivalence-classes } q) = p\end{aligned}$$

This states that if p is a partition, then there is an equivalence relation q whose equivalence classes are exactly the elements of p. We now demonstrate how a partly interactive and partly automatic proof can be constructed, and show how the integration of TPS and ΩMEGA can help with this task.

Suppose that the user begins by providing the appropriate instantiation for q (namely $\lambda x_\iota.\lambda y_\iota.\exists s_{o\iota}.p\ s \wedge s\ x \wedge s\ y$).This reduces the problem to two subgoals: proving that this lambda-term defines an equivalence relation, and proving that the equivalence classes of this relation are exactly p. In both cases, we have the hypothesis that p is a partition. The former subgoal can be proven automatically by TPS in about 35 seconds. The latter subgoal is harder for TPS; however, by using the interactive tactics for extensionality and universal generalisation, the user can reduce it to $(\textit{equivalence-classes}\ (\lambda x_\iota.\lambda y_\iota.\exists s.p\ s \wedge s\ x \wedge s\ y)\ b_{o\iota}) \equiv p\ b$. This equivalence can in turn be reduced interactively to a pair of implications, of which one (the right-to-left direction) can be proven automatically by TPS in about 30 seconds. This leaves the left-to-right direction of the equivalence as the only remaining subgoal to be proven. The automatic procedures of TPS cannot produce a proof of this subgoal, due to the complexity of the equality reasoning which is required, and so a user constructing this proof from within TPS would have to complete the proof interactively. The proof of this subgoal is non-trivial, and requires a significant amount of work on the part of the user.

However, with the integrated system, the user can begin proving THM262 in ΩMEGA, exactly as above, calling TPS to complete two of the three subgoals (none of the other systems integrated to ΩMEGA is known to be able to complete either subproof). For the remaining subgoal, instead of laboriously constructing an interactive proof, the user now has the additional option of invoking one of the other automated provers which are integrated to ΩMEGA or to call ΩMEGA's proof planner. It is very likely that an improved version of ΩMEGA$'s$ own higher-order theorem prover LEO, which specialises in reasoning about equality and extensionality, will be able to find an automatic proof of this subgoal.[7]

---

[6] The definitions used in this examples are as follows:
$\textit{partition} := \lambda s_{o(o\iota)}.(\forall p_{o\iota}.s\ p \Rightarrow (\exists z_\iota p\ z)) \wedge (\forall x_\iota.\exists p.s\ p \wedge p\ x \wedge (\forall q_{o\iota}.s\ q \wedge q\ x \Rightarrow q = p))$
$\textit{equivalence-rel} := \lambda r_{o\iota\iota}.\textit{reflexive } r \wedge \textit{symmetric } r \wedge \textit{transitive } r$
$\textit{equivalence-classes} := \lambda r_{o\iota\iota}.\lambda s_{o\iota}.(\exists z_\iota.s\ z) \wedge (\forall x_\iota.s\ x \Rightarrow (\forall y_\iota.s\ y \equiv r\ x\ y))$
$\textit{reflexive} := \lambda r_{o\iota\iota}.\forall x_\iota.r\ x\ x$ $\quad$ $\textit{symmetric} := \lambda r_{o\iota\iota}.\forall x_\iota.\forall y_\iota.r\ x\ y \Rightarrow r\ y\ x$
$\textit{transitive} := \lambda r_{o\iota\iota}.\forall x_\iota.\forall y_\iota.\forall z_\iota.r\ x\ y \wedge r\ y\ z \Rightarrow r\ x\ z$ $\quad$ $\emptyset := \lambda X_\alpha.\bot$

[7] In principle LEO provides exactly the required extensionality treatment to solve this subgoal, but due to its prototypical implementation LEO can still handle only small search spaces; the search space defined by this problem is rather large because many free predicate variables are involved. A technically improved and heuristically better-

The following statement (which we admit is rather contrived) serves to illustrate some of the strengths and weaknesses of TPS and LEO, as it is only provable when both systems cooperate.[8]

$$(\exists \circ_{ooo} . \top \circ \top \wedge \neg(\bot \circ \bot) \wedge \neg(\bot \circ \top) \wedge \neg(\top \circ \bot)) \wedge (\forall m_{oo}. \top \in m \equiv (\top \Rightarrow \top) \in m)$$

The first conjunct claims the existence of the logical connective $\wedge$ which is specified by its truth table. In order to prove this statement primitive substitution[9] has to be employed, which is strongly supported in TPS but widely avoided in LEO. For the proof of the second statement, on the other hand, the unification of $\top \in m$ and $(\top \Rightarrow \top) \in m$ requires a recursive call to the higher-order theorem prover from within higher-order unification. This most general form of extensionality treatment is supported in LEO but not in TPS. Hence this conjunction is provable in the combined system with three straightforward interactions.

Both examples illustrate that the integrated system of TPS and ΩMEGA allows the user to complete some proofs in much fewer interactions than would be required by either system alone. In fact, the few interactions which are required are already supported by the suggestion mechanism in ΩMEGA [Benzmüller and Sorge, 1998a]. While this in itself is already a major benefit to the user, it also suggests that it should be possible to use the built-in proof planner of ΩMEGA to oversee the cooperation of the various external systems, and to produce proofs such as the one above without the necessity of user interaction.

5 Conclusion

Our objective was to integrate the two knowledge-based higher-order theorem proving environments TPS and ΩMEGA in a way that would be as transparent to the user as possible. We believe that the approach to integration described above, although designed specifically for these two systems, provides some generally interesting and elegant ideas.

Our work (see also [Benzmüller and Sorge, 1998b]) is closely related to, and was developed simultaneously with, the approach for integrating the proof planner CL^{A}M and the interactive theorem prover HOL [Slind *et al.*, 1998a; Slind *et al.*, 1998b]. Although we must admit that our work was simplified by the fact that ΩMEGA and TPS are much more similar than are HOL and CL^{A}M, we believe that our approach provides some additional features, e.g. the communication of definitions between the two systems, and a more transparent proof transformation process.

In conclusion, we now summarise some of the more interesting general properties of our integration method.

- The integration of ΩMEGA and TPS also includes the communication of system-specific knowledge defined in the systems' knowledge bases. TPS and

guided version of LEO, which is currently being re-implemented, will most likely be able to find the proof.

[8] Although this example looks rather trivial at first glance, to the knowledge of the authors it is currently not automatically provable by any system.

[9] The primitive substitution principle guesses instantiations for free predicate variables. In this case the prover has to guess the instantiation $\wedge$ for $\circ$ and then to verify the conditions specified by the truth table.

ΩMEGA, which are both based on classical higher-order logic, do not need to agree on common definitions, rules or other logical concepts (apart from the logical connectives which are in any case identical in both systems), as is necessary for the integration of, for example, CL^{A}M and HOL [Slind *et al.*, 1998a; Slind *et al.*, 1998b]. Instead, ΩMEGA need only communicate to TPS all of the potentially important definitions and concepts belonging to the the specific subproblem to be solved. Most importantly, ΩMEGA does not expand any definition in the focused subproblem, but leaves the decision as to whether this is useful or necessary to TPS, which can use its mechanism for selectively instantiating definitions [Bishop and Andrews, 1998]. The user may even actively prevent some defined concepts from being passed to TPS.

- TPS is not only integrated as a fully automated black box system, but can also be called as an interactive theorem prover. Thus ΩMEGA, with its hierarchically structured knowledge base, can be seen in the integrated system as a second user interface to the TPS system, with its own knowledge base. As an automated black box system, TPS can be called from ΩMEGA either alone or concurrently with other integrated theorem provers such as the first-order systems OTTER, SPASS and PROTEIN.
- The ΩMEGA system models the particular ND-calculus variant used by TPS by providing corresponding tactics in a special theory *TPS* which introduces one ΩMEGA tactic for each TPS justification. Hence any TPS proof can be translated one to one into a corresponding ΩMEGA proof plan using the tactics from theory *TPS*. As the structure of the resulting proof plans can be visualised graphically in ΩMEGA's graphical user interface LΩUI [Siekmann *et al.*, 1998] TPS thereby gains a visualisation tool and graphical interface for free.
- Proof transformation of TPS proofs (mirrored as proof plans in ΩMEGA) into proofs in ΩMEGA's basic ND-calculus is realised by tactic expansion. As ΩMEGA's 3-dimensional proof data structure ($\mathcal{PDS}$) permanently stores all different abstraction levels of a proof (the ΩMEGA basic ND-level proof at the bottom layer, the mirrored TPS proof at an abstract level, and all intermediate abstraction levels between those), proof transformation becomes and remains transparent to the user, who can freely move between different levels of abstraction in the proof.
- Non-trivial tactic expansions (such as the one for RuleP) are supported by other external reasoners that are already integrated to ΩMEGA, or even by TPS itself. This saves us from having to define and implement complicated tactic expansions from scratch. Indeed, this can serve as a general approach for a tactic-based proof transformation within a system like ΩMEGA that already provides other integrated systems: as soon as a particular expansion step seems overly complicated, one can recursively call other integrated systems that are suited to support this particular expansion step.
- The reuse of mirrored TPS proof plans within an analogy-based theorem proving approach [Melis and Carbonell, 1998] is supported by our integration, as these proof plans are explicitly stored and thus available in ΩMEGA's $\mathcal{PDS}$. They can also be stored in ΩMEGA's knowledge base.

We are currently investigating whether TPS, ΩMEGA's own higher-order theorem prover LEO (which is specialised in reasoning about extensionality) and the various first-order theorem provers which have been integrated with ΩMEGA can

fruitfully cooperate. We hope to use ΩMEGA's $\mathcal{PDS}$ as the central data structure for the necessary information exchange between the cooperating systems, and ΩMEGA's planning mechanism to guide the cooperation between them.

References

[Andrews *et al.*, 1996] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[Andrews *et al.*, 1997] P. B. Andrews, S. Issar, D. Nesmith, F. Pfenning, H. Xi, and M. Bishop. *TPS3 Facilities Guide for Programmers and Users*, 1997. 207+viii pp.

[Andrews, 1981] P. B. Andrews. Theorem Proving via General Matings. *Journal of the Association for Computing Machinery*, 28(2):193–214, 1981.

[Baumgartner and Furbach, 1994] P. Baumgartner and U. Furbach. PROTEIN: A PROver with a Theory Extension INterface. In Bundy [1994], pages 769–773.

[Benzmüller and Kohlhase, 1998] C. Benzmüller and M. Kohlhase. LEO, a higher-order theorem prover. In Kirchner and Kirchner [1998], pages 139–143.

[Benzmüller and Sorge, 1998a] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In F. Giunchiglia, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 1480 in LNAI, pages 102–114, Sozopol, Bulgaria, 1998. Springer Verlag.

[Benzmüller and Sorge, 1998b] C. Benzmüller and V. Sorge. Integrating TPS with ΩMEGA. In J. Grundy and M. Newey, editors, *Theorem Proving in Higher Order Logics: Emerging Trends*, Technical Report 98-08, Department of Computer Science, pages 1–19, Canberra, Australia, 1998. The Australian National University.

[Benzmüller *et al.*, 1997] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMega: Towards a Mathematical Assistant. In McCune [1997].

[Bishop and Andrews, 1998] M. Bishop and P. B. Andrews. Selectively Instantiating Definitions. In Kirchner and Kirchner [1998], pages 365–380.

[Bundy, 1994] A. Bundy, editor. *Proceedings of CADE–12*, volume 814 of *LNAI*. Springer, Berlin, Germany, 1994.

[Dahn *et al.*, 1994] B. I. Dahn, J. Gehne, T. Honigmann, L. Walther, and A. Wolf. Integrating Logical Functions with ILF. Technical Report 94-10, Institut für Mathematik, Humboldt Universität zu Berlin, Germany, 1994.

[Felty and Howe, 1997] A. P. Felty and D. J. Howe. Hybrid Interactive Theorem Proving Using Nuprl and HOL. In McCune [1997], pages 351–365.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[Giunchiglia *et al.*, 1996] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories – Towards an Architecture for Open Mechanized Reasoning Systems. In F. Baader and K. U. Schulz, editors, *Frontiers of combining systems*, volume 3 of *Applied logic series*, pages 157 – 174, Dordrecht, The Netherlands, 1996. Kluwer Academic Publishers.

[Huang and Fiedler, 1997] X. Huang and A. Fiedler. Proof Verbalization in *PROVERB*. In *Proceedings of the First International Workshop on Proof Transformation and Presentation*, pages 35–36, Schloss Dagstuhl, Germany, 1997.

[Huang *et al.*, 1994] X. Huang, M. Kerber, J. Richts, and A. Sehn. Planning Mathematical Proofs with Methods. *Journal of Information Processing and Cybernetics (formerly: EIK)*, 30(5/6):277–291, 1994.

[Kerber *et al.*, 1998] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[Kirchner and Kirchner, 1998] C. Kirchner and H. Kirchner, editors. *Proceedings of CADE–15*, volume 1421 of *LNAI*. Springer, Berlin, Germany, 1998.
[McCune, 1994] W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.
[McCune, 1997] W. McCune, editor. *Proceedings of CADE–14*, volume 1249 of *LNAI*. Springer, Berlin, Germany, 1997.
[Meier, 1997] A. Meier. Übersetzung automatisch erzeugter Beweise auf Faktenebene. Master's thesis, Computer Science Department, Universität des Saarlandes, Saarbrücken, Germany, 1997.
[Melis and Carbonell, 1998] E. Melis and J .G. Carbonell. An argument for derivational analogy. *Advances in Analogy and Research*, 1998.
[Miller, 1984] D. Miller. Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs. In R.E. Shostak, editor, *Proceedings of CADE–7*, volume 170 of *LNCS*, pages 375–303. Springer, Berlin, Germany, 1984.
[Pfenning, 1987] F. Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, Pittsburgh Pa., 1987.
[Sehn, 1995] A. Sehn. DECLAME – eine deklarative Sprache zur Repräsentation von Methoden. Master's thesis, Computer Science Department, Universität des Saarlandes, 1995.
[Siekmann *et al.*, 1998] J. Siekmann, S. M. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, M. Kohlhase, K. Konrad, E. Melis, A. Meier, and V. Sorge. LΩUI: A Distributed Graphical User Interface for the Omega Proof System. International Workshop on User Interfaces for Theorem Provers, 1998.
[Slind *et al.*, 1998a] K. Slind, M. Gordon, R. Boulton, and A. Bundy. An Interface between CLAM and HOL. In Kirchner and Kirchner [1998], pages 134–138.
[Slind *et al.*, 1998b] K. Slind, M. Gordon, R. Boulton, and A. Bundy. An Interface between CLAM and HOL. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume 1479 of *LNCS*, pages 87–104. Springer, Berlin, Germany, 1998.
[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP Problem Library. In Bundy [1994], pages 252–266.
[Weidenbach *et al.*, 1996] Ch. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER, version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of CADE–13*, volume 1104 of *LNAI*. Springer, Berlin, Germany, 1996.

In the appendix we illustrate the integration architecture by presenting some concrete information on the interaction between ΩMEGA and TPS when proving THM136 (see Example 1).

A: Translating from ΩMEGA to TPS

Example 2 Problem-file. This is the content of the *problem-file* for THM136 generated by ΩMEGA and passed to TPS. The line with keyword ASSERTION defines the theorem to be proved and the line with keyword LINES introduces the initial partial proof to be completed by TPS, which here consists only of one line. A reference to the corresponding open proof line in ΩMEGA (the entry "(OMEGA-LABEL THM136)") and some further information belonging to ΩMEGA can be found at the end of this proof line. Note that the defined concepts *transitive*, *transitive-closure* and *sub-relation* are not expanded in this initial partial proof; they are passed to TPS as defined abbreviations (in the three lines with keyword DEF-ABBREV).

```
(DEFSAVEDPROOF OMEGA-SUBPROBLEM-THM136 (1998 9 30)
 (ASSERTION
  "[FORALL R(OaA)[TRANSITIVE(O(OAA))[TRANSITIVE-CLOSURE(OAA(OAA))R(OAA)] ] ] ")
 (NEXT-PLAN-NO 2) (PLANS ((1)))
 (LINES
  (1 NIL "[FORALL R(OAA)[TRANSITIVE [TRANSITIVE-CLOSURE R(OAA)] ] ] "
```

```
  PLAN1 NIL NIL "((OMEGA-LABEL THM136) (OMEGA-JUSTIFICATION OPEN))"))
0
((DEF-ABBREV TRANSITIVE (TYPE "O(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE TRANSITIVE) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-50(OAA)
     [FORALL DC-51(A)
      [FORALL DC-52(A)
       [FORALL DC-53(A)
        [IMPLIES [AND [DC-50(OAA)DC-51(A)DC-52(A)] [DC-50(OAA)DC-52(A)DC-53(A)]]
                 [DC-50(OAA)DC-51(A)DC-53(A)]]]]]]")
  (MHELP
   "Definition of the predicate for transitivity. (transitive R) is true, iff Rxy and Ryz imply Rxz. "))
 (DEF-ABBREV SUB-RELATION (TYPE "O(OAA)(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE SUB-RELATION) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-54(OAA)
     [LAMBDA DC-55(OAA)
      [FORALL DC-56(A)
       [FORALL DC-57(A)[IMPLIES [DC-54(OAA)DC-56(A)DC-57(A)] [DC-55(OAA)DC-56(A)DC-57(A)]]]]]]")
  (MHELP
 "Definition of the predicate for sub-relations. (sub-relation R R') is true, iff Rxy implies R'xy. "))
 (DEF-ABBREV TRANSITIVE-CLOSURE (TYPE "OAA(OAA)") (TYPELIST ("A"))
  (PRINTNOTYPE T) (FACE TRANSITIVE-CLOSURE) (FO-SINGLE-SYMBOL T)
  (DEFN
   "[LAMBDA DC-58(OAA)
     [LAMBDA DC-59(A)
      [LAMBDA DC-60(A)
       [FORALL DC-61(OAA)
        [IMPLIES [AND [SUB-RELATION(O(OAA)(OAA))DC-58(OAA)DC-61(OAA)] [TRANSITIVE DC-61(OAA)]]
                 [DC-61(OAA)DC-59(A)DC-60(A)]]]]]]")
  (MHELP "Definition of the transitive closure as in TPS. ")))
(COMMENT "OMEGA proof (report problems to the OMEGA group)")
(LOCKED (1)))
```

B: Proof Construction in TPS

Example 3 TPS *Proof.* Figure 4 presents a screenshot of the TPS interface displaying the TPS proof for THM136. This proof is discussed in detail in [Bishop and Andrews, 1998].

C & D: Translating from TPS to ΩMEGA and Inserting the Proof Plan

Example 4 ΩMEGA *Proof Plan.* ΩMEGA's special theory *TPS* provides one proof tactic for each TPS justification. Thus the proof presented in Example 3 can be translated one to one into a proof plan using the proof tactics of this theory. Tactics defined in this special theory have the prefix "TPS". The structure of this proof plan can be graphically visualised in ΩMEGA's graphical user interface LΩUI, as presented in Example 6.

```
THM136 ()        ! (FORALL [R:(O BB BB)]                                    TPS*UGEN: (R) (L23)
                    (TRANSITIVE (TRANSITIVE-CLOSURE R)))
L23    ()        ! (TRANSITIVE (TRANSITIVE-CLOSURE R))                   TPS*EQUIVWFFS: (L22)
L22    ()        ! (FORALL [DC-51:BB,DC-52:BB,DC-53:BB]                TPS*UGEN: (DC-51) (L21)
                    (IMPLIES
                     (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                          (TRANSITIVE-CLOSURE R DC-52 DC-53))
                     (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L21    ()        ! (FORALL [DC-52:BB,DC-53:BB]                          TPS*UGEN: (DC-52) (L20)
                    (IMPLIES
                     (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                          (TRANSITIVE-CLOSURE R DC-52 DC-53))
                     (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L20    ()        ! (FORALL [DC-53:BB]                                   TPS*UGEN: (DC-53) (L19)
                    (IMPLIES
                     (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                          (TRANSITIVE-CLOSURE R DC-52 DC-53))
                     (TRANSITIVE-CLOSURE R DC-51 DC-53)))
L19    ()        ! (IMPLIES                                                  TPS*DEDUCT: (L18)
                    (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)
                         (TRANSITIVE-CLOSURE R DC-52 DC-53))
                    (TRANSITIVE-CLOSURE R DC-51 DC-53))
```

Figure 4: TPS-Xterm with the proof of THM136

```
L18    (L1)       ! (TRANSITIVE-CLOSURE R DC-51 DC-53)                    TPS*EQUIVWFFS: (L17)
L17    (L1)       ! (FORALL [DC-61:(O BB BB)]                            TPS*UGEN: (DC-61) (L16)
                     (IMPLIES
                      (AND (SUB-RELATION R DC-61)
                           (TRANSITIVE DC-61))
                      (DC-61 DC-51 DC-53)))
L16    (L1)       ! (IMPLIES                                                TPS*DEDUCT: (L15)
                     (AND (SUB-RELATION R DC-61)
                          (TRANSITIVE DC-61))
                     (DC-61 DC-51 DC-53))
L15    (L1 L6)    ! (DC-61 DC-51 DC-53)                              TPS*RULEP: (L7 L8 L12 L13 L14)
L14    (L1)       ! (IMPLIES                                             TPS*UI: (DC-61) (L4)
                     (AND (SUB-RELATION R DC-61)
                          (TRANSITIVE DC-61))
                     (DC-61 DC-52 DC-53))
L13    (L1)       ! (IMPLIES                                             TPS*UI: (DC-61) (L5)
                     (AND (SUB-RELATION R DC-61)
                          (TRANSITIVE DC-61))
                     (DC-61 DC-51 DC-52))
L12    (L6)       ! (IMPLIES                                             TPS*UI: (DC-53) (L11)
                     (AND (DC-61 DC-51 DC-52) (DC-61 DC-52 DC-53))
                     (DC-61 DC-51 DC-53))
L11    (L6)       ! (FORALL [DC-53^1:BB]                                 TPS*UI: (DC-52) (L10)
                     (IMPLIES
                      (AND (DC-61 DC-51 DC-52)
                           (DC-61 DC-52 DC-53^1))
                      (DC-61 DC-51 DC-53^1)))
L10    (L6)       ! (FORALL [DC-52^1:BB,DC-53^1:BB]                      TPS*UI: (DC-51) (L9)
                     (IMPLIES
                      (AND (DC-61 DC-51 DC-52^1)
                           (DC-61 DC-52^1 DC-53^1))
                      (DC-61 DC-51 DC-53^1)))
```

```
L9     (L6)      ! (FORALL [DC-51^1:BB,DC-52^1:BB,DC-53^1:BB]                    TPS*EQUIVWFFS: (L8)
                    (IMPLIES
                     (AND (DC-61 DC-51^1 DC-52^1)
                          (DC-61 DC-52^1 DC-53^1))
                     (DC-61 DC-51^1 DC-53^1)))
L8     (L6)      ! (TRANSITIVE DC-61)                                               TPS*RULEP: (L6)
L7     (L6)      ! (SUB-RELATION R DC-61)                                           TPS*RULEP: (L6)
L6     (L6)      ! (AND (SUB-RELATION R DC-61) (TRANSITIVE DC-61))                        TPS*HYP
L5     (L1)      ! (FORALL [DC-61^1:(O BB BB)]                                   TPS*EQUIVWFFS: (L2)
                    (IMPLIES
                     (AND (SUB-RELATION R DC-61^1)
                          (TRANSITIVE DC-61^1))
                     (DC-61^1 DC-51 DC-52)))
L4     (L1)      ! (FORALL [DC-61^1:(O BB BB)]                                   TPS*EQUIVWFFS: (L3)
                    (IMPLIES
                     (AND (SUB-RELATION R DC-61^1)
                          (TRANSITIVE DC-61^1))
                     (DC-61^1 DC-52 DC-53)))
L3     (L1)      ! (TRANSITIVE-CLOSURE R DC-52 DC-53)                               TPS*RULEP: (L1)
L2     (L1)      ! (TRANSITIVE-CLOSURE R DC-51 DC-52)                               TPS*RULEP: (L1)
L1     (L1)      ! (AND (TRANSITIVE-CLOSURE R DC-51 DC-52)                                TPS*HYP
                        (TRANSITIVE-CLOSURE R DC-52 DC-53))
```

E: Transparent Proof Transformation by Proof Plan Expansion

Example 5 Modelling TPS*'s calculus in* ΩMEGA*'s theory TPS .* The tactics in ΩMEGA's special theory TPS contain expansion information that allows proof plans constructed in this theory to be mapped to ΩMEGA proofs on a lower abstraction level. We present some sample expansions here. The simplest is tps*Conj, which is simply mapped to the ΩMEGA tactic ande. The expansion of tps*Neg first analyses the given situation and then maps either to Pushneg or Pullneg. tps*RuleP recursively invokes an external propositional logic prover integrated to ΩMEGA.

```
(defun tpstac=expand-tps*Conj (outline parameters)
  (tacl~init outline)
   (tacl~apply 'ande outline nil)
  (tacl~end))

(defun tpstac=expand-tps*Neg (outline parameters)
  (tacl~init outline)
  (cond ((tpstac=pushneg-a-p (node~formula (car outline)) (node~formula (cadr outline)))
         (tacl~apply 'pushneg outline nil))
        ((tpstac=pullneg-a-p (node~formula (cadr outline)) (node~formula (car outline)))
         (tacl~apply 'pullneg outline nil))
        (t (warn "Something went wrong while expanding justification tps*Neg")))
  (tacl~end))

(defun tpstac=expand-tps*RuleP (outline parameters)
  (declare (ignore parameters))
  (let* ((node (car outline))
         (premises (just~premises (node~justification node))))
    (tacl~init outline)
    (tpstac=call-external-atp node premises)
    (tacl~end)
    (setf (pdsj~status (node~justification node)) "untested")))
```

Example 6 ΩMEGA*-proof.* Finally, we present in figure 5 the visualization of the original TPS proof (as a proof plan) in ΩMEGA's graphical user interface LΩUI. By expanding all nodes exactly one step, we reach another layer in ΩMEGA's 3-dimensional $\mathcal{PDS}$ which is visualized in the second screenshot. Here the squares represent the recursive calls to a propositional theorem prover which are obtained by the expansion of tactic *tps*RuleP*.

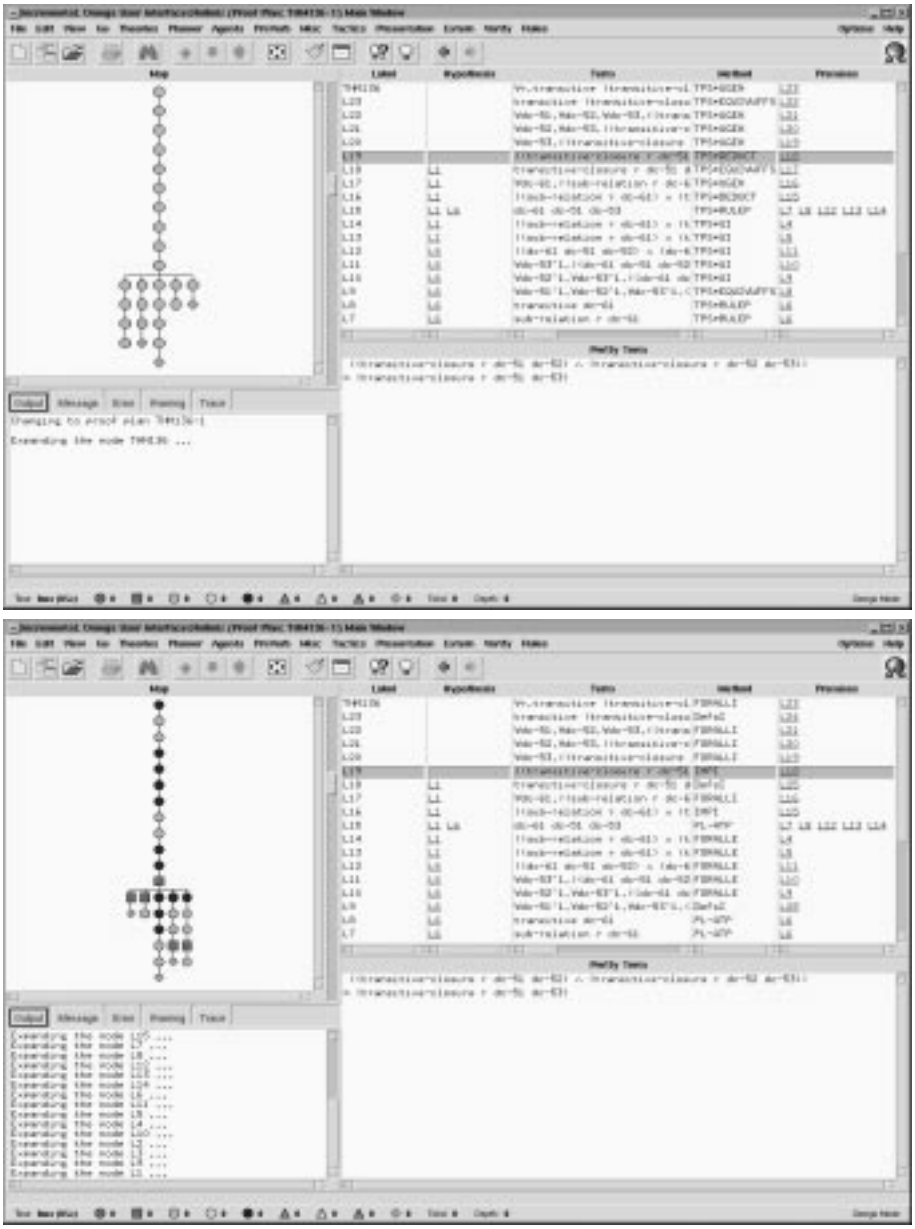


Figure 5: Transparent proof transformation within ΩMEGA's 3-dimensional $\mathcal{PDS}$.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88
JAL:m1a v 1.50 Prn:17/11/2005; 11:16
[DTD5] P.1 (1-27)
by:Jolanta p. 1

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Journal of Applied Logic ●●● (●●●●) ●●●–●●●

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Computer supported mathematics with ΩMEGA

Jörg Siekmann *, Christoph Benzmüller, Serge Autexier

Universität des Saarlandes and DFKI Gmblt, Saarbrücken, Germany

Abstract

Classical automated theorem proving of today is based on ingenious search techniques to find a proof for a given theorem in very large search spaces—often in the range of several billion clauses. But in spite of many successful attempts to prove even open mathematical problems automatically, their use in everyday mathematical practice is still limited.

The shift from search based methods to more abstract planning techniques however opened up a paradigm for mathematical reasoning on a computer and several systems of that kind now employ a mix of interactive, search based as well as proof planning techniques.

The ΩMEGA system is at the core of several related and well-integrated research projects of the ΩMEGA research group, whose aim is to develop system support for a working mathematician as well as a software engineer when employing formal methods for quality assurance. In particular, ΩMEGA supports proof development at a human-oriented abstract level of proof granularity. It is a modular system with a central proof data structure and several supplementary subsystems including automated deduction and computer algebra systems. ΩMEGA has many characteristics in common with systems like NUPRL, COQ, HOL, PVS, and ISABELLE. However, it differs from these systems with respect to its focus on *proof planning* and in that respect it is more similar to the proof planning systems CLAM and λCLAM at Edinburgh.

Keywords: Artificial intelligence; Computer-supported mathematics; Proof assistant systems; Interactive and automated theorem-proving; Proof planning; Omega

* Corresponding author.
E-mail addresses: siekmann@dfki.de (J. Siekmann), chris@ags.uni-sb.de (C. Benzmüller), autexier@ags.uni-sb.de (S. Autexier).
URLs: http://www-ags.dfki.uni-sb.de (J. Siekmann), http://www.ags.uni-sb.de/~chris (C. Benzmüller), http://www.ags.uni-sb.de/~autexier (S. Autexier).

1570-8683/$ – see front matter
doi:10.1016/j.jal.2005.10.008

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.2 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 2

1. Introduction

The vision of computer-supported mathematics and a system which provides integrated support for all work phases of a mathematician has always fascinated researchers in artificial intelligence, particularly in the deduction systems area, and more recently in mathematics as well. The dream of mechanizing (mathematical) reasoning dates back to Gottfried Wilhelm Leibniz in the 17th century with the touching vision that two philosophers engaged in a dispute would one day simply code their arguments into an appropriate formalism and then *calculate* (Calculemus!) who is right. At the end of the 19th century modern mathematical logic was born with Frege's Begriffsschrift and an important milestone in the formalization of mathematics was Hilbert's program and the 20th century Bourbakism.

With the logical formalism for the representation and calculation of mathematical arguments emerging in the first part of the twentieth century it was but a small step to implement these techniques now on a computer as soon as it was widely available.

In 1954 Martin Davis' Presburger Arithmetic Program was reported to the US Army Ordnance and the Dartmouth Conference in 1956, which is not only known for giving birth to artificial intelligence in general but also more specifically for the demonstration of the first automated reasoning programs for mathematics by Herb Simon and Alan Newell.

However, after the early enthusiasm of the 1960s, in particular the publication of the resolution principle in 1965 [84], and the developments in the 70s a more sober realization of the actual difficulties involved in automating everyday mathematics set in and the field increasingly fragmented into many subareas which all developed their specific techniques and systems.[1]

It is only very recently that this trend appears to be reversed, with the CALCULEMUS[2] and MKM[3] communities as driving forces of this movement. In CALCULEMUS the viewpoint is bottom-up, starting from existing techniques and tools developed in the computer-algebra and deduction systems communities. MKM[3] approaches the goal of computer-based mathematics for the new millennium by a complementary top-down approach starting from existing, mainly pen and paper based mathematical practice down to system support.

The ΩMEGA project aims at an integrated approach since its start in the mid 80s and it is deeply rooted in both initiatives. The ΩMEGA system is at the core of the project and it has many characteristics in common with systems like NUPRL [1], COQ [34], HOL [47], PVS [79], and ISABELLE [80,78]. However, it differs from these systems with respect to its focus on *proof planning* and in that respect it is more similar to the proof planning systems CLAM and λCLAM at Edinburgh [83,29]. In this article we shall first provide an overview of the main developments of the ΩMEGA project and then point to current research and some future goals.

[1] The history of the field is presented in a classical paper by Martin Davis [35] and also in [36] and more generally in his history of the first electronic computers [37]. Another source is Jörg Siekmann [86] and more recently [87].

[2] http://www.calculemus.org.

[3] http://www.mkm-ig.org.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.3 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 3

2. ΩMEGA

The ΩMEGA project represents one of the major attempts to build an all encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The inference mechanism at the lowest level of granularity is an interactive theorem prover based on a higher-order natural deduction (ND) variant of a soft-sorted version of Church's simply typed λ-calculus [33]. The logical language, which also provides some support for partial functions, is called $\mathcal{POST}$, for ***p***artial functions and ***o***rder ***s***orted ***t***ype theory. The search for a proof, however, is usually conducted at a higher level of granularity defined by *tactics* and *methods*. Automated proof search at this 'abstract' (i.e., less granular) level is called *proof planning* (see Section 2.3). Proof construction is also supported by already proven assertions, i.e., theorems and lemmata, and by calls to external systems to simplify or solve subproblems. Resource-guided search for applicable tactics, methods, and external systems is conducted by ΩANTS, an agent-based reasoning system.

2.1. System overview

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [32], in which *proofs* and *proof plans* are represented at various levels of granularity (see Fig. 1). The $\mathcal{PDS}$ is a di-

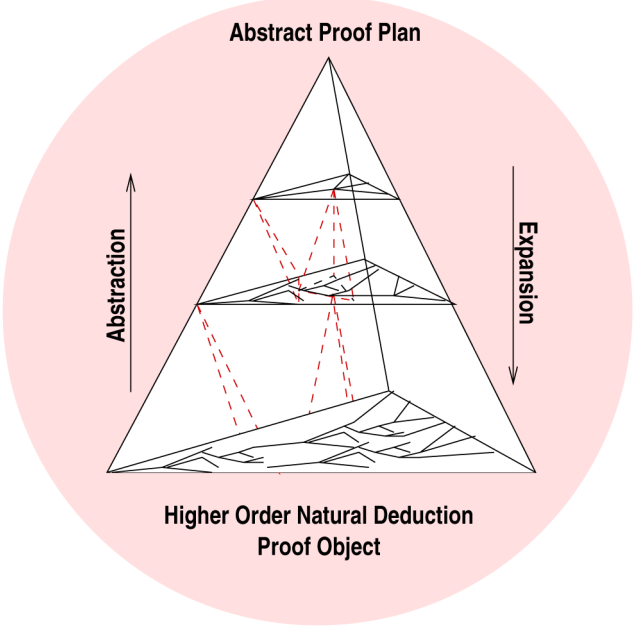


Fig. 1. The proof plan datastructure $\mathcal{PDS}$ is at the core of the ΩMEGA system.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.4 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 4

rected acyclic graph, where *open nodes* represent unjustified propositions that still need to be proved and *closed nodes* represent propositions that are already proved. The proof plans are developed and classified with respect to a taxonomy of mathematical theories in the mathematical knowledge base MBASE [42,56]. The user of ΩMEGA, or the proof planner MULTI [73,64], or else the agent-based reasoning system ΩANTS [19] modify the $\mathcal{PDS}$ during proof development until a complete proof plan, where all nodes are closed, has been found. They can also invoke external reasoning systems, whose results are included in the $\mathcal{PDS}$ after appropriate transformation. Once a complete proof plan at an appropriate level of granularity has been found, this plan must be expanded by sub-methods and sub-tactics into lower levels of granularity until finally a proof at the level of the logical calculus is established. After expansion of these high-level proofs to the underlying ND-calculus, the $\mathcal{PDS}$ can be checked by ΩMEGA's proof checker.

Hence, there are two main tasks supported by this system, namely (i) to find a proof plan, and (ii) to expand this proof plan into a calculus-level proof; and both jobs can be equally difficult and time consuming. Task (ii) employs a combination of an LCF-style tactic based expansion mechanism as well as deductive proof search in order to generate a lower-level proof object. It is a design objective of the $\mathcal{PDS}$ that various *proof levels* coexist with their respective dynamic relationships being maintained.

The graphical user interface $\mathcal{LOUI}$ [90] provides both a graphical and a tabular view of the proof under consideration, and the interactive proof explanation system *P.rex* [40, 39,41] generates a natural language presentation of the proof (see Figs. 5 and 6).

The previously monolithic system has been split up and separated into several independent modules (see Fig. 2), which are connected via the mathematical software bus MATHWEB-SB [99]. An important benefit is that MATHWEB-SB modules can be distributed over the Internet and are then remotely accessible by other research groups as well. There is now a very active MathWeb user community with sometimes several thousand theorems and lemmata being proven per day. Many theorems are generated automatically as (currently non-reusable and non-indexed) subproblems in natural language processing (see the Doris system[4]), proof planning and verification tasks.

2.2. Proof objects

The central data structure for the overall search is the proof plan data structure $\mathcal{PDS}$ in Fig. 1 and the subsystems cooperate to construct a proof whose status is stored again in the $\mathcal{PDS}$. The facilities provided by the subsystems include support for interactive and mixed-initiative theorem proving by the user, the proof planner, and by external systems such as automated theorem provers and computer algebra systems. These facilities require, in particular, the representation of proof steps at different levels of granularity ranging from abstract, human-oriented reasoning to logic-level justifications.

Therefore ΩMEGA provides a hierarchical proof plan data structure that represents a (partial) proof at different levels of granularity (called partial proof plans). Technically, the $\mathcal{PDS}$ is a directed acyclic graph consisting of nodes, justifications and hierarchical edges

[4] http://www.cogsci.ed.ac.uk/~jbos/doris/.

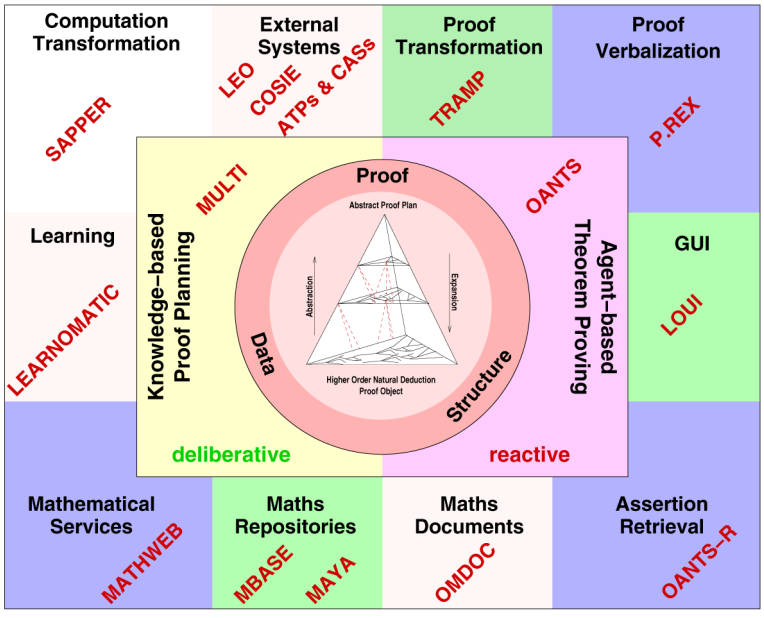


Fig. 2. The vision of an all encompassing mathematical assistance environment: we have now modularized and out-sourced many of the support tools (whose names are printed in red) such that they can also be used by other systems via the MATHWEB-SB software bus. (For interpretation of the references in color in this figure legend, the reader is referred to the web version of this article.)

(see [32] for more details). Each node represents a sequent and can be *open* or *closed*. An open node corresponds to a sequent that is to be proved and a closed node to a sequent which is already proved or reduced to other sequents using an inference rule $R := \frac{A_1 \ldots A_k}{B}$; where R may represent a calculus rule, a tactic, a method, or a call to an external system. Such a rule denotes that we can conclude B from $A_1, \ldots, A_k$ or reading it the other way round that B can be reduced to $A_1, \ldots A_k$. Thus, an inference step is represented by a justification R which connects a node n_b containing the sequent B to nodes $n_1, \ldots, n_k$ containing the sequents $A_1, \ldots A_k$. If a node has more than one outgoing justification, each of them represents a proof attempt of the sequent stored in the source node, but at different granularity. These justifications are ordered with respect to their granularity using hierarchical edges. A hierarchical edge connects two justifications j_1 and j_2 with the meaning that justification j_1 represents a more detailed proof attempt than justification j_2. Thus, ΩMEGA's $\mathcal{PDS}$ explicitly maintains the original proof plan as well as intermediate expansion layers in an expansion hierarchy.

Normally, the user wants to see the proof only at a specific level of granularity and therefore he can chose the granularity by selecting the justification for each node in the $\mathcal{PDS}$. Fig. 3 shows an example of how the selection of a justification of a node determines the level of granularity. It shows a node n with two outgoing justifications j_1 and j_2, which are connected by a hierarchical edge h from j_1 to j_2 indicating that j_1 is a more granular justification than j_2. The user can decide whether he wants to see the more detailed version of the proof given by j_1 (and its subtree t_1) or the more abstract version given by j_2 (and

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.6 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 6

Fig. 3. Representation of a $\mathcal{PDS}$ node with justifications at different levels of granularity.

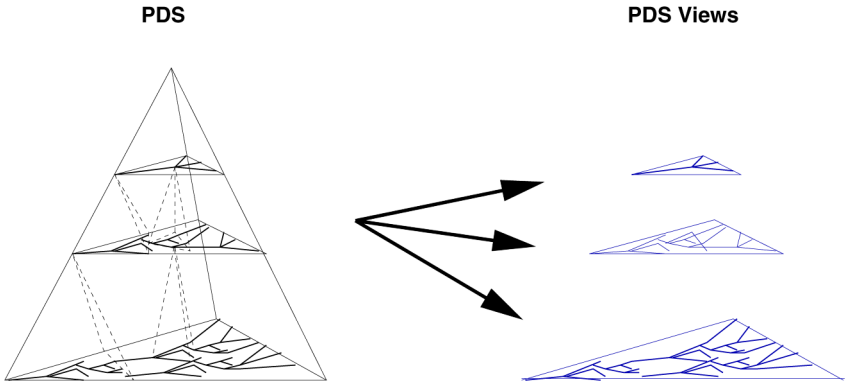


Fig. 4. Possible views of proofs at different levels of granularity inside a $\mathcal{PDS}$.

its subtree t_2). The two different possible selections are shaded. Selecting the justifications for each node the user gets a view into the $\mathcal{PDS}$-graph, called a $\mathcal{PDS}$-view (see Fig. 4), at the selected level of granularity.

Note that in contrast to the traditional LCF approach, it is not mandatory to immediately expand a high-level proof plan to a lower-level, because we explicitly represent the high-level proof plans in the $\mathcal{PDS}$ and thus conceptually separate plan formation from plan validation (by recursive expansion). Validation of proof plans can thus be postponed and executed at any time later on. In case of an unsuccessful expansion attempt, ΩMEGA's $\mathcal{PDS}$ provides mechanisms which change the status of the affected proof nodes from *justified*, i.e., *closed*, to *open* and then consistently clean up all structures, which depend on these nodes. Thus, failing expansion may in particular introduce new gaps into a previously closed proof plan and hence proof planning has to start again in order to fill the gaps and search for a new plan.

Because the $\mathcal{PDS}$ represents the dependencies among goals and subgoals as well as between high-level inference rules and lower-level inference rules, we can traverse the datastructure in many ways for different purposes like visualization, proof explanation, natural language generation and dependency-directed pruning of the proof object.

In summary, coexistence of several granularity levels and the dynamical maintenance of their relationship is a central and distinguishing design objective of ΩMEGA's $\mathcal{PDS}$. The $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof expansion, proof explanation with *P.rex* or an analogical transfer of plans.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.7 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 7

Currently, however, we cannot change the representation *inside* of a proof node, which is still something to be desired. For example, it would be nice to be able to change the logical propositions in naive set theory into Venn diagrams such that a diagrammatic reasoning system could be used. Support for representational shifts of this kind in combination with different levels of granularity is future work.

The proof object generated by ΩMEGA for the theorem "$\sqrt{2}$ is irrational", which has a well known human proof of less than a dozen lines, is recorded in a technical report [14], where the unexpanded and the expanded proof objects are presented in great detail: The most abstract proof at the level of the proof plan has about twenty steps and the fully expanded proof has about 750. The final proof in natural language generated by the ΩMEGA-system is shown in Fig. 6. A general presentation of this interesting case study is [88].

2.3. *Proof planning*

ΩMEGA's main focus is on knowledge-based proof planning [25,26,74], where proofs are not conceived in terms of low-level calculus rules, but at a less detailed granularity, i.e., a more abstract level, that highlights the main ideas and de-emphasizes minor logical or mathematical manipulations on formulae.

Knowledge-based proof planning is a paradigm in automated theorem proving, which swings its motivational pendulum back to the AI origins in that it employs and further develops many AI principles and techniques such as hierarchical planning, knowledge representation in frames and control rules, constraint solving, tactical theorem proving, and meta-level reasoning. It differs from traditional search based techniques in automated theorem proving not least in its level of granularity: The proof of a theorem is planned at an abstract-level where an outline of the proof is found first. This outline, that is, the abstract proof plan, can be recursively expanded to construct a proof within a logical calculus provided the expansion of the proof plan does not fail. The plan operators, called *methods*, represent mathematical techniques familiar to a working mathematician. While the knowledge of a mathematical domain as represented by methods and control rules is specific to the mathematical field, the representational techniques and reasoning procedures are general-purpose. For example, one of our first case studies [74] used the limit theorems proposed by Woody Bledsoe [23] as a challenge to automated reasoning systems. The general-purpose planner makes use of the mathematical domain knowledge of ϵ–δ-proofs and of the guidance provided by declaratively represented control rules, which correspond to mathematical intuition about how to prove a theorem in a particular situation. These rules are the basis for our meta-level reasoning and the goal-directed behavior.

Domain knowledge is encoded into methods, control rules, and strategies. Moreover, methods and control rules can employ external systems (e.g., one method is to call one of the computer algebra systems) and make use of the knowledge in these systems. ΩMEGA's multi-strategy proof planner MULTI [73,64] searches then for a plan using the acquired methods and strategies guided by the control knowledge in the control rules.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.8 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 8

2.3.1. AI principles in proof planning

A *planning problem* is a formal description of an *initial state*, a *goal*, and some *operators* that can be used to transform the initial state via some intermediate states to a state that satisfies the goal. Applied to a planning problem, a *planner* returns a sequence of *actions*, that is, instantiated operators (i.e., methods), which reach a goal state from the initial state when executed. Such a sequence of actions is called a *solution plan*.

Proof planning considers mathematical theorems as planning problems [25]. The initial state of a proof planning problem consists of the proof *assumptions* of the theorem, whereas the goal is the *theorem* itself. The operators in proof planning are the methods, traditionally they are tactics augmented by pre- and postconditions.

In ΩMEGA, proof planning is the process that computes actions, that is, instantiations of methods, and assembles them sequentially in order to derive a theorem from a set of assumptions. The effects and the preconditions of an action in proof planning are formulae in the higher-order language $\mathcal{POST}$, where the effects are considered as logically inferable from the preconditions using this method. A proof plan under construction is represented in the proof plan data structure $\mathcal{PDS}$, which consists initially of an open node containing the conjecture to be proven, and closed, i.e., justified nodes for the proof assumptions. The introduction of a method changes the $\mathcal{PDS}$ by adding new proof nodes and justifying the effects of the method by applications of the method to its premises. The aim of the proof planning process is to reach a *closed* $\mathcal{PDS}$, that is, a $\mathcal{PDS}$ without open nodes. The *solution proof plan* produced is then a record of the sequence of actions that lead to a closed $\mathcal{PDS}$.

By allowing for forward and backward methods, ΩMEGA's proof planner MULTI combines forward and backward state-space planning. Thus, a *planning state* in MULTI is a pair of the current world state and the current goal state. The initial world state consists of the given proof assumptions and is transferred by forward methods into a new world state. The goal state consists of the initial open node and is transferred by backward methods into a new goal state containing new open nodes. From this point of view the aim of proof planning is to compute a sequence of actions that derives a current world state in which all the goals are satisfied.

As opposed to precondition achievement planning (e.g., see [96]), effects of methods in proof planning do not cancel each other. For instance, a method with effect $\neg F$ introduced for an open node L_1 does not threaten the effect F introduced by another method for an open node L_2. Dependencies among open nodes result from shared variables for witness terms and their constraints. Constraints can, for instance, be instantiations for the variables but they can also be mathematical constraints such as $x < c$, which states that, whatever the instantiation for x is, it has to be smaller than c. The constraints created during the proof planning process are collected in the constraint store of the $\mathcal{CoSIE}$ system [76, 100], which is a domain-independent extension of existing propagation-based constraint solvers. The extension turned out to be necessary, since proof planning has peculiar requirements that are not met by off-the-shelf constraint solvers: $\mathcal{CoSIE}$ computes symbolic constraint inferences while respecting the logical side-conditions of proof planning, for instance, the Eigenvariable condition and the logical dependencies between constraints and their context. The search procedure of $\mathcal{CoSIE}$ computes logically correct instantiations for the meta-variables.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.9 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 9

A proof-planning method is applicable only if its constraints are consistent with the constraints collected so far. Dependencies among goals with shared variables are difficult to analyze and can cause various kinds of failures in a proof planning attempt (see [63] for more details).

2.3.2. Methods, control rules, and strategies

Methods are traditionally perceived as tactics in tactical theorem proving [78] augmented with preconditions and effects, called *premises* and *conclusions*, respectively. A method represents a large inference of the conclusion from the premises based on the body of the tactic. For instance, NotI-m is a (very low-level) method whose purpose is to prove a goal $\Gamma \vdash \neg P$ by contradiction. If NotI-m is applied to a goal $\Gamma \vdash \neg P$ then it closes this goal and introduces the new goal to prove falsity, $\bot$, under the assumption P, that is, $\Gamma, P \vdash \bot$. Thereby, $\Gamma \vdash \neg P$ is the conclusion of the method, whereas $\Gamma, P \vdash \bot$ is the premise of the method. NotI-m is a *backward* method, which reduces a goal (the conclusion) to new goals (the premises). *Forward* methods, in contrast, derive new conclusions from given premises. For instance, =Subst-m performs equality substitutions, for example, by deriving from the two premises $\Gamma \vdash P[a]$ and $\Gamma \vdash a = b$ the conclusion $\Gamma \vdash P[b]$, where an occurrence of a is replaced by an occurrence of b. Note that NotI-m and =Subst-m are simple examples of domain-independent, logic-related methods, which are needed in addition to domain-specific, mathematically motivated methods as illustrated below in Section 2.3.3. Knowledge-based proof planning expands on these ideas and allows for more general mathematical methods to be encapsulated into the proof planning *methods*.

Control rules represent mathematical knowledge about how to proceed in the proof planning process. They can influence the planner's behavior at choice points (e.g., which goal to tackle next or which method to apply next) by preferring members of the corresponding list of alternatives (e.g., the list of possible goals or the list of possible methods). This way promising search paths are preferred and the search space can be pruned.

Strategies employ a fixed set of methods and control rules and, thus, tackle a theorem by some mathematical standard that happens to be typical for this theorem. The reasoning as to which strategy to employ on a problem is an explicit choice point in MULTI. In particular, MULTI can backtrack from a chosen strategy and commence search with different strategies.

Detailed discussions of ΩMEGA's method and control rule language can be found in [63,65]. A detailed introduction to proof planning with multiple strategies is given in [73, 64] and more recently in [69]. In the following we briefly sketch how proof planning with generic and domain specific methods along with domain specific control strategies can be applied to plan "irrationality of $\sqrt[j]{l}$"-conjectures for arbitrary natural numbers j and l (see also [88]).

2.3.3. Exploiting domain specific knowledge: proof planning $\sqrt[j]{l}$-problems

ΩMEGA can successfully proof plan and proof/disprove the irrationality of $\sqrt[j]{l}$ for arbitrary natural numbers j and l. In order to find a general approach to tackle these problems, we first showed the challenge problem "$\sqrt{2}$ is irrational" (see [97]) and then analyzed proofs for statements such as $\sqrt{8}$, $\sqrt{(3 \cdot 3) - 1}$, or $\sqrt[3]{2}$. We found that some of the concepts and inference steps we used for $\sqrt{2}$ are particular to this problem and do not generalize

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.10 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 10

whereas others do. Thus, the analysis led to some generalized concepts, theorems, and proof steps, which we encoded into methods and control rules, which together form one *planner strategy* for this kind of problems. We shall now discuss the acquired methods and control rules.

The essential idea of the proofs is as follows:

1. Use the MBASE-theorem RAT-CRITERION (it states that for each rational number x, there are integers y and z, such that $x \cdot y = z$, where y and z have no common divisor besides 1) and construct an indirect proof.
2. In order to derive the contradiction show that the two witnesses (i.e., the existential variables y and z) in RAT-CRITERION, which are supposed to have no common divisor, actually do have a common divisor d.
3. In order to find a common divisor transform equations (for example, $\sqrt{2} \cdot n = m \rightarrow 2 \cdot n^2 = m^2$), derive new divisor statements (for example, from $2 \cdot n^2 = m^2$ derive that m^2 has divisor 2, or from the statement that m^2 has divisor 2 derive that m has divisor 2), and derive from given divisor statements new representations of terms, which can be used again for equational transformations (for example, from the statement that m has divisor 2 derive that $m = 2 \cdot k$ for some k).
   Note that we are particularly interested in prime divisors, since only for prime numbers d is it true that if d is a divisor of m^j then d is also a divisor of m. A corresponding theorem is available in ΩMEGA's knowledge base MBASE.

To realize the first idea (1), the planner MULTI has to decide for an indirect proof, apply the theorem RAT-CRITERION, and derive $l \cdot n^j = m^j$ for integers m and n, which are supposed to have no common divisor. These steps are canonical for arbitrary $\sqrt[j]{l}$ problems. Hence, we could implement them all into one method. However, to avoid the well known problem of over-fitting methods, i.e., to make them special just for a particular theorem, we decided to employ already existing methods from other domains: NotI-m (contradiction of negated statements), MAssertion-m (apply a theorem or an axiom from the theory), ExistsE-Sort-m (decompose existentially quantified formulae), AndE-m (decompose conjunctions).

The application of the methods ExistsE-Sort-m, AndE-m, and NotI-m do not need any further control, but the application of MAssertion-m has to be guided by selecting the theorem or axiom to be applied. This is achieved by a control rule apply-ratcriterion, which determines that the theorem RAT-CRITERION should be used for MAssertion-m, whenever there is a goal formula $\sqrt[j]{l}$.

The second idea (2) is realized with the method ContradictionCommonDivisor-m. When MULTI tries to apply the method it searches first for an assumption stating that two terms t_1, t_2 have no common divisor, and then it searches for two (derived) assumptions stating that t_1 and t_2 both have a divisor d. This method is not guided by control rules, but MULTI tries to apply it to some derived assumptions in each planning cycle.

The third idea (3) of the proof technique is encoded into several collaborating methods: TransFormEquation-m, =Subst-m, PrimeFacsProduct-m, PrimeDivPower-m, and CollectDivs-m. The method TransFormEquation-m contains knowledge about suitable equational transformations for our problem domain. It is applied to an equation and derives a new equation. For instance, TransFormEquation-m derives $l \cdot n^j = m^j$ from $\sqrt[j]{l} \cdot n = m$,

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.11 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 11

or it derives $n^2 = 2 \cdot k^2$ from $2 \cdot n^2 = (2 \cdot k)^2$. The method =Subst-m performs equality substitutions.

PrimeFacsProduct-m and PrimeDivPower-m encapsulate the knowledge of how to derive divisor statements. PrimeFacsProduct-m is applied to equations $x = l \cdot y$ (or $l \cdot y = x$) and derives a new assumption which is a conjunction of statements that x has particular prime divisors. The method employs MAPLE to compute the prime divisors of l using MAPLE's function `with(numtheory, factorset)`. It derives that x has to have all prime divisors of l. For instance, from $2 \cdot n^2 = m^2$ PrimeFacsProduct-m derives that m^2 has the prime divisor 2, from $6 \cdot n^2 = m^2$ it derives that m^2 has the prime divisors 2 and 3. PrimeDivPower-m is applied to an assumption that states that y^j has prime divisor d and derives that y has prime divisor d.

For a term t CollectDivs-m searches for assumptions stating that t has some prime divisors. Then, it computes different possible representations of t based on the set of the prime divisors $\{p_1, \ldots, p_n\}$. That is, for each subset $\{p'_1, \ldots, p'_{n'}\}$ of $\{p_1, \ldots, p_n\}$ it adds a new assumption $t = p'_1 \cdots p'_{n'} \cdot c'$ for some integer c'.

TransFormEquation-m, PrimeFacsProduct-m and PrimeDivPower-m are applied whenever possible and no guidance is required. The application of the method CollectDivs-m, however, is guided by the control rule `apply-collectdivs`, which prefers CollectDivs-m with respect to a term t as soon as there are assumptions stating that t has some prime divisors. The application of =Subst-m is guided by the control rule `apply-=subst`, which states that, after an application of CollectDivs-m, the method =Subst-m should be applied in order to use the equations resulting from CollectDivs-m. When a method such as =Subst-m, PrimeFacsProduct-m, or PrimeDivPower-m is applied to some premises, then the same method is afterwards applicable again to the same premises, deriving the same result. To avoid endless loops of such methods, we added the control rule `reject-loop`, which blocks the repeated application of a forward method to the same premises.

2.4. ΩANTS: *agent-oriented theorem proving*

ΩANTS has originally been developed to support interactive theorem proving [18] and later its was extended to a fully automated reasoning system [19,92]. The basic idea of ΩANTS is to encapsulate each inference rule into a pro-active agent, which checks automatically for its own applicability. For each proof situation the $\mathcal{PDS}$ is continuously checked by these agents and thus composes a ranked list of potentially applicable inference rules. In this process all calculus rules, tactics, external system calls and methods, collectively called *inference rules*, are uniformly viewed with respect to three sets: premises, conclusions, and additional parameters. The elements of these three sets are called *arguments* of the inference rule and they usually depend on each other. An inference rule is applicable if at least some of its arguments can be instantiated with respect to the given proof context. The task of the ΩANTS-system is now to determine the applicability of inference rules by computing instantiations for their arguments.

The ΩANTS-architecture consists of two layers. On the bottom layer, possible instantiations of the arguments of individual inference rules are computed. In particular, each inference rule is associated with a blackboard and some concurrent processes, one for each argument of the inference rule. The role of every process is to compute possible in-

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.12 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 12

stantiations for its designated argument of the inference rule, and to record these on the blackboard. The computation is carried out with respect to the given proof context and exploits the information already present on the blackboard, that is, argument instantiations computed by other processes. On the upper layer, the information from the lower layer is used for computing and heuristically ranking the inference rules that are applicable in the current proof state. The heuristically most promising rule is then applied to the central proof object and the data on the blackboards is cleared for the next round of computation.

ΩANTS uses resource reasoning to guide the search [22]. The integration of external reasoning systems such as automated theorem provers, computer algebra systems, or model generators into the architecture of ΩANTS presupposes the declaration of some resource limits these reasoning agents are allowed to spend (e.g., by specifying time-outs). The external systems are encapsulated into inference rules, usually one for each system. For example, an inference rule modeling the application of an ATP has its conclusion argument set as "open goal". A process can then place this open goal onto the blackboard, where it is picked up by a process that applies the prover to it. Any computed proof or partial proof from the external system is again written onto the blackboard from where it is subsequently inserted into the $\mathcal{PDS}$ when the inference rule is applied. While this setup enables proof construction by a collaborative effort of diverse reasoning systems, the cooperation is achieved via the central $\mathcal{PDS}$. This means that all partial results have to be translated back and forth between the syntaxes of the integrated systems and the representation language of the $\mathcal{PDS}$. In some cases efficient communication between inference systems is difficult to achieve [15]. Therefore we have recently developed an alternative model of cooperating systems in ΩANTS which has been successfully applied to the combination of automated higher-order and first-order theorem provers [20].

2.5. External systems

Proof problems require many different skills for their solution and it is desirable to have access to several systems with complementary capabilities, to orchestrate their use, and to integrate their results. ΩMEGA interfaces heterogeneous external systems such as *computer algebra systems* (*CASs*), higher- and first-order *automated theorem proving systems* (*ATPs*), *constraint solvers* (*CSs*), and *model generation systems* (*MGs*).

Their use is twofold: they may provide a solution to a subproblem, or they may give hints for the control of the search for a proof. In the former case, the output of an incorporated reasoning system is translated and inserted as a subproof into the $\mathcal{PDS}$. This is beneficial for interfacing systems that operate at different levels of granularity, and also for a human-oriented display and inspection of a partial proof. In particular we can now check the soundness of each contribution by expanding the inserted subproof to a basic logic-level proof in the $\mathcal{PDS}$ and then verify it by ΩMEGA's proof checker.

Currently, the following external systems are integrated and used in ΩMEGA:

CASs provide symbolic computation, which can be used in two ways: first, to compute hints to guide the proof search (e.g., witnesses for existential variables), and, second, to perform some complex algebraic computation such as to normalize or simplify terms. In the latter case the symbolic computation is directly translated

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.13 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 13

into proof steps in ΩMEGA. CASs are integrated via the transformation and translation module SAPPER [91]. Currently, ΩMEGA uses the systems MAPLE [30] and GAP [85].

ATPs are employed to solve subgoals. Currently ΩMEGA uses the first-order provers BLIKSEM [38], EQP [60], OTTER [61], PROTEIN [10], SPASS [95], WALDMEISTER [50], the higher-order systems TPS [2], and $\mathcal{LEO}$ [16,11], and we plan to incorporate VAMPIRE [82]. The first-order ATPs are connected via TRAMP [62], which is a proof transformation system that transforms resolution-style proofs into assertion-level ND-proofs which can then be integrated into ΩMEGA's $\mathcal{PDS}$. TPS already provides ND-proofs, which can be further processed and checked with little transformational effort [12].

MGs provide either witnesses for free (existential) variables, or counter-models, which show that some subgoal is not a theorem. Hence, they help to guide the proof search. Currently, ΩMEGA uses the model generators SATCHMO [58] and SEM [98].

CSs construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a constraint solver can help to reduce the proof search by checking for inconsistencies of constraints. Currently, ΩMEGA employs $\mathcal{CoSIE}$ [76,100], a constraint solver for inequalities and equations over the field of real numbers.

2.6. Interface and system support

ΩMEGA's graphical user interface $\mathcal{L\Omega UI}$ [90] displays the current $\mathcal{PDS}$ in multiple modalities: a graphical map of the proof tree, a linearized presentation of the proof nodes with their formulae and justifications, a term browser, and a natural language presentation of the proof via *P.rex* (see Figs. 5 and 6).

When inspecting a part of a proof, the user can switch between alternative levels of granularity coexisting in the $\mathcal{PDS}$, for example, by expanding an abstract justification of a proof node into its associated, less abstract partial subproof, which causes appropriate changes in the other presentation modes. Moreover, an interactive natural language *explanation* of the proof is provided by the system *P.rex* [40,39,41], which is adaptive in the following sense: it explains a proof step at the most abstract level (which the user is assumed to know) and then reacts flexibly to questions and requests, possibly at a lower level of granularity, for example, by detailing some ill-understood subproof.

Another system support is the guidance mechanism provided by the suggestion module ΩANTS (see Section 2.4), which searches pro-actively for possible actions that may be helpful in finding a proof and presents them in a preference list.

2.7. Case studies

Early developments of proof planning in Alan Bundy's group at Edinburgh used proofs by induction as their favorite case studies [25]. The ΩMEGA system has been used in several other case studies, which illustrate in particular the interplay of the various components, such as proof planning supported by heterogeneous external reasoning systems.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.14 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 14

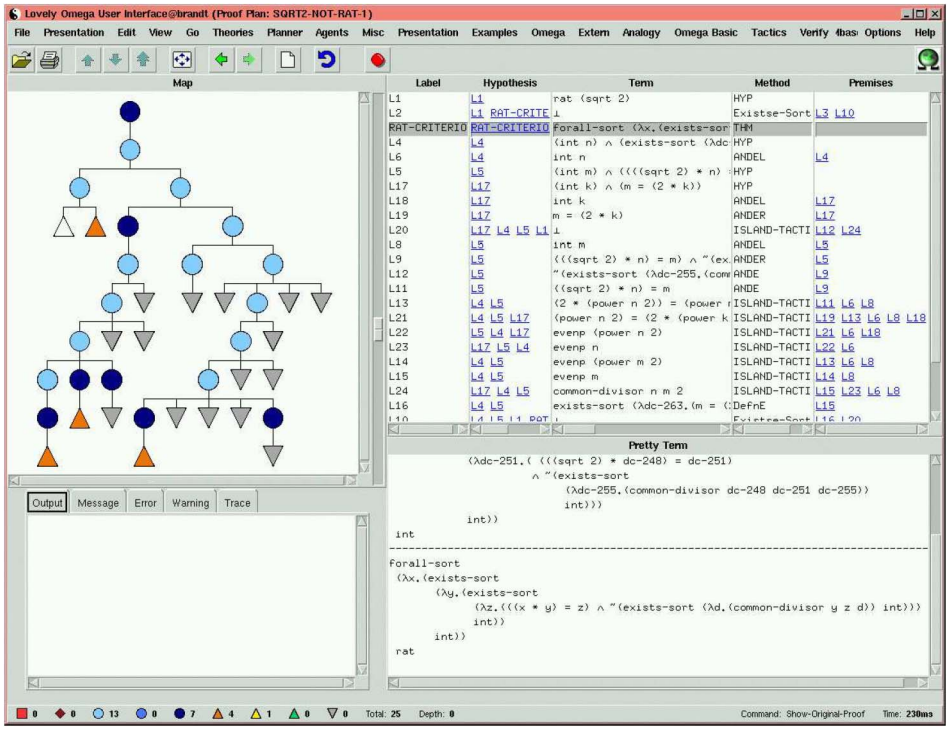


Fig. 5. Multi-modal proof presentation in the graphical user interface $\mathcal{LOUI}$.

A typical example for a class of problems that cannot be solved by traditional automated theorem provers is the class of ϵ–δ-proofs [74,71]. This class was originally proposed by Woody Bledsoe [23] as a challenge and it comprises theorems such as LIM+ and LIM*, where LIM+ states that the limit of the sum of two functions equals the sum of their limits and LIM* makes the corresponding statement for multiplication. The difficulty of this domain arises from the need for arithmetic computation in order to find a suitable instantiation of free (existential) variables (such as a δ depending on an ϵ). Crucial for the success of ΩMEGA's proof planning is the integration of suitable experts for these tasks: the arithmetic computation is done by the computer algebra system MAPLE, and an appropriate instantiation for δ is computed by the constraint solver $\mathcal{CoSIE}$. We have been able to solve all challenge problems suggested by Bledsoe and many more theorems in this class taken from a standard textbook on real analysis [9].

Another class of problems we tackled with proof planning is concerned with residue classes [67,66]. In this domain we showed theorems such as: "the residue class structure $(\mathbb{Z}_5, \bar{+})$ is associative", "it has a unit element", and similar properties, where $\mathbb{Z}_5$ is the set of all congruence classes modulo 5 (i.e., $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$) and $\bar{+}$ is the addition on residue classes. We have also investigated whether two given structures are isomorphic or not and altogether we have proved more than 10,000 theorems of this kind (see [92]). Although the problems in this domain are not too difficult and still within the success range

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.15 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 15

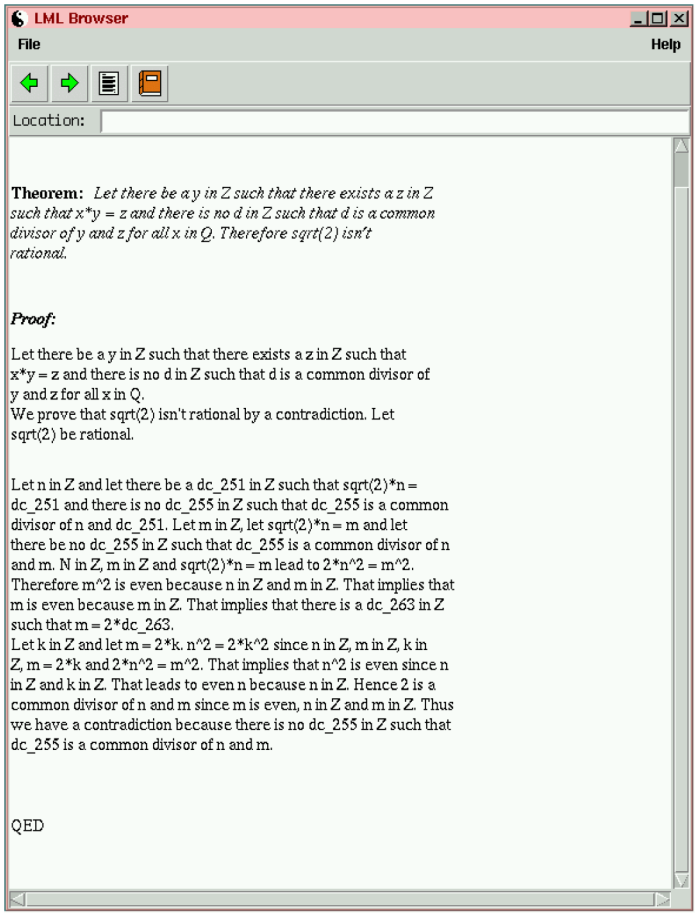


Fig. 6. Natural language proof presentation by *P.rex* in $\mathcal{LOUI}$.

of a traditional automated theorem prover, it was nevertheless an interesting case study for proof planning, since multi-strategy proof planning generated substantially different proofs based on entirely different proof ideas.

Another important proof technique is Cantor's diagonalization technique and we also developed methods and strategies for this class [31]. Important theorems we have been able to prove are the undecidability of the halting problem and Cantor's theorem (cardinality of the set of subsets), the non-countability of the reals in the interval $[0, 1]$ and of the set of total functions, and similar theorems.

Finally, a good example for a standard proof technique is the excess-literal-number technique. This is routinely used for completeness proofs of refinements of resolution, where the theorem is usually first shown at the ground level using the excess-literal-number technique and then ground completeness is lifted to the predicate calculus level. We have done this for many refinements of resolution with ΩMEGA [45].

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.16 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 16

However, ΩMEGA's main aim is to become a proof assistant for the working mathematician. Hence, it should support interactive proof development at a human-oriented level of granularity. The already mentioned theorem that $\sqrt{2}$ is irrational, and its well-known proof dating back to the School of Pythagoras, provides an excellent challenge to evaluate whether this ambitious goal has been reached. In [97] seventeen systems that have solved the $\sqrt{2}$-problem show their results. The protocols of their respective sessions have been compared on a multi-dimensional scale in order to assess the "naturalness" by which real mathematical problems of this kind can be shown. This represents an important shift of emphasis in the field of automated deduction away from the somehow artificial problems of the past—as represented, for example, in the test set of the TPTP library [93]—back to real mathematical challenges. We participated in this case study essentially with three different contributions. Our initial contribution was an interactive proof in ΩMEGA without adding any special domain knowledge to the system. This demonstrates the use of ΩMEGA as a tactical theorem prover (see [14]). The most important albeit not entirely new lesson to be learned from this experiment is that the level of granularity common in most automated and tactical theorem proving environments is far too low. While our proof representation in this first study is already an abstraction (called the *assertion level* in [51]) from the calculus level typical for most ATPs, it is nevertheless clear that as long as a system does not hide all these excruciating details, no working mathematician will feel inclined to use such a system. In fact, this is in our opinion one of the critical impediments for using first-order ATPs and one, albeit not the only one, of the reasons why they are not used as widely as computer algebra systems. This is the crucial issue of the ΩMEGA project and our main motivation for departing from the classical paradigm of automated theorem proving about fifteen years ago.

Our second contribution to the case study of the $\sqrt{2}$-problem is based on interactive *island planning* [70], a technique that expects an outline of the proof, i.e., the user provides main subgoals, called *islands*, together with their assumptions. In fact, we are able to proof plan arbitrary $\sqrt[j]{l}$-problems as sketched in Section 2.3.3. Hence, the user can write down *his* proof idea in a natural way with as many gaps as there are open at this first stage of the proof. Closing the gaps is ideally fully automatic, in particular, by exploiting external systems. However, for difficult theorems it is necessary more often than not that the user provides additional information and applies the island approach recursively. In comparison to our first tactic-based solution the island style supports a much more abstract and user-friendly interaction level. The proofs are now at a level of granularity similar to proofs in mathematical textbooks.

Our third contribution to the case study of the $\sqrt{2}$-problem are fully automatically planned and expanded proofs of $\sqrt[j]{l}$-problems for arbitrary natural numbers j and l. The details of this important case study, that shows best what can (and what cannot) be achieved with current proof planning technology are presented in [88,89,14].

2.8. Discussion

2.8.1. Proof-planning as an alternative approach to automated theorem proving?

The most important question to ask here is: Can we find the essential and creative steps automatically, for example, for the $\sqrt{2}$-problem discussed in Section 2.3.3? The answer is

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.17 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 17

yes, as we have shown in [88]. However, while we can answer the question in the affirmative, not every reader may be convinced, as our solution touches upon a subtle point, which opens the Pandora Box of critical issues in the paradigm of proof planning [28]: It is always easy to write some specific methods, which perform just the steps in the interactively found proof and then calls the proof planner MULTI to fit the methods together into a proof plan for the given problem. This, of course, shows nothing of substance: Just as we could write down all the definitions and theorems required and sufficient for the problem in first-order predicate logic and then hand them to a first-order prover,[5] we would just hand-code the final solution into appropriate methods.

Instead, the goal of the game is to find *general* methods for a whole class of theorems within some theory that can solve not only this particular problem, but also all the other theorems in that class. While our approach essentially follows the proof idea of the interactively constructed proof for the $\sqrt{2}$-problem, it relies essentially on more general concepts.

However, this is certainly not the end of the story. In order to evaluate the appropriateness of a proof planning approach we suggest the following four criteria:

(1) How general and how rich in mathematical content are the methods and control rules?
(2) How much search is involved in the proof planning process?
(3) What kind of proof plans, that is, what kind of proofs, can we find?
(4) If the proof planning procedure fails on some given conjecture, how likely is it that the given conjecture is not a theorem?

These criteria should allow us to judge how general and how robust our solution is. The art of proof planning is to acquire domain knowledge that, on the one hand, comprises meaningful mathematical techniques and powerful heuristic guidance, and, on the other hand, is general enough to tackle a broad class of problems. For instance, as one extreme, we could have methods that encode ΩMEGA's ND-calculus and we could run MULTI without any control. This approach would certainly be very general, but MULTI would fail to prove any interesting problems. As the other extreme, we could cut a known proof into pieces, and code the pieces as methods. Guided by control rules that always pick the next right piece of the proof, MULTI would assemble the methods again to the original proof without performing any search. However, in that case if MULTI fails to find a proof then it is not unlikely that the conjecture is nevertheless a theorem.

2.8.2. What lessons have we learned?

The problem domains on which proof planning has been applied so far are small but nevertheless typical. Some interesting observations gained from this experience are the following:

(1) The devil is in the detail, that is, it is always possible to hide the crucial creative step (represented as a specific method or represented in the object language by an appropriate lemma) and to pretend a level of generality that has not actually been achieved.

[5] This was done when OTTER tackled the $\sqrt{2}$-problem; see [97] for the original OTTER case study and [14] for its replay with ΩMEGA.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.18 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 18

To evaluate a solution *all* tactics, methods, theorems, lemmata and definitions have to be made explicit.

(2) The enormous distance between the well-known (top-level) proof of the Pythagorean School, which consists of about a dozen proof steps in comparison to the final (non-optimized) proof at ΩMEGA's ND-calculus level with about 750 inference steps is striking. This is, of course, not a new insight. While mathematics can *in principle* be reduced to purely formal logic-level reasoning as demonstrated by Russell and Whitehead as well as the Hilbert School, nobody would actually want to do so *in practice* as the Bourbaki group of French mathematicians states explicitly: The first quarter of the first volume in the several dozen volume set on the foundation of mathematics starts with elementary, logic-level reasoning and then proceeds with the crucial sentence [24]: "No great experience is necessary to perceive that such a project [of complete formalization] is absolutely unrealizable: the tiniest proof at the beginning of the theory of sets would already require several hundreds of signs for its complete formalization".

(3) Finally and more to the general point of interest in mathematical support systems: Now that we can prove theorems in the $\sqrt[j]{l}$-problem class, the skeptical reader may still ask: *So what?* Will this ever lead to a *general* system for mathematical proof assistance? We have shown that the class of ϵ–δ-proofs for limit theorems can indeed be solved with a few dozen mathematically meaningful methods and control rules (see [74,72, 63]). Similarly, the domain of group theory with its class of residue theorems can be formalized with even fewer methods (see [68,66,67]).[6] An interesting observation is also that these methods by and large correspond to the kind of mathematical knowledge a freshman would have to learn to master this level of professionalism.

Do the above observations now hold for our $\sqrt[j]{l}$-problems? The unfortunate answer is probably *No!* Imagine the subcommittee of the United Nations in charge of the maintenance of the global mathematical knowledge base in a hundred years from now. Would they accept the entry of our methods, tactics and control rules for the $\sqrt[j]{l}$-problems? Probably not!

Factual mathematical knowledge is preserved in books and monographs, *but the art of doing mathematics* [81,49] is passed on by word of mouth from generation to generation. The methods and control rules of the proof planner correspond to important mathematical techniques and "ways to solve it" [81], and they make this implicit and informal mathematical knowledge explicit and formal.

The theorems about $\sqrt[j]{l}$-problems are shown by contradiction, that is, the planner derives a contradiction from the equation $l \cdot n^j = m^j$, where n and m are integers with no common divisor. However, these problems belong to the more general class to determine whether two complex mathematical objects $\mathcal{X}$ and $\mathcal{Y}$ are equal. A general mathematical principle for comparison of two complex objects is to look at their characteristic properties, for example, their normal forms or some other uniform notation in the respective theory.

[6] The generally important observation is not, of course, whether we need a dozen or a hundred methods, but that we don't need a few thousand or a million. A few dozen methods seem to be generally enough for a restricted mathematical domain.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.19 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 19

And this is the crux of the matter: to find general mathematical principles and encode them into appropriate methods, control rules and strategies such that an appropriately large *class of problems* can be solved with these methods.

3. The future: what next?

The longterm goal of the ΩMEGA project is an integrated environment of tools supporting a wide range of typical mathematical activities. Examples of mathematical activities are computing, proving, solving, modeling, verifying, structuring, searching, inventing, publishing, explaining, illustrating, etc. We anticipate that in the long run assistance systems for mathematics will change mathematical practice and they will have a strong societal impact, not least in the sense that a powerful infrastructure for mathematical research and education will become commercially available. Computer supported mathematical reasoning tools and integrated assistance systems will be further specialized to have a strong impact also in many other theoretical fields such as safety and security verification of computer software and hardware, theoretical physics and chemistry and other related subjects.

The research questions we plan to investigate in the immediate future arise from the following scenario of preparing a mathematical research article with formalized content in a textbook style and in professional type-setting quality.

Mathematical research article preparation scenario. The author starts writing a new mathematical document in a format suitable for publication by using mathematical concepts from different mathematical domains. New mathematical concepts or lemmata introduced in the paper should result in corresponding new formal objects. Furthermore, when writing the document appropriate service tools can be used to compute intermediate results for an illustrating example, querying mathematical databases for mathematical publications introducing similar concepts and send subproblems to be solved to special reasoning or computation systems. Proofs of lemmata and theorems contained in the document should be amenable to formal proof checking techniques such that the submitted paper can be proof checked semi-automatically by the journal. A long-term goal may be fully automated verification.

3.1. Formalization and proving at a higher level of granularity

Mathematical reasoning with the ΩMEGA system is at the comparatively high level of the proof planning methods. However, as these methods have to be expanded eventually to our base-level ND-calculus, the system still suffers from the effect and influence this logical representation has. In contrast, the proofs developed by a mathematician, say for a mathematical publication, and the proofs developed by a student in a mathematical tutoring system are typically developed at a less fine-grained argumentative level. This level has been formally categorized as *proofs at the assertion level* [51]. While so far assertion level proofs needed to be constructed from the underlying ND-calculus proof in ΩMEGA, the recently developed CORE system [3,4] supports proof construction directly on the assertion level and defines a communication infrastructure, i.e., a mediator, between the user and the automatic reasoning procedures. Currently, we exchange ΩMEGA's ND-calculus

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.20 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 20

by the CoRe calculus, which supports the presentation of the proof state via relevant contextual information about possible proof continuations and also supports hierarchical proof development. The proof theory of CoRe is uniform for a variety of logics and exploits proof-theoretic annotations in formulas for an assertion-level contextual reasoning style.

An unfortunate aspect of typical mathematical proofs is their *under-specification*,[7] for example, missing references to premise assertions, to rule and instantiation specifications or simply the specific part of the formula the author is talking about. One particular challenge here is to define an appropriate proof format which allows to represent human-constructed proofs as they are and to develop means to resolve the under-specification later by deductive methods. First steps in that direction and a description of the types of under-specifications can be found in [5,13].

3.2. Mathematical knowledge representation

A mathematical proof assistance system relies upon different kinds of knowledge: First, of course, the formalized mathematical domain as organized in structured theories of definitions, lemmata, and theorems. Secondly, there is mathematical knowledge on how to prove a theorem, which is encoded in tactics and methods, in ΩANTS agents, in control knowledge and in strategies. This type of knowledge can be general, theory specific or even problem specific.

The integration of a mathematical proof assistant into the typical and everyday activities of a mathematician requires, however, other types of knowledge as well. For example, a tutoring system for maths students may rely upon a database with different samples of proofs and proof plans linked by meta-data in order to advise the student. Another example is the support for mathematical publications: The documents containing both formalized and non-formalized parts need to be related to specific theories, lemmata, theorems, and proofs. This raises the research challenge on how the usual structuring mechanisms for mathematical theories (such as theory hierarchies or the import of theories via renaming or general morphisms) can be extended to tactics and methods as well as to proofs, proof plans and mathematical documents. Furthermore, changing any of these elements requires maintenance support as any change in one part may have consequences in other parts. For example, the validity of a proof needs to be checked again after changing parts of a theory, which in turn may affect the validity of the mathematical documents. Thus, technology supporting the *management of change* [7,8,6,52,77], originally developed for evolutionary formal software engineering at the DFKI,[8] will now be integrated into the ΩMEGA system as well.

Hierarchically structured mathematical knowledge, i.e., an ontology of mathematical theories and assertions has initially been stored in ΩMEGAs hardwired mathematical

[7] "Under-specification" is a technical term borrowed from research on the semantics of natural language. Roughly it means that certain aspects in the semantic representation of a natural language utterance are left uninterpreted, such that their proper treatment can be deferred to later stages of processing in which more contextual information is available.

[8] http://www.dfki.de.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.21 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 21

knowledge base. This mathematical knowledge base was later (end of the 90s) out-sourced and linked to the development of MBASE [43]. We now assume that a mathematical knowledge base also maintains domain specific control rules, strategies, and linguistic knowledge. While this is not directly a subject of research in the ΩMEGA project, relying here on other groups of the MKM community and especially the OMDOC format,[9] we shall nevertheless concentrate on one aspect, namely how to find the appropriate information as outlined in the next paragraph.

3.2.1. Semantic mediators for mathematical knowledge bases

Knowledge acquisition and retrieval in the currently emerging large repositories of formalized mathematical knowledge should not be based purely on syntactic matching, but it needs to be supported by *semantic* mediators.

To prove a mathematical theorem in a particular domain is initially blind. Indeed, in order to prevent a search space explosion, only part of the relevant knowledge is made available at the start. For instance, in the ΩMEGA system the proof planner MULTI selects a subset of the available knowledge which consists, for each theorem, of a set of assertions (axioms, definitions, lemmata), tactics and proof-planning methods. As this selection is naturally incomplete, there is the need to incrementally incorporate additional knowledge if needed.

We are working on appropriately limited higher-order reasoning agents for domain- and context-specific retrieval of mathematical knowledge from a mathematical knowledge base. For this we shall adapt a two stage approach as in [17], which combines syntactically oriented pre-filtering with semantic analysis. The pre-filter employ efficiently processable criteria based on meta-data and ontologies that identify sets of candidate theorems of a mathematical knowledge base that are potentially applicable to a focused proof context. The higher-order agents then act as post-filters to exactly determine the applicable theorems of this set.

3.3. MathServ: a global web for mathematical services

The Internet provides a vast collection of data and computational resources. For example, a travel booking system combines different information sources, such as the search engines, price computation schemes, and the travel information in distributed very large databases, in order to answer complex booking requests. The access to such specialized travel information sources has to be planned, the obtained results combined, and, in addition, the consistency of time constraints has to be guaranteed. We want to transfer and apply this methodology to mathematical problem solving and develop a system that plans the combination of several mathematical information sources (such as mathematical databases), computer algebra systems, and reasoning processes (such as theorem provers or constraint solvers). Based on the well-developed MATHWEB-SB network of mathematical services, the existing client-server architecture will be extended by advanced problem solving capabilities and semantic brokering of mathematical services (see [101]).

[9] http://www.mathweb.org/omdoc/.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.22 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 22

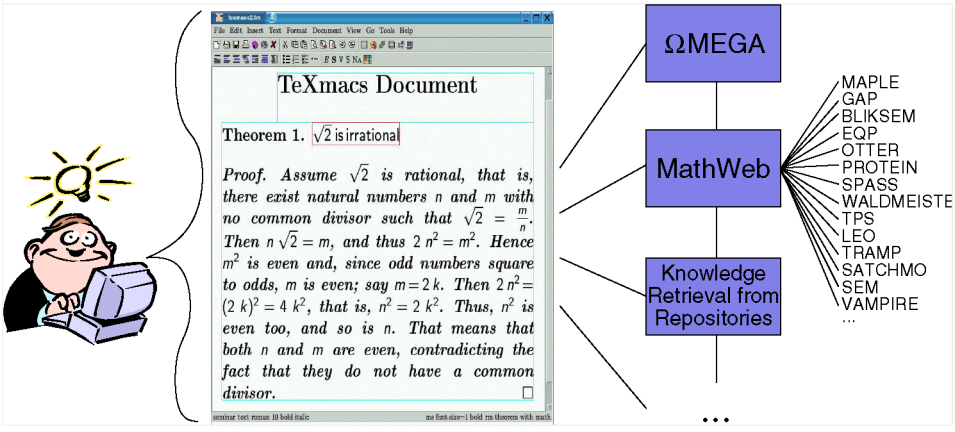


Fig. 7. Documents in TeXmacs: The user will be supported by different mathematical reasoning services that "understand" the document content.

3.4. *Support for mathematical activities*

Proof construction is an important but only a small part of a much wider range of mathematical activities an assistance system for mathematics should support.

3.4.1. *Certified mathematics texts*

A mathematician or software engineer writes a paper usually in a LaTeX-like environment. The definitions, lemmata, theorems and especially their proofs give rise to extensions of the original theory he started with. If the proofs of the new theorems and their consistency with previous assertions are computer checked, we have mathematical documents in a publishable style which in addition are formally validated, hence obtaining *certified mathematical documents*. A first step in that direction is currently under development by linking the WYSIWYG mathematical editor TEXMACS [94] with the ΩMEGA system (see Fig. 7).

The TEXMACS-system provides LaTeX-like editing and macro-definition features, and we are defining macros for theory-specific knowledge such as types, constants, axioms, and lemmata. This allows us to translate new textual definitions and lemmata into the formal representation, as well as to translate (partial) textbook proofs into (partial) proof plans.

3.4.2. *Mathematical advice in tutoring systems*

We are also involved in the DFKI project ActiveMath [75], which develops an e-learning tool for tutoring maths students, in particular in advising a student how to prove a theorem. This scenario is currently also under investigation in the DIALOG[10] project [13,21] and, aside from all linguistic analysis problems, gives rise to the problem to bridge the gap

[10] The DIALOG project is a collaboration between the Computer Science and Computational Linguistics departments of Saarland University as part of the Collaborative Research Center on *Resource-Adaptive Cognitive Processes*, SFB 378 (http://www.coli.uni-saarland.de/projects/sfb378/).

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.23 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 23

between the human style of proofs and machine-oriented proof representations. Human-authored proofs are often imprecise in several respects, namely (i) the used inference rule is not mentioned, (ii) some of the premises needed for a step in the derivation are not mentioned, and (iii) some steps of the derivation are completely omitted.

Another interesting and novel application for theorem proving systems in the DIALOG project is proof step evaluation (see [21]): Each proof step uttered by a student within a tutorial context has to be analyzed with respect to the following criteria:

Soundness: Can the proof step be reconstructed by a formal inference system and logically and tutorially verified?

Granularity: Is the 'argumentative complexity' or 'size' of the proof step logically and tutorially acceptable?

Relevance: Is the proof step logically and tutorially useful for achieving the final goal?

References

[1] S. Allen, R. Constable, R. Eaton, C. Kreitz, L. Lorigo, The Nuprl open logical environment, in: [59].
[2] P.B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, H. Xi, TPS: A theorem proving system for classical type theory, J. Automat. Reason. 16 (3) (1996) 321–353.
[3] S. Autexier, Hierarchical contextual reasoning, PhD thesis, Computer Science Department, Saarland University, Saarbrücken, Germany, 2003.
[4] S. Autexier, The CORE calculus, in: R. Nieuwenhuis (Ed.), Proceedings of the 20th Conference on Automated Deduction (CADE-20), Tallinn, Estonia, in: Lecture Notes in Artificial Intelligence, vol. 3632, Springer, Berlin, 2005, pp. 84–98.
[5] S. Autexier, C. Benzmüller, A. Fiedler, H. Horacek, Q. Bao Vo, Assertion-level proof representation with under-specification, Electronic Notes in Theoret. Comput. Sci. 93 (2003) 5–23.
[6] S. Autexier, D. Hutter, Maintenance of formal software development by stratified verification, in: M. Baaz, A. Voronkov (Eds.), Proceedings of LPAR'02, Tbilisi, Georgia, in: Lecture Notes in Computer Science, Springer, Berlin, 2002.
[7] S. Autexier, D. Hutter, T. Mossakowski, A. Schairer, The development graph manager MAYA, in: H. Kirchner, C. Ringeissen (Eds.), Proceedings 9th International Conference on Algebraic Methodology and Software Technology (AMAST'02), in: Lecture Notes in Computer Science, vol. 2422, Springer, Berlin, 2002.
[8] S. Autexier, T. Mossakowski, Integrating HOL-CASL into the development graph manager MAYA, in: A. Armando (Ed.), Proceedings of FROCOS'02, in: Lecture Notes in Artificial Intelligence, vol. 2309, Springer, Berlin, 2002, pp. 2–17.
[9] R. Bartle, D. Sherbert, Introduction to Real Analysis, second ed., Wiley, New York, 1982.
[10] P. Baumgartner, U. Furbach, PROTEIN, a PROver with a theory INterface, in: [27], pp. 769–773.
[11] C. Benzmüller, Equality and extensionality in higher-order theorem proving, PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.
[12] C. Benzmüller, M. Bishop, V. Sorge, Integrating TPS and ΩMEGA, J. Universal Comput. Sci. 5 (1999) 188–207.
[13] C. Benzmüller, A. Fiedler, M. Gabsdil, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann, D. Tsovaltzi, B. Quoc Vo, M. Wolska, Tutorial dialogs on mathematical proofs, In: Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems, Acapulco, Mexico, 2003, pp. 12–22.
[14] C. Benzmüller, A. Fiedler, A. Meier, M. Pollet, Irrationality of $\sqrt{2}$—a case study in ΩMEGA, Seki-Report SR-02-03, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.
[15] C. Benzmüller, M. Jamnik, M. Kerber, V. Sorge, Experiments with an agent-oriented reasoning system, in: F. Baader, G. Brewka, T. Eiter (Eds.), KI 2001: Advances in Artificial Intelligence, Proceedings of Joint

German/Austrian Conference on AI, Vienna, Austria, in: Lecture Notes in Artificial Intelligence, vol. 2174, Springer, Berlin, 2001, pp. 409–424.
[16] C. Benzmüller, M. Kohlhase, LEO—a higher-order theorem prover, in: [54].
[17] C. Benzmüller, A. Meier, V. Sorge, Bridging theorem proving and mathematical knowledge retrieval, in: D. Hutter, W. Stephan (Eds.), Festschrift in Honour of Jörg Siekmann's 60s Birthday, in: Lecture Notes in Artificial Intelligence, vol. 2605, Springer, Berlin, 2004.
[18] C. Benzmüller, V. Sorge, A blackboard architecture for guiding interactive proofs, in: [46].
[19] C. Benzmüller, V. Sorge, ΩANTS—An open approach at combining Interactive and Automated Theorem Proving, in: [53].
[20] C. Benzmüller, V. Sorge, M. Jamnik, M. Kerber, Can a higher-order and a first-order theorem prover cooperate? in: F. Baader, A. Voronkov (Eds.), Proceedings of the 11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR), in: Lecture Notes in Artificial Intelligence, vol. 3452, Springer, Berlin, 2005, pp. 415–431.
[21] C.E. Benzmüller, Q.B. Vo, Mathematical domain reasoning tasks in natural language tutorial dialog on proofs, in: M. Veloso, S. Kambhampati (Eds.), Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, PA, AAAI Press/MIT Press, 2005, pp. 516–522.
[22] C. Benzmüller, V. Sorge, Critical agents supporting interactive theorem proving, in: P. Borahona, J.J. Alferes (Eds.), Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA'99), Evora, Portugal, in: Lecture Notes in Artificial Intelligence, vol. 1695, Springer, Berlin, 1999, pp. 208–221.
[23] W. Bledsoe, Challenge problems in elementary calculus, J. Automat. Reason. 6 (1990) 341–359.
[24] N. Bourbaki, Theory of sets, Elements of Mathematics, vol. 1, Addison-Wesley, Reading, MA, 1968.
[25] A. Bundy, The use of explicit plans to guide inductive proofs, in: [57], pp. 111–120.
[26] A. Bundy, A science of reasoning, in: G. Plotkin, J.-L. Lasser (Eds.), Computational Logic: Essays in Honor of Alan Robinson, MIT Press, Cambridge, MA, 1991, pp. 178–199.
[27] A. Bundy (Ed.), Proceedings of the 12th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, vol. 814, Springer, Berlin, 1994.
[28] A. Bundy, A critique of proof planning, in: Computational Logic: Logic Programming and Beyond, in: Lecture Notes in Computer Science, vol. 2408, Springer, Berlin, 2002, pp. 160–177.
[29] A. Bundy, F. van Harmelen, C. Horn, A. Smaill, The Oyster-Clam system, in: M. Stickel (Ed.), Proceedings of the 10th Conference on Automated Deduction, Kaiserslautern, Germany, in: Lecture Notes in Computer Science, vol. 449, Springer, Berlin, 1990, pp. 647–648.
[30] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, S. Watt, First Leaves: A Tutorial Introduction to Maple V, Springer, Berlin, 1992.
[31] L. Cheikhrouhou, J. Siekmann, Planning diagonalization proofs, in: [46], pp. 167–180.
[32] L. Cheikhrouhou, V. Sorge, PDS—A three-dimensional data structure for proof plans, In: Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000), Monastir, Tunisia, 2000.
[33] A. Church, A formulation of the simple theory of types, J. Symbolic Logic 5 (1940) 56–68.
[34] Coq Development Team, The Coq proof assistant reference manual, INRIA, 1999–2003, see http://coq.inria.fr/doc/main.html.
[35] M. Davis, The prehistory and early history of automated deduction, in: J. Siekmann, G. Wrightson (Eds.), Automation of Reasoning, vol. 2, Classical Papers on Computational Logic 1967–1970 of Symbolic Computation, Springer, Berlin, 1983.
[36] M. Davis, The early history of automated deduction, in: A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, vol. I, Elsevier Science, Amsterdam, 2001, pp. 3–15, Chapter 1.
[37] M. Davis (Ed.), The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Dover Publications, New York, 2004.
[38] H. de Nivelle, Bliksem 1.10 user manual, Technical Report, Max-Planck-Institut für Informatik, 1999.
[39] A. Fiedler, Dialog-driven adaptation of explanations of proofs, in: B. Nebel (Ed.), Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, Morgan Kaufmann, San Mateo, CA, 2001, pp. 1295–1300.
[40] A. Fiedler, P.rex: An interactive proof explainer, in: [48].
[41] A. Fiedler, User-adaptive proof explanation, PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.25 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 25

[42] A. Franke, M. Kohlhase, System description: MBase, an open mathematical knowledge base, in: [59].
[43] A. Franke, M. Kohlhase, System description: Mbase, an open mathematical knowledge base, in: [59].
[44] H. Ganzinger (Ed.), Proceedings of the 16th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, vol. 1632, Springer, Berlin, 1999.
[45] H. Gebhard, Beweisplanung für die Beweise der Vollständigkeit verschiedener Resolutionskalküle in ΩMEGA, Master's thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.
[46] F. Giunchiglia (Ed.), Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98), Lecture Notes in Artificial Intelligence, vol. 1480, Springer, Berlin, 1998.
[47] M. Gordon, T. Melham, Introduction to HOL—A Theorem Proving Environment for Higher Order Logic, Cambridge University Press, Cambridge, 1993.
[48] R. Goré, A. Leitsch, T. Nipkow (Eds.), Automated Reasoning—1st International Joint Conference, IJCAR 2001, Lecture Notes in Artificial Intelligence, vol. 2083, Springer, Berlin, 2001.
[49] J. Hadamard, The Psychology of Invention in the Mathematical Field, Dover Publications, New York, 1949, 1944.
[50] Th. Hillenbrand, A. Jaeger, B. Löchner, System description: Waldmeister—improvements in performance and ease of use, in: [44], pp. 232–236.
[51] X. Huang, Reconstructing proofs at the assertion level, in: [27], pp. 738–752.
[52] D. Hutter, Management of change in structured verification, in: Proceedings of Automated Software Engineering, ASE-2000, IEEE, 2000.
[53] M. Kerber, M. Kohlhase (Eds.), 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000), AK Peters, 2000.
[54] C. Kirchner, H. Kirchner (Eds.), Proceedings of the 15th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, vol. 1421, Springer, Berlin, 1998.
[55] H. Kirchner, C. Ringeissen (Eds.), Frontiers of Combining systems: Third International Workshop, FroCoS 2000, Lecture Notes in Artificial Intelligence, vol. 1794, Springer, Berlin, 2000.
[56] M. Kohlhase, A. Franke, MBASE: Representing knowledge and context for the integration of mathematical software systems, J. Symbolic Comput. 32 (4) (2001) 365–402 (special issue on the Integration of Computer Algebra and Deduction Systems).
[57] E. Lusk, R. Overbeek (Eds.), Proceedings of the 9th Conference on Automated Deduction, Argonne, Illinois, Lecture Notes in Computer Science, vol. 310, Springer, Berlin, 1988.
[58] R. Manthey, F. Bry, SATCHMO: A theorem prover implemented in Prolog, in: [57], pp. 415–434.
[59] D. McAllester (Ed.), Proceedings of the 17th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, vol. 1831, Springer, Berlin, 2000.
[60] W. McCune, Solution of the Robbins problem, J. Automat. Reason. 19 (3) (1997) 263–276.
[61] W.W. McCune, Otter 3.0 reference manual and guide, Technical Report ANL-94-6, Argonne National Laboratory, Argonne, IL 60439, USA, 1994.
[62] A. Meier, TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level, in: [59].
[63] A. Meier, Proof planning with multiple strategies, PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2004.
[64] A. Meier, E. Melis, MULTI: A multi-strategy proof planner (system description), in: R. Nieuwenhuis (Ed.), Proceedings of the 20th Conference on Automated Deduction (CADE-20), Tallinn, Estonia, in: Lecture Notes in Artificial Intelligence, vol. 3632, Springer, Berlin, 2005, pp. 250–254.
[65] A. Meier, E. Melis, M. Pollet, Towards extending domain representations, Seki Report SR-02-01, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.
[66] A. Meier, M. Pollet, V. Sorge, Classifying isomorphic residue classes, in: R. Moreno-Diaz, B. Buchberger, L. Freire (Eds.), A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001), in: Lecture Notes in Computer Science, vol. 2178, Springer, Berlin, 2001, pp. 494–508.
[67] A. Meier, M. Pollet, V. Sorge, Comparing approaches to the exploration of the domain of residue classes, in: S. Linton, R. Sebastiani (Eds.), J. Symbolic Comput. 34 (4) (2002) 287–306 (special issue on the Integration of Automated Reasoning and Computer Algebra Systems).

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.26 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 26

[68] A. Meier, V. Sorge, Exploring properties of residue classes, in: [53].
[69] A. Meier, E. Melis, J. Siekmann, Proof planning with multiple strategies, Artificial Intelligence, submitted for publication.
[70] E. Melis, Island planning and refinement, Seki-Report SR-96-10, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1996.
[71] E. Melis, AI-techniques in proof planning, in: H. Prade (Ed.), Proceedings of the 13th European Conference on Artificial Intelligence, Brighton, UK, John Wiley & Sons, Chichester, UK, 1998, pp. 494–498.
[72] E. Melis, AI-techniques in proof planning, in: European Conference on Artificial Intelligence, Brighton, UK, Kluwer, Dordrecht, 1998, pp. 494–498.
[73] E. Melis, A. Meier, Proof planning with multiple strategies, in: J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagivand, P. Stuckey (Eds.), First International Conference on Computational Logic (CL-2000), London, UK, in: Lecture Notes in Artificial Intelligence, vol. 1861, Springer, Berlin, 2000, pp. 644–659.
[74] E. Melis, J. Siekmann, Knowledge-based proof planning, Artificial Intelligence 115 (1) (1999) 65–105.
[75] E. Melis, J. Siekmann, Activemath: An Intelligent Tutoring System for Mathematics, vol. 3070, Springer, Berlin, 2004, pp. 91–101.
[76] E. Melis, J. Zimmer, T. Müller, Integrating constraint solving into proof planning, in: [55].
[77] T. Mossakowski, S. Autexier, D. Hutter, Extending development graphs with hiding, in: H. Hussmann (Ed.), Proceedings of Fundamental Approaches to Software Engineering (FASE 2001), Genova, in: Lecture Notes in Computer Science, vol. 2029, Springer, Berlin, 2001, pp. 269–283.
[78] T. Nipkow, L.C. Paulson, M. Wenzel, Isabelle/HOL: A Proof Assistant for Higher-Order Logic, in: Lecture Notes in Computer Science, vol. 2283, Springer, Berlin, 2002.
[79] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, M. Srivas, PVS: Combining specification, proof checking, and model checking, in: R. Alur, T. Henzinger (Eds.), Computer-Aided Verification, CAV '96, New Brunswick, NJ, in: Lecture Notes in Computer Science, vol. 1102, Springer, Berlin, 1996, pp. 411–414.
[80] L. Paulson, Isabelle: A Generic Theorem Prover, Lecture Notes in Computer Science, vol. 828, Springer, Berlin, 1994.
[81] G. Polya, How to Solve it, Princeton University Press, Princeton, NJ, 1973.
[82] A. Riazanov, A. Voronkov, Vampire 1.1 (system description), in: [48].
[83] J. Richardson, A. Smaill, I. Green, System description: Proof planning in higher-order logic with λClam, in: [54].
[84] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1) (1965) 23–41.
[85] M. Schönert, et al., GAP—Groups, Algorithms, Programming, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.
[86] J. Siekmann, Geschichte des Automatischen Beweisens (History of Automated Deduction), in: Deduktionssysteme, Automatisierung des Logischen Denkens, second ed., R. Oldenbourg Verlag, 1992, also in English with Elsewood.
[87] J. Siekmann, History of computational logic, in: D. Gabbay, J. Woods (Eds.), The Handbook of the History of Logic, vols. I–IX, Elsevier, Amsterdam, 2004.
[88] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, M. Pollet, Proof development in OMEGA: The irrationality of square root of 2, in: F. Kamareddine (Ed.), Thirty Five Years of Automating Mathematics, in: Kluwer Applied Logic Series, vol. 28, Kluwer Academic Publishers, Dordrecht, ISBN 1-4020-1656-5, 2003, pp. 271–314.
[89] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, M. Pollet, Proof development with OMEGA: Sqrt(2) is irrational, in: M. Baaz, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002, in: Lecture Notes in Artificial Intelligence, vol. 2514, Springer, Berlin, 2002, pp. 367–387.
[90] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, V. Sorge, $\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface, Formal Aspects of Computing 11 (1999) 326–342.
[91] V. Sorge, Non-trivial computations in proof planning, in: [55].
[92] V. Sorge, *ΩANTS*—a blackboard architecture for the integration of reasoning techniques into proof planning, PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.
[93] G. Sutcliffe, C. Suttner, T. Yemenis, The TPTP problem library, in: [27].

S1570-8683(05)00073-X/FLA AID:88 Vol.●●●(●●●) jal88 [DTD5] P.27 (1-27)
JAL:m1a v 1.50 Prn:17/11/2005; 11:16 by:Jolanta p. 27

[94] J. van der Hoeven, GNU TeXmacs: A free, structured, wysiwyg and technical text editor, in: Actes du congrès Gutenberg, Metz, Actes du congrès Gutenberg, vols. 39–40, May 2001, pp. 39–50.
[95] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, Th. Engel, E. Keen, C. Theobalt, D. Topic, System description: SPASS version 1.0.0, in: [44], pp. 378–382.
[96] D. Weld, An introduction to least commitment planning, AI Magazine 15 (4) (1994) 27–61.
[97] F. Wiedijk, The Seventeen Provers of the World, in: Lecture Notes in Artificial Intelligence, Springer, Berlin, 2005, in press.
[98] J. Zhang, H. Zhang, SEM: A system for enumerating models, in: C.S. Mellish (Ed.), Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Montreal, Canada, Morgan Kaufmann, San Mateo, CA, 1995, pp. 298–303.
[99] J. Zimmer, M. Kohlhase, System description: The Mathweb software bus for distributed mathematical reasoning, in: A. Voronkov (Ed.), Proceedings of the 18th International Conference on Automated Deduction, in: Lecture Notes in Artificial Intelligence, vol. 2392, Springer, Berlin, 2002, pp. 138–142.
[100] J. Zimmer, E. Melis, Constraint solving for proof planning, J. Automat. Reason. 33 (2004) 51–88.
[101] J. Zimmer, A framework for agent-based brokering of reasoning services, in: R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, J.H.S. Azuela (Eds.), MICAI, in: Lecture Notes in Computer Science, vol. 2972, Springer, Berlin, 2004.

JÖRG SIEKMANN, CHRISTOPH BENZMÜLLER, ARMIN FIEDLER, ANDREAS MEIER, IMMANUEL NORMANN, AND MARTIN POLLET

PROOF DEVELOPMENT WITH ΩMEGA: THE IRRATIONALITY OF $\sqrt{2}$

The well-known theorem asserting the irrationality of $\sqrt{2}$ was proposed as a case study for a comparison of fifteen (interactive) theorem proving systems [Wiedijk, 2002]. This represents an important shift of emphasis in the field of automated deduction away from the somehow artificial problems of the past back to real mathematical challenges.

We present an overview of the ΩMEGA system as far as it is relevant for the purpose of this paper, and then we discuss three different styles of proof development in ΩMEGA using the example of the irrationality of $\sqrt{2}$: The first follows the traditional tactical theorem proving approach without any mathematical knowledge, the second employs the idea of interactive island proof planning, and the third is a fully automated proof based on planning with ΩMEGA's proof planner MULTI. Moreover, we illustrate the expansion and subsequent verification of the proofs at the logic level. We also discuss the knowledge engineering process by which the main ideas of the interactive island proof are generalized into respective proof methods, such that automatic proof planning becomes feasible for this domain.

1 ΩMEGA

The ΩMEGA proof development system [Siekmann *et al.*, 2002] is at the core of several related and well-integrated research projects of the ΩMEGA research group, whose aim is to develop system support for the working mathematician.

ΩMEGA is a mathematical assistant tool that supports proof development in mathematical domains at a user-friendly level of abstraction. It is a modular system with a central proof data structure and several supplementary subsystems. ΩMEGA has many characteristics in common with systems like NUPRL [Allen *et al.*, 2000], COQ [Coq Development Team, 1999-2003], HOL [Gordon and Melham, 1993], PVS [Owre *et al.*, 1996], and Isabelle [Paulson, 1994; Nipkow *et al.*, 2002]. However, it differs significantly from these systems with respect to its focus on *proof planning* and in that respect it is more similar to the proof planning systems CLAM and

Fairouz Kamareddine (ed.),
Thirty Five Years of Automating Mathematics *271–314.*

λCLAM at Edinburgh [Richardson *et al.*, 1998; Bundy *et al.*, 1990]. We shall now present an overview of the architecture of the ΩMEGA system and show some of its novel features, which include facilities to access several external reasoning systems and to integrate their results into a single proof structure; substantial support for interactive proof development through some non-standard inspection facilities and for guidance in the search for a proof; and finally methods to develop proofs at a human-oriented, higher level of abstraction.

1.1 System Overview

The ΩMEGA project currently represents one of the largest attempts worldwide to build an assistant tool for the working mathematician. It is a representative of systems in the new paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The inference mechanism at the lowest level of abstraction is an interactive theorem prover based on a higher-order natural deduction (ND) variant of a soft-sorted[1] version of Church's simply typed λ-calculus [Church, 1940]. The logical language, which also supports partial functions, is called $\mathcal{POST}$, for ***p***artial functions ***o***rder ***s***orted ***t***ype theory. While this represents the "machine code" of the system the user will seldom want to see, the search for a proof is usually conducted at a higher level of abstraction defined by *tactics* and *methods*. Automated proof search at this abstract level is called *proof planning* (see Section 1.3). Proof construction is also supported by already proven assertions and theorems and by calls to external systems to simplify or solve subproblems, as will be shown in Section 1.2.

At the core of ΩMEGA is the *proof plan data structure* $\mathcal{PDS}$ [Cheikhrouhou and Sorge, 2000], in which proofs and *proof plans* are represented at various levels of granularity and abstraction. The $\mathcal{PDS}$ is a directed acyclic graph, where *open nodes* represent unjustified propositions that still need to be proved and *closed nodes* represent propositions that are already proved. The proof plans are developed and classified with respect to a taxonomy of mathematical theories, which is currently being replaced by the mathematical knowledge base MBASE [Franke and Kohlhase, 2000; Kohlhase and Franke, 2001]. The user of ΩMEGA, or the proof planner MULTI [Melis and Meier, 2000], or else the suggestion mechanism Ω-ANTS [Benzmüller and Sorge, 2000] modify the $\mathcal{PDS}$ during proof development until a complete proof plan has been found. They can also invoke external reasoning systems, whose results are included in the $\mathcal{PDS}$ after appropriate transformation. Once a complete proof plan at the most appropriate level of abstraction has been found, this plan must be expanded by sub-methods

[1] Unary predicates are interpreted as sorts and theorems of a certain syntactical form as sort declarations. Sort inferences using this information are explicit proof steps.

and sub-tactics into lower levels of abstraction until finally a proof at the level of the logical calculus is established. After expansion of these high-level proofs to the underlying ND calculus, the $\mathcal{PDS}$ can be checked by ΩMEGA's proof checker.

Hence, there are two main tasks supported by this system, namely (i) to find a proof plan, and (ii) to expand this proof plan into a calculus-level proof; and both jobs can be equally difficult and time consuming. Task (ii) employs an LCF-style tactic expansion mechanism, proof search or a combination of both in order to generate a lower-level proof object. It is a design objective of the $\mathcal{PDS}$ that various *proof levels* coexist with their respective relationships being dynamically maintained. Failing expansions of proof steps typically lead to open nodes at a lower proof level and thus result in an incomplete proof. The reasons for the failure can in principle be analyzed in the $\mathcal{PDS}$ in the sense of proof critics [Ireland and Bundy, 1996], however, this option has not yet been further explored.

User interaction is supported by the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [Siekmann *et al.*, 1999], which provides both a graphical and a tabular view of the proof under consideration, and the interactive proof explanation system *P.rex* [Fiedler, 2001a; Fiedler, 2001b], which provides the user with a natural-language presentation of the proof.

The previously monolithic system has been split up and separated into several independent modules, which are connected via the mathematical software bus MATHWEB-SB [Zimmer and Kohlhase, 2002]. An important benefit is that MATHWEB modules can be distributed over the Internet and are then remotely accessible by other research groups as well. There is a very active user community with sometimes several thousand theorems and lemmata being proved per day; most theorems are generated automatically as (currently non-reusable and non-indexed) subproblems in natural language processing (see the Doris system [Doris, 2001]), proof planning [Meier *et al.*, 2002b; Sorge, 2001], and verification tasks.

1.2 External Systems

Proof problems require many different skills for their solution. Therefore, it is desirable to have access to several systems with complementary capabilities, to orchestrate their use, and to integrate their results. ΩMEGA interfaces heterogeneous external systems such as *computer algebra systems (CASs)*, higher- and first-order *automated theorem proving systems (ATPs)*, *constraint solvers (CSs)*, and *model generation systems (MGs)*.

Their use is twofold: they may provide a solution to a subproblem, or they may give hints for the control of the search for a proof. In the former case, the output of an incorporated reasoning system is translated and inserted as a subproof into the $\mathcal{PDS}$. This is beneficial for interfacing systems that operate at different levels of abstraction, and also for a human-oriented

display and inspection of a partial proof. Importantly, it also enables us to check the soundness of each contribution by refining the inserted subproof to a logic-level proof to be verified by ΩMEGA's proof checker.

Currently, the following external systems are integrated in ΩMEGA:

CASs provide symbolic computation, which can be used in two ways: first, to compute hints to guide the proof search (e.g., witnesses for existential variables), and, second, to perform some complex algebraic computation such as to normalize or simplify terms. In the latter case the symbolic computation is directly translated into proof steps in ΩMEGA. CASs are integrated via the transformation and translation module SAPPER [Sorge, 2000]. Currently, ΩMEGA uses the systems MAPLE [Char *et al.*, 1992] and GAP [Schönert and others, 1995].

ATPs are employed to solve subgoals. Currently ΩMEGA uses the first-order provers BLIKSEM [de Nivelle, 1999], EQP [McCune, 1997], OTTER [McCune, 1994], PROTEIN [Baumgartner and Furbach, 1994], SPASS [Weidenbach *et al.*, 1999], WALDMEISTER [Hillenbrand *et al.*, 1999], the higher-order systems TPS [Andrews *et al.*, 1996], and $\mathcal{LEO}$ [Benzmüller and Kohlhase, 1998; Benzmüller, 1999], and we plan to incorporate VAMPIRE [Riazanov and Voronkov, 2001]. The first-order ATPs are connected via TRAMP [Meier, 2000], which is a proof transformation system that transforms resolution-style proofs into assertion-level ND proofs to be integrated into ΩMEGA's $\mathcal{PDS}$. TPS already provides ND proofs, which can be further processed and checked with little transformational effort [Benzmüller *et al.*, 1999].

MGs provide either witnesses for free (existential) variables, or countermodels, which show that some subgoal is not a theorem. Hence, they help to guide the proof search. Currently, ΩMEGA uses the MGs SATCHMO [Manthey and Bry, 1988] and SEM [Zhang and Zhang, 1995].

CSs construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a CS can help to reduce the proof search by checking for inconsistencies of constraints. Currently, ΩMEGA employs $\mathcal{CoSIE}$ [Melis *et al.*, 2000], a CS for inequalities and equations over the field of real numbers.

1.3 Proof Planning

ΩMEGA's main focus is on knowledge-based proof planning [Bundy, 1988; Melis and Siekmann, 1999], where proofs are not conceived in terms of low-level calculus rules, but at a much higher level of abstraction that highlights

the main ideas and de-emphasizes minor logical or mathematical manipulations on formulae.

Knowledge-based proof planning is a new paradigm in automated theorem proving, which swings the motivational pendulum back to its AI origins in that it employs and further develops many AI principles and techniques such as hierarchical planning, knowledge representation in frames and control rules, constraint solving, tactical theorem proving, and meta-level reasoning. It differs from traditional search-based techniques in automated theorem proving not least in its level of abstraction: the proof of a theorem is planned at an abstract level where an outline of the proof is found. This outline, that is, the abstract proof plan, can be recursively expanded to construct a proof within a logical calculus provided the proof plan does not fail. The plan operators represent mathematical techniques familiar to a working mathematician. While the knowledge of such a mathematical domain as represented within methods and control rules is specific to the mathematical field, the representational techniques and reasoning procedures are general-purpose. For example, one of our first case studies [Melis and Siekmann, 1999] used the limit theorems proposed by Woody Bledsoe [Bledsoe, 1990] as a challenge to automated reasoning systems. The general-purpose planner makes use of this mathematical domain knowledge and of the guidance provided by declaratively represented control rules, which correspond to mathematical intuition about how to prove a theorem in a particular situation. These rules provide a basis for meta-level reasoning and goal-directed behavior.

In ΩMEGA, domain knowledge is encoded in methods, control rules, and strategies. Moreover, methods and control rules can employ external systems (e.g., computer algebra systems) and make use of the knowledge in these systems. ΩMEGA's multi-strategy proof planner MULTI [Melis and Meier, 2000] searches then for a plan using the acquired methods and strategies guided by the control knowledge in the control rules.

AI Principles in Proof Planning

In AI, a *planning problem* is a formal description of an *initial state*, a *goal*, and some *operators* that can be used to transform the initial state via some intermediate states to a state that satisfies the goal. Applied to a planning problem, a *planner* returns a sequence of *actions*, that is, instantiated operators, which reach a goal state from the initial state when executed. Such a sequence of actions is also called a *solution plan.*

A simple, yet very influential language is the STRIPS representation [Fikes and Nilsson, 1971; Fikes *et al.*, 1972]. Formalized in propositional logic, STRIPS describes the initial state by a set of ground literals. A goal is described by a conjunction of positive literals. Operators in STRIPS have *preconditions* and *effects*, which formalize to which states the operator can

be applied and how these states are changed by its application, respectively. Whereas the preconditions of an operator are represented by a conjunction of positive literals, the effects are represented by a conjunction of positive and negative literals. The positive literals in the operator's effects are called the *add list* of the operator, while the negative literals are called the *delete list* of the operator.

The classical approach to planning problems is *precondition achievement planning* [Drummond, 1994], which goes back to the General Problem Solver, GPS [Newell and Simon, 1963]. The planning process starts from the goal, which is considered as an unachieved precondition. First, the add list of all operators is checked to see whether it contains an effect that achieves some literal of the goal. Then, one such operator is chosen and appropriately instantiated, and the resulting action is inserted into the plan under development, thus satisfying part of the goal. The preconditions of the introduced action become new unsatisfied preconditions of the plan, and the planning process recurses on those.

Proof planning considers mathematical theorems as planning problems [Bundy, 1988]. The initial state of a proof planning problem consists of the proof *assumptions* of the theorem, whereas the goal is the *theorem* itself. The operators in proof planning are the methods.

In ΩMEGA, proof planning is the process that computes actions, that is, instantiations of methods, and assembles them in order to derive a theorem from a set of assumptions. The effects and the preconditions of an action in proof planning are proof nodes with formulae in the higher-order language $\mathcal{POST}$, where the effects are considered as logically inferable from the preconditions. A proof plan under construction is represented in the proof plan data structure $\mathcal{PDS}$ (see Section 1.5). Initially, the $\mathcal{PDS}$ consists of an open node containing the statement to be proved, and closed, that is, justified, nodes for the proof assumptions. The introduction of an action changes the $\mathcal{PDS}$ by adding new proof nodes and justifying the effects of the action by applications of the method of the action to its premises. The aim of the proof planning process is to reach a *closed* $\mathcal{PDS}$, that is, a $\mathcal{PDS}$ without open nodes. The *solution proof plan* produced is then a record of the sequence of actions that lead to a closed $\mathcal{PDS}$.

By allowing for forward and backward actions ΩMEGA's proof planning combines forward and backward state-space planning. Thus, a *planning state* is a pair of the current world state and the current goal state. The initial world state consists of the given proof assumptions and is transfered by forward actions into a new world state. The goal state consists of the initial open node and is transfered by backward actions into a new goal state containing new open nodes. From this point of view the aim of proof planning is to compute a sequence of actions that derives a current world state in which all the goals in the current goal state are satisfied.

As opposed to precondition achievement planning (e.g., see [Weld, 1994]),

effects of methods in proof planning do not cancel each other. For instance, an action with effect $\neg F$ introduced for the open node L_1 does not threaten the effect F introduced by another action for the open node L_2. Dependencies among open nodes result from shared variables for witness terms and their constraints. Constraints can be, for instance, instantiations for the variables but they can also be mathematical constraints such as $x < c$, which states that, whatever the instantiation for x is, it has to be smaller than c. The constraints created during the proof planning process are collected in a constraint store. An action introducing new constraints is applicable only if its constraints are consistent with the constraints collected so far. Dependencies among goals with shared variables are difficult to analyze and can cause various kinds of failures in a proof planning attempt. First results about how to analyze and deal with such failures are discussed in [Meier, 2003].

Methods, Control Rules, and Strategies

In ΩMEGA, *methods* can be perceived as tactics in tactical theorem proving augmented with preconditions and effects, called *premises* and *conclusions*, respectively. A method represents the inference of the conclusion from the premises. For instance, NotI-M is a method whose purpose is to prove a goal $\Gamma \vdash \neg P$ by contradiction. If NotI-M is applied to a goal $\Gamma \vdash \neg P$ then it closes this goal and introduces the new goal to prove falsity, $\bot$, under the assumption P, that is, $\Gamma, P \vdash \bot$. Thereby, $\Gamma \vdash \neg P$ is the conclusion of the method, whereas $\Gamma, P \vdash \bot$ is the premise of the method. NotI-M is a *backward* method, which reduces a goal (the conclusion) to new goals (the premises). *Forward* methods, in contrast, derive new conclusions from given premises. For instance, =Subst-m performs equality substitutions by deriving from two premises $\Gamma \vdash P[a]$ and $\Gamma \vdash a = b$ the conclusion $\Gamma \vdash P[b]$ where an occurrence of a is replaced by an occurrence of b. Note that NotI-M and =Subst-m are simple examples of domain-independent, logic-related methods, which are needed in addition to domain-specific, mathematically motivated methods. Examples of the latter will be discussed in detail in Section 3.

Control rules represent mathematical knowledge about how to proceed in the proof planning process. They can influence the planner's behavior at choice points (e.g., which goal to tackle next or which method to apply next) by preferring members of the corresponding list of alternatives (e.g., the list of possible goals or the list of possible methods). This way promising search paths are preferred and the search space can be pruned. We shall discuss examples for control rules in Section 3.1.

Strategies employ different sets of methods and control rules and, thus, tackle the same problem in different ways. The reasoning as to which strategy to employ on a problem is an explicit choice point in MULTI. In partic-

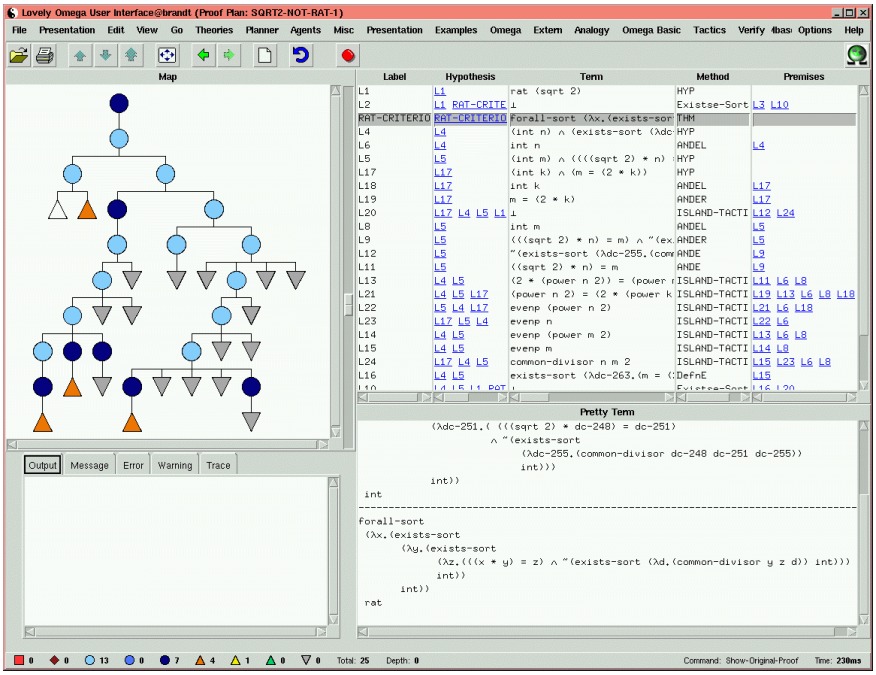


Figure 1. Multi-modal proof presentation in the graphical user interface $\mathcal{LOUI}$.

ular, MULTI can backtrack from chosen strategies and search at the level of strategies.

Detailed discussions of ΩMEGA's method and control rule language can be found in [Meier *et al.*, 2002a]. A detailed introduction to proof planning with multiple strategies is given in [Melis and Meier, 2000].

1.4 Interface and System Support

ΩMEGA's graphical user interface $\mathcal{LOUI}$ [Siekmann *et al.*, 1999] displays the current proof state in multiple modalities: a graphical map of the proof tree, a linearized presentation of the proof nodes with their formulae and justifications, a term browser, and a natural language presentation of the proof via *P.rex* (see Fig. 1 and 2).

When inspecting portions of a proof by these facilities, the user can switch between alternative levels of abstraction, for example, by expanding a node in the graphical map of the proof tree, which causes appropriate changes in the other presentation modes. Moreover, an interactive natural language *explanation* of the proof is provided by the system *P.rex* [Fiedler, 2001a;

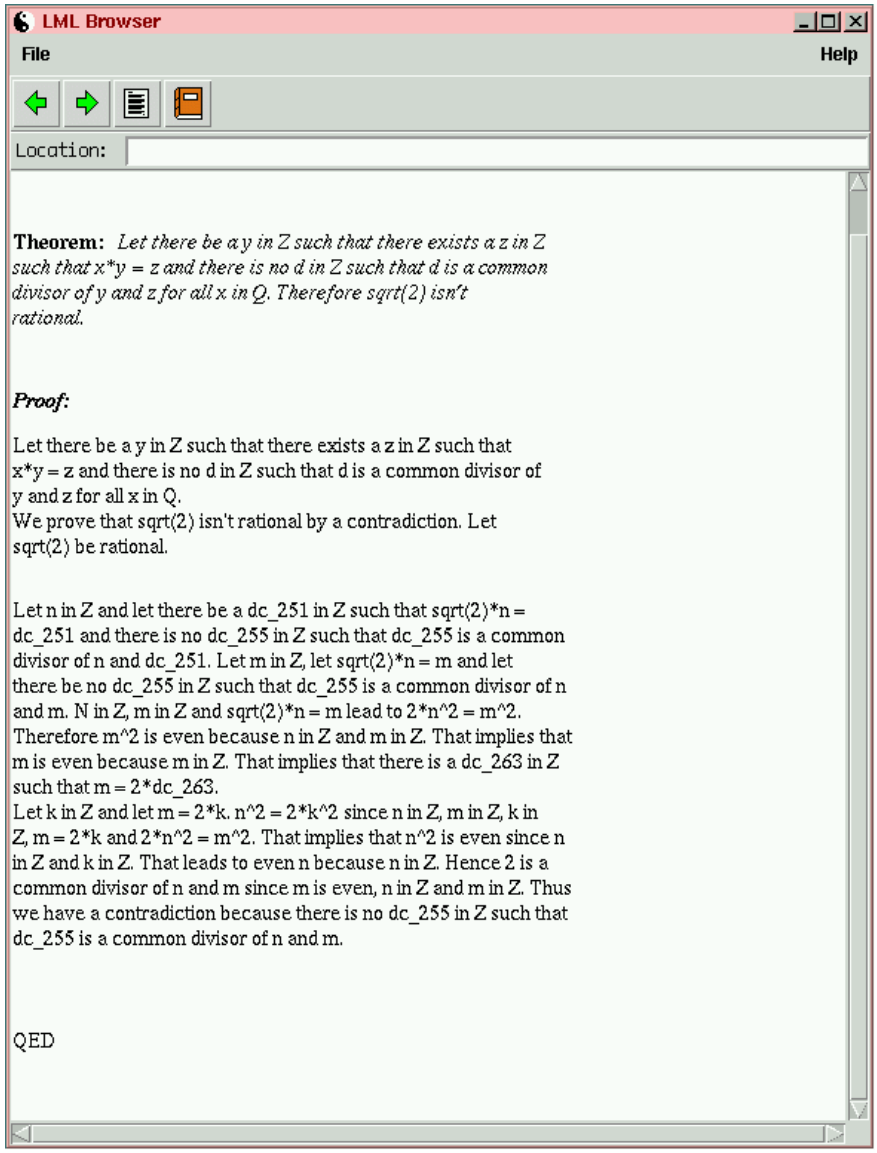


Figure 2. Natural language proof presentation by *P.rex* in $\mathcal{LOUI}$.

Fiedler, 2001b; Fiedler, 2001c], which is adaptive in the following sense: it explains a proof step at the most abstract level (which the user is assumed to know) and then reacts flexibly to questions and requests, possibly at lower levels of abstractions, for example, by detailing some ill-understood subproof.

Another system support feature of ΩMEGA is the guidance mechanism provided by the suggestion module Ω-ANTS [Benzmüller and Sorge, 1998; Benzmüller and Sorge, 2000; Sorge, 2001], which searches proactively for possible actions that may be helpful in finding a proof and orders them in a preference list. Examples for such actions are an application of a particular calculus rule, the call of a tactic or a proof method as well as a call of an external reasoning system, or the search for and insertion of facts from the knowledge base MBASE. The general idea is the following: every inference rule, tactic, method or external system is "agentified" in the sense that every possible *action* searches concurrently for the fulfillment of its application conditions and once these are satisfied it suggests its execution. User-definable heuristics select and display the suggestions to the user. Ω-ANTS is based on a hierarchical blackboard, which collects the data about the current proof state.

1.5 Proof Objects

The central data structure for the overall search is the proof plan data structure $\mathcal{PDS}$. This is a hierarchical data structure that represents a (partial) proof at different levels of abstraction (called partial proof plans). Technically, it is an acyclic graph, where the nodes are justified by tactic applications. Conceptually, each such justification represents a proof plan (the expansion of the justification) at a lower level of abstraction, which is computed when the tactic is executed. In ΩMEGA, we explicitly keep the original proof plan as well as intermediate expansion layers in an expansion hierarchy. The coexistence of several abstraction levels and the dynamical maintenance of their relationship is a central design objective of ΩMEGA's $\mathcal{PDS}$. Thus the $\mathcal{PDS}$ makes the hierarchical structure of proof plans explicit and retains it for further applications such as proof explanation with *P.rex* or analogical transfer of plans. The lowest level of abstraction of a $\mathcal{PDS}$ represents the ND calculus.

The proof object generated by ΩMEGA for the "irrationality of $\sqrt{2}$" theorem is recorded in a technical report [Benzmüller *et al.*, 2002], where the unexpanded and the expanded proof objects are presented in great detail, that is in a little less than a thousand proof steps.

1.6 Case Studies

Early developments of proof planning in Alan Bundy's group at Edinburgh used proofs by induction as their favorite case studies [Bundy, 1988]. The ΩMEGA system has been used in several other case studies, which illustrate in particular the interplay of the various components, such as proof planning supported by heterogeneous external reasoning systems.

A typical example for a class of problems that cannot be solved by traditional automated theorem provers is the class of ϵ–δ–proofs [Melis and Siekmann, 1999; Melis, 1998]. This class was originally proposed by Woody Bledsoe [Bledsoe, 1990] and it comprises theorems such as LIM+ and LIM*, where LIM+ states that the limit of the sum of two functions equals the sum of their limits and LIM* makes the corresponding statement for multiplication. The difficulty of this domain arises from the need for arithmetic computation in order to find a suitable instantiation of free (existential) variables (such as a δ depending on an ϵ). Crucial for the success of ΩMEGA's proof planning is the integration of suitable experts for these tasks: the arithmetic computation is done by the computer algebra system MAPLE, and an appropriate instantiation for δ is computed by the constraint solver $\mathcal{CoSIE}$. We have been able to solve all challenge problems suggested by Bledsoe and many more theorems in this class taken from a standard textbook on real analysis [Bartle and Sherbert, 1982].

Another class of problems we tackled with proof planning is concerned with residue classes [Meier *et al.*, 2002b; Meier *et al.*, 2001]. In this domain we show theorems such as: "the residue class structure $(\mathbb{Z}_5, \bar{+})$ is associative", "it has a unit element", and similar properties, where $\mathbb{Z}_5$ is the set of all congruence classes modulo 5 $\{\bar{0}_5, \bar{1}_5, \bar{2}_5, \bar{3}_5, \bar{4}_5\}$ and $\bar{+}$ is the addition on residue classes. We have also investigated whether two given structures are isomorphic or not and altogether we have proved more than 10,000 theorems of this kind (see [Sorge, 2001]). Although the problems in this domain are still within the range of difficulty a traditional automated theorem prover can handle, it was nevertheless an interesting case study for proof planning, since multi-strategy proof planning generated substantially different proofs based on entirely different proof ideas.

Another important proof technique is Cantor's diagonalization technique and we also developed methods and strategies for this class [Cheikhrouhou and Siekmann, 1998]. Important theorems we have been able to prove are the undecidability of the halting problem and Cantor's theorem (cardinality of the set of subsets), the non-countability of the reals in the interval $[0, 1]$ and of the set of total functions, and similar theorems.

Finally, a good candidate for a standard proof technique are completeness proofs for refinements of resolution, where the theorem is usually first shown at the ground level using the excess-literal-number technique and then ground completeness is lifted to the predicate calculus. We have done

this for many refinements of resolution with ΩMEGA [Gebhard, 1999].

2 A CASE STUDY: $\sqrt{2}$ IS NOT RATIONAL

ΩMEGA's main aim is to become a proof assistant tool for the working mathematician. Hence, it should support interactive proof development at a user-friendly level of abstraction. The mathematical theorem that $\sqrt{2}$ is not rational, and its well-known proof dating back to the School of Pythagoras, provides an excellent challenge to evaluate whether this ambitious goal has been reached. In the remainder of the paper, we will refer to this proof problem as the $\sqrt{2}$-*problem*. In [Wiedijk, 2002] fifteen systems that have solved the $\sqrt{2}$-problem show their respective results. The protocols of their respective sessions have been compared on a multi-dimensional scale in order to assess the "naturalness" by which real mathematical problems of this kind can be proved within the respective system.

This represents an important shift of emphasis in the field of automated deduction away from the somehow artificial problems of the past — as represented, for example, in the test set of the TPTP library [Sutcliffe *et al.*, 1994] — back to real mathematical challenges.

We participated in this case study essentially with three different contributions. Our initial contribution was an interactive proof in ΩMEGA without adding special domain knowledge to the system. For further details on this case study, which particularly demonstrates the use of ΩMEGA as a usual tactical theorem prover, we refer to [Benzmüller *et al.*, 2002]. The most important albeit not entirely new lesson to be learned from this experiment is that the level of abstraction common in most automated and tactical theorem proving environments is far too low. While our proof representation is already an abstraction (called the *assertion level* in [Huang, 1994]) from the calculus level typical for most ATPs, it is nevertheless clear that as long as a system does not hide all these excruciating details, no working mathematician will feel inclined to use such a system. In fact, this is in our opinion one of the critical impediments for using ATPs and one, albeit not the only one, of the reasons why they are not used as widely as, say, computer algebra systems.

This is the crucial issue in the ΩMEGA project and our main motivation for departing from the classical paradigm of automated theorem proving about fifteen years ago.

Our second contribution to the case study of the $\sqrt{2}$-problem is based on interactive *island planning* [Melis, 1996], a technique that expects an outline of the proof and has the user provide main subgoals, called *islands*, together with their assumptions. The details of the proof, eventually down to the logic level, are postponed. Hence, the user can write down *his* proof idea in a natural way with as many gaps as there are open at this first stage

of the proof. Closing the gaps is ideally fully automatic, in particular, by exploiting the external systems interfaced to ΩMEGA. However, for difficult theorems it is necessary more often than not that the user provides additional information and applies the island approach recursively.

In comparison to our first tactic-based solution the island style supports a much more abstract and user-friendly interaction level. The proofs are now at a level of abstraction similar to proofs in mathematical textbooks.

Our third contribution to the case study of the $\sqrt{2}$-problem is a fully automatically planned and expanded proof of the theorem as presented in Section 3.

In the following sections we shall first describe the problem formalization (Section 2.1). Then, we shall present part of the tactic-level proof (Section 2.2) and the interactive island approach (Sections 2.3 to 2.5). Finally, in Section 3 we show the fully automated solution and we provide a generalization of it covering a whole class of similar problems. The actual challenge, attributed to the Pythagorean School, is as follows:

THEOREM. $\sqrt{2}$ is irrational.

Proof. [by contradiction] Assume $\sqrt{2}$ is rational, that is, there exist natural numbers m, n with no common divisor such that $\sqrt{2} = m/n$. Then $n\sqrt{2} = m$, and thus $2n^2 = m^2$. Hence m^2 is even and, since odd numbers square to odds, m is even; say $m = 2k$. Then $2n^2 = (2k)^2 = 4k^2$, that is, $n^2 = 2k^2$. Thus, n^2 is even too, and so is n. That means that both n and m are even, contradicting the fact that they do not have a common divisor. ■

2.1 Problem Formalization

The theorem is initially formulated in ΩMEGA's knowledge base as an open problem in the theory `REAL`. The problem is encoded in $\mathcal{POST}$ syntax, which is the logical input language for ΩMEGA:

```
(th~defproblem sqrt2-not-rat (in real)
  (conclusion (not (rat (sqrt 2))))
  (help "sqrt 2 is not a rational number."))
```

The concepts of the rational numbers (`rat`) and the square root (`sqrt`) are defined in the knowledge base as well. Since they are not needed in the interactive session at this abstract level and because of lack of space, we do not display them here (cf. [Benzmüller *et al.*, 2002] for the details).

To prove the given problem, further mathematical knowledge is required. Our proof employs the definition of evenness (`evenp`) and some theorems about rational numbers, evenness, and common divisors (`common-divisor`). However, the definition of `sqrt` is not needed in the main proof, because we use the computer algebra system MAPLE to justify the transformation

of $n\sqrt{2} = m$ into $2n^2 = m^2$. To do so, expressions in ΩMEGA such as $\sqrt{2}$ are mapped to corresponding MAPLE representations, and MAPLE uses its own built-in knowledge to manipulate them. Using and verifying these computation steps requires the expansion of MAPLE's computations to the calculus layer in ΩMEGA. As shown in [Sorge, 2000], this can be done by replaying MAPLE's computation by special computational tactics in ΩMEGA, which may also unfold some definitions such as `sqrt`. These tactics and their expansions are part of the SAPPER system and correspond directly to the mathematical definitions available in ΩMEGA's knowledge base. For example, the number 2 is defined in theory `NATURAL` as $s(s(0))$. Again, this knowledge is only required when expanding the abstract proof to the basic calculus layer and it is not visible to the user at this stage. However, SAPPER is still too weak in general, this is a subject of further development.

We now give examples for definitions and theorems used in our proofs of the $\sqrt{2}$-problem.

```
(th~defdef evenp (in integer)
  (definition
     (lam (x  num) (exists-sort (lam (y num) (= x (times 2 y))) int)))
  (help "Definition of even."))



(th~deftheorem rat-criterion (in real)
  (conclusion (forall-sort (lam (x num)
                (exists-sort (lam (y num)
                  (exists-sort (lam (z num)
                    (and (= (times x y) z)
                         (not (exists-sort (lam (d num)
                                                (common-divisor y z d))
                                int))))
                     int))
                  int))
                rat))
  (help "x rational implies there exist integers y,z which
         have no common divisor and furthermore z=x*y."))



(th~deftheorem square-even (in integer)
  (conclusion (forall-sort (lam (x num) (equiv (evenp (power x 2))
                                               (evenp x)))
              int))
 (help "x is even, iff x^2 is even."))
```

2.2 Tactical Theorem Proving in ΩMEGA

One way to construct proofs in ΩMEGA interactively is based on traditional tactical theorem proving. When employed in this mode ΩMEGA is comparable to many other interactive systems like NUPRL [Allen *et al.*, 2000], COQ [Coq Development Team, 1999-2003], HOL [Gordon and Melham, 1993], PVS [Owre *et al.*, 1996], and Isabelle [Paulson, 1994; Nipkow *et al.*, 2002]. However, a characteristic special to ΩMEGA is that several tools

(e.g., the various external systems) are available to support the interactive proof development process (cf. Section 1.2).

Our first case study on the $\sqrt{2}$-problem used the tactical theorem proving approach and proved the theorem in 33 interactive steps. These steps were automatically recorded by ΩMEGA in a so-called replay file, which contains all information that is needed to automatically replay a proof.[2] Here, we only sketch the interactions required for the problem by presenting the content of this replay file. A detailed description of the interactive proof can be found in [Benzmüller *et al.*, 2002].

```
Step 0:   OMEGA-BASIC PROVE (SQRT2-NOT-RAT)
Step 1:   DECLARATION DECLARE ((CONSTANTS (M NUM) (N NUM) (K NUM)))
Step 2:   RULES NOTI default default
Step 3:   MBASE IMPORT-ASS (RAT-CRITERION)
Step 4:   TACTICS FORALLE-SORT default default ((SQRT 2)) default
Step 5:   TACTICS EXISTSE-SORT default default (N) default
Step 6:   TACTICS ANDE default default default
Step 7:   TACTICS EXISTSE-SORT (L7) default (M) default
Step 8:   TACTICS ANDE* (L8) (NIL)
Step 9:   OMEGA-BASIC LEMMA default ((= (POWER M 2) (TIMES 2 (POWER N 2))))
Step 10:  TACTICS BY-COMPUTATION (L13) ((L11))
Step 11:  OMEGA-BASIC LEMMA (L9) ((EVENP (POWER M 2)))
Step 12:  RULES DEFN-CONTRACT default default default
Step 13:  OMEGA-BASIC LEMMA (L9) ((INT (POWER N 2)))
Step 14:  TACTICS WELLSORTED default default
Step 15:  TACTICS EXISTSI-SORT (L15) ((POWER N 2)) (L13) (L16) default
Step 16:  MBASE IMPORT-ASS (SQUARE-EVEN)
Step 17:  TACTICS ASSERT ((EVENP M)) ((SQUARE-EVEN L10 L14)) (NIL)
Step 18:  RULES DEFN-EXPAND (L17) default default
Step 19:  TACTICS EXISTSE-SORT default default (K) default
Step 20:  TACTICS ANDE (L19) default default
Step 21:  OMEGA-BASIC LEMMA default ((= (POWER N 2) (TIMES 2 (POWER K 2))))
Step 22:  TACTICS BY-COMPUTATION (L23) ((L13 L22))
Step 23:  OMEGA-BASIC LEMMA default ((EVENP (POWER N 2)))
Step 24:  RULES DEFN-CONTRACT default default default
Step 25:  OMEGA-BASIC LEMMA (L20) ((INT (POWER K 2)))
Step 26:  TACTICS WELLSORTED (L26) ((L21))
Step 27:  TACTICS EXISTSI-SORT default ((POWER K 2)) (L23) default default
Step 28:  TACTICS ASSERT ((EVENP N)) ((SQUARE-EVEN L6 L24)) (NIL)
Step 29:  MBASE IMPORT-ASS (EVEN-COMMON-DIVISOR)
Step 30:  OMEGA-BASIC LEMMA (L20) ((INT 2))
Step 31:  TACTICS WELLSORTED (L28) (NIL)
Step 32:  TACTICS ASSERT (FALSE) ((EVEN-COMMON-DIVISOR L10 L6 L12 L17 L27 L28)) (NIL)
Step 33:  RULES WEAKEN default default
```

The lesson to be learned from this protocol is that the wrong level of abstraction is still common in most automated and tactical theorem proving environments. This is our conviction even though De Bruijn's conjecture that the formalized proof object is at most a linear blow-up of the informal mathematical proof can be argued to actually hold for this example.

In the following section, we shall show how a proof at a more user-friendly level of abstraction can be achieved.

[2]The structure of a line entry of a replay file is as follows: First an identifier of the command category (e.g., `MBASE`) is given. Then follows the command to be executed (e.g., `IMPORT-ASS`) and a sequence of parameters (e.g., `RAT-CRITERION`). The parameter information "`default`" leaves the choice of the parameter to ΩMEGA.

2.3 Interactive Island Proof Development in ΩMEGA

The $\sqrt{2}$-problem can also be solved interactively in ΩMEGA along the lines of the previously given textbook proof (cf. the introduction of Section 2). Due to space restrictions we cannot show the proof development using ΩMEGA's graphical user interface $\mathcal{LOUI}$, but the more cumbersome command line interface of the emacs editor. Otherwise we would have to show a $\mathcal{LOUI}$ screen shot for every user interaction. Thus, the following presentation gives an insufficient impression of the interaction with ΩMEGA, which is in the style of the final island proof plan in $\mathcal{LOUI}$ and *P.rex* as shown in Fig. 1 and 2.

For every command we show both the newly introduced proof nodes and the previously open proof nodes that are closed by the command. The input to ΩMEGA (entered after the `OMEGA` prompt) and its output are given in `typewriter` font.

We present the steps of the proof in a linearized style and therefore call proof nodes also *proof lines*. In the following, we shall write proof lines as $L \quad (\Delta) \vdash \varphi \quad \mathcal{R}$, where L is a unique label, $(\Delta) \vdash \varphi$ denotes that the formula φ can be derived from the formulae whose labels are in the list Δ, and $\mathcal{R}$ is the justification for this derivation of φ from Δ by naming the used inference rule, tactic, or method along with parameters and premises.

Step 0 We start by loading the theory `REAL`, in which the problem is declared.

```
OMEGA: load-problems real
;;; Rules loaded for theory REAL.
;;; Theorems loaded for theory REAL.
;;; Tactics loaded for theory REAL.
;;; Methods loaded for theory REAL.
;;; Strategies loaded for theory REAL.
```

Now, we set the focus on our problem and declare some constant symbols, which we shall use later.

```
OMEGA: prove sqrt2-not-rat
Changing to proof plan SQRT2-NOT-RAT-1
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                                    OPEN

OMEGA: declare (constants (m num) (n num) (k num))
```

Step 1: We prove the goal indirectly, that is, we use the inference rule `noti`.

```
OMEGA: noti
NEGATION (NDLINE) A negated line: [SQRT2-NOT-RAT]
FALSITY (NDLINE) A falsity line: [()]

L1 (L1)           |- (RAT (SQRT 2))                                          HYP
L2 (L1)           |- FALSE                                                  OPEN
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                                NOTI: (L2)
```

Step 2: We load from the database the theorem `RAT-CRITERION`, which states that for each rational number x, there are integers y and z, such that $x \cdot y = z$, where y and z have no common divisor beside 1 (as the formalization of `common-divisor` states, cf. [Benzmüller *et al.*, 2002]). As a side effect, the newly introduced proof line containing that theorem is implicitly added to the hypotheses lists of all other proof lines. The retrieval of this theorem and its application in the appropriate context is non-trivial, since the only syntactical criterion that can be exploited to restrict the potentially large set of applicable theorems is the predicate symbol `RAT`. Automatic acquisition of theorems is a challenging problem and requires a combination of syntax-oriented and semantics-oriented tools. First ideas in this direction are presented in [Benzmüller *et al.*, 2003], assertion application based on resolution is described in [Vo *et al.*, 2003]. However, dynamic retrieval of appropriate assertions from a very large data base with mathematical knowledge is unsolved.

```
OMEGA: import-ass rat-criterion

RAT-CRITERION (RAT-CRITERION) |- (FORALL-SORT ([X].                          THM
                                   (EXISTS-SORT ([Y].
                                     (EXISTS-SORT ([Z].
                                       (AND (= (TIMES X Y) Z)
                                            (NOT (EXISTS-SORT ([D].
                                                   (COMMON-DIVISOR Y Z D))
                                                   INT))))
                                       INT))
                                     INT))
                                   RAT)
```

Step 3: We eliminate the sorted universal quantifier by instantiating its variable `X` with $\sqrt{2}$. This step is again non-trivial. A naive approach to automation, however, is to identify and subsequently instantiate terms of appropriate sort occurring in the proof context.

```
OMEGA: foralle-sort
UNIV-LINE (NDLINE) Universal line: [RAT-CRITERION]
LINE (NDLINE) A line: [()]
TERM (TERM) Term to substitute: (sqrt 2)
SO-LINE (NDLINE) A line with sort: [L1]
```

```
L3 (L1) |- (EXISTS-SORT ([DC-248].                    FORALLE-SORT: ((SQRT 2))
             (EXISTS-SORT ([DC-251].                        (RAT-CRITERION L1)
               (AND (= (TIMES (SQRT 2) DC-248) DC-251)
                    (NOT (EXISTS-SORT ([DC-255].
                           (COMMON-DIVISOR DC-248 DC-251 DC-255))
                           INT))))
               INT))
             INT)
```

Step 4: We eliminate the two sorted existential quantifiers by introducing the constants n and m. Since the quantifiers are soft-sorted, this introduces the additional information that n and m are integers. The newly introduced hypotheses L4 and L5, which express this sort information, are decomposed right away.

```
OMEGA: mexistse-sort*
CONCLINE (NDLINE) Conclusion Line.: [L2]
EXLINE (NDLINE) An existentially quantified line: [L3]
SUBGOAL (NDLINE) Subgoal Line.: [()]
PARAMETER (TERMSYM-LIST) Termsym List.: [(dc-2481 dc-2511)](n m)

L4  (L4)       |- (AND (INT N)                                              HYP
                      (EXISTS-SORT ([DC-251].
                        (AND (= (TIMES (SQRT 2) N) DC-251)
                             (NOT (EXISTS-SORT ([DC-255].
                                    (COMMON-DIVISOR N DC-251 DC-255))
                                    INT))))
                        INT))
L6  (L4)       |- (INT N)                                        ANDEL: (L4)
L5  (L5)       |- (AND (INT M)                                           HYP
                      (AND (= (TIMES (SQRT 2) N) M)
                           (NOT (EXISTS-SORT ([DC-255].
                                  (COMMON-DIVISOR N M DC-255))
                                  INT))))
L8  (L5)       |- (INT M)                                        ANDEL: (L5)
L9  (L5)       |- (AND (= (TIMES (SQRT 2) N) M)                  ANDER: (L5)
                      (NOT (EXISTS-SORT ([DC-255].
                             (COMMON-DIVISOR N M DC-255))
                             INT)))
L10 (L4 L5 L1) |- FALSE                                                  OPEN
L2  (L1)       |- FALSE                    Existse-Sort*-m: ((N M)) (L3 L10)
```

Step 5: Line L9 is further decomposed:

```
OMEGA: ande
CONJUNCTION (NDLINE) Conjunction to split: [L9]
LCONJ (NDLINE) Left conjunct: [()]
RCONJ (NDLINE) Right conjunct: [()]

L11 (L5) |- (= (TIMES (SQRT 2) N) M)                              ANDE: (L9)
L12 (L5) |- (NOT (EXISTS-SORT ([DC-255].                          ANDE: (L9)
                  (COMMON-DIVISOR N M DC-255)) INT))
```

Step 6: While the previous five steps were essentially canonical, we shall now start the island approach to sketch the refutation argument. First, we need some calculations to infer $2n^2 = m^2$ from $\sqrt{2}n = m$. To do so, we use the tactic ISLAND-TACTIC, which allows us to insert arbitrarily large

steps into our proof. The correctness of these steps is checked later when ISLAND-TACTIC is expanded. Note that we specify the premises we want to employ. This is important information because if we specify premises that are too weak, this may have the effect that the island gap cannot be successfully closed later on.

```
OMEGA: island-tactic
CONC (NDLINE) Conclusion of step: nil
PREMS (NDLINE-LIST) Premises of step: (L11 L6 L8)
PARAM (TERM) Formula of Conclusion: (= (times 2 (power n 2)) (power m 2))

L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2)) ISLAND-TACTIC: (L11 L6 L8)
```

Step 7: Next, we infer from $2n^2 = m^2$ that m^2 is even...

```
OMEGA: island-tactic nil (L13 L6 L8) (evenp (power m 2))

L14 (L4 L5) |- (EVENP (POWER M 2))                  ISLAND-TACTIC: (L13 L6 L8)
```

Step 8: ... and therefore m is even, too.

```
OMEGA: island-tactic nil (L14 L8) (evenp m)

L15 (L4 L5) |- (EVENP M)                              ISLAND-TACTIC: (L14 L8)
```

Step 9: Next, we unfold[3] the definition of EVENP.

```
OMEGA: defn-expand
LINE (NDLINE) Line to be rewritten: [RAT-CRITERION]L15
DEFINITION (THY-ASSUMPTION) Definition to be expanded: [EVENP]
POSITION (POSITION) Position of occurrence: [(0)]

L16 (L4 L5) |- (EXISTS-SORT ([DC-263].                              DefnE: (L15)
                (= M (TIMES 2 DC-263))) INT)
```

Step 10: As before, we eliminate the sorted existential quantifier by introducing the constant k. Again, the information that k is an integer is added automatically.

```
OMEGA: mexistse-sort*
CONCLINE (NDLINE) Conclusion Line.: [L10]
EXLINE (NDLINE) An existentially quanitified line: [L3]L16
SUBGOAL (NDLINE) Subgoal Line.: [()]
PARAMETER (TERMSYM-LIST) Termsym List.: [(dc-2631)](k)

L17 (L17)          |- (AND (INT K) (= M (TIMES 2 K)))                    HYP
L18 (L17)          |- (INT K)                                    ANDEL: (L17)
L19 (L17)          |- (= M (TIMES 2 K))                          ANDER: (L17)
L20 (L17 L4 L5 L1) |- FALSE                                          OPEN
L10 (L4 L5 L1)     |- FALSE                  Existse-Sort*-m: ((K)) (L16 L20)
```

[3]The folding and unfolding of definitions have historically been called definition contraction (**defn-contract**) and expansion (**defn-expand**) in ΩMEGA.

Step 11: Now, we can go on with our calculation using the ISLAND-TACTIC. By inserting $2k$ for m in $2n^2 = m^2$ we obtain $n^2 = 2k^2$.

```
OMEGA: island-tactic nil (L19 L13 L6 L8 L18)
                         (= (power n 2) (times 2 (power k 2)))

L21 (L4 L5 L17) |- (= (POWER N 2) (TIMES 2 (POWER K 2)))
                                        ISLAND-TACTIC: (L19 L13 L6 L8 L18)
```

Step 12: That means that n^2 is even...

```
OMEGA: island-tactic nil (L21 L6 L18) (evenp (power n 2))

L22 (L5 L4 L17) |- (EVENP (POWER N 2))              ISLAND-TACTIC: (L21 L6 L18)
```

Step 13: ... and so is n.

```
OMEGA: island-tactic nil (L22 L6) (evenp n)

L23 (L17 L5 L4) |- (EVENP N)                          ISLAND-TACTIC: (L22 L6)
```

Step 14: Since both n and m are even, they have a common divisor, namely 2.

```
OMEGA: island-tactic nil (L15 L23 L6 L8) (common-divisor n m 2)

L24 (L17 L4 L5) |- (COMMON-DIVISOR N M 2)        ISLAND-TACTIC: (L15 L23 L6 L8)
```

Step 15: This proves our contradiction and we are done.

```
OMEGA: island-tactic L20 (L12 L24) false

L20 (L17 L4 L5 L1) |- FALSE                           ISLAND-TACTIC: (L12 L24)
```

A verbal presentation of this proof is given in Fig. 2.

2.4 Closing the Gaps

The application of ISLAND-TACTIC does not necessarily result in an automatically verifiable logic-level proof, as filling the gaps can become a challenging task in its own right.

We shall now describe the (semi-automated) task of closing the gaps between the islands in our case study, which leads to a verifiable proof object at the logic level. ΩMEGA supports this process by providing interfaces to external systems. In particular, we use the theorem prover OTTER and the computer algebra system MAPLE. Although these external systems are essentially capable of closing the gaps in this case, the user still has to call them "in the right way" and to provide missing information. For instance, the user has to decide which system he wants to employ, he has to

load additional theorems from the database, and he has to manually unfold some definitions.

The first step when dealing with ISLAND-TACTIC is always to expand its application and we present this step only for the first application of ISLAND-TACTIC. By expanding the tactic its conclusion node becomes open (i.e., unjustified) again and all its premises are now support nodes, that is, the open node is supposed to be derivable from these support nodes. Moreover, theorems and axioms imported from the database automatically become support nodes as well.

Note that the expansion of a tactic application in ΩMEGA can result in proof segments that again contain tactic applications. Thus, expanding a tactic down to the ND level is a recursive process over several levels. We call the recursive process over all necessary levels until an ND-level proof is reached the *full expansion of a tactic*, whereas with *expansion of a tactic* we mean only the direct one-level expansion.

Proving L20 (n and m were supposed to have no common divisor but they actually do have 2 as a common divisor, hence contradiction): Line L20 is justified by an application of ISLAND-TACTIC to the premises L12 and L24. When we expand L20, it becomes open again and its support nodes specify that RAT-CRITERION, L12 and L24 can be used to close it (indeed, RAT-CRITERION is not necessary as we shall see later).

```
OMEGA: expand-node L20
Expanding the node L20 ...

L20 (L17 L4 L5 L1) |- FALSE                                              OPEN

OMEGA: show-supports L20
RAT-CRITERION L12 L24

L12 (L5)        |- (NOT (EXISTS-SORT ([DC-255].                    ANDE: (L9)
                         (COMMON-DIVISOR N M DC-255))
                         INT))
L24 (L17 L4 L5) |- (COMMON-DIVISOR N M 2)        ISLAND-TACTIC: (L15 L23 L6 L8)
```

Although the formulae involved are in first-order logic, OTTER fails to prove L20 with these supports.

```
OMEGA: call-otter-on-node L20
 Normalizing ...
 Calling otter process 27411 with time resource 10sec .
 otter Time Resource in seconds: 10sec
 Search stopped because sos empty.
 Parsing Otter Proof ...
 OTTER HAS FAILED TO FIND A PROOF
```

OTTER fails because one premise is missing, which is not inferable from the context, namely that 2 is an integer. Thus, we speculate this statement as a lemma for L20. This creates the new line L25, which is added as support for L20. We can prove L25 directly with the tactic WELLSORTED.

```
OMEGA: lemma L20 (INT 2)

L25 (L17 L4 L5 L1) |- (INT 2)                                          OPEN

OMEGA: wellsorted l25 ()

L25 (L17 L4 L5 L1) |- (INT 2)                                  WELLSORTED: ()
```

We apply OTTER again to L20, and this time it succeeds. TRAMP automatically translates the OTTER proof to an ND proof at the more abstract assertion level [Huang, 1994].

```
OMEGA: call-otter-on-node L20
 Normalizing ...
 Calling otter process 27554 with time resource 10sec .
 otter Time Resource in seconds: 10sec
 -------- PROOF --------
 Search stopped by max_proofs option.
 Parsing Otter Proof ...
 OTTER HAS FOUND A PROOF
 Creating Refutation-Graph ...
 Translating ...
 Translation finished!

L12 (L5)           |-  (NOT (EXISTS-SORT ([DC-255].                 ANDE: (L9)
                             (COMMON-DIVISOR N M DC-255))
                            INT))
L24 (L17 L4 L5)    |- (COMMON-DIVISOR N M 2)     ISLAND-TACTIC: (L15 L23 L6 L8)
L28 (L5)           |- (NOT (EXISTS [DC-97457]                     DEFNE: (L12)
                           (AND (INT DC-97457)
                                (COMMON-DIVISOR N M DC-97457))))
L30 (L17 L4 L5)    |- (NOT (INT 2))                        ASSERTION: (L28 L24)
L25 (L17 L4 L5 L1) |- (INT 2)                                WELLSORTED: ()
L31 (L1 L17 L4 L5) |- FALSE                                    NOTE: (L25 L30)
L29 (L17 L4 L5 L1) |- FALSE                                     WEAKEN: (L31)
L20 (L17 L4 L5 L1) |- FALSE                                     WEAKEN: (L29)
```

This proves that L20 is derivable from L12 and L24 at a lower level of abstraction. However, the nodes L30 and L25 are still not at the ND level, but justified by tactics. To verify these steps we have to fully expand them also, which works automatically and results in an ND-level subproof for L25 that consists of 13 steps and an ND-level subproof for L30 with 40 steps.

Proving L15 (m^2 is even implies m is even) and L23 (n^2 is even implies n is even): In order to close the gap between L15 and its premises L14 and L8 and between L23 and its premises L22 and L6 we need the theorem SQUARE-EVEN from the database. With this theorem OTTER manages to prove L15 and L23, respectively, and TRAMP outputs the corresponding assertion level proofs, which consist essentially of an application of the assertion SQUARE-EVEN. When fully expanded, the subproofs, that is, the ND-level proof deriving L15 from L14 and L8 and the ND-level proof deriving L23 from L22 and L6, consist of 6 steps each.

Proving L14 ($2n^2 = m^2$ implies m^2 is even) and L22 ($n^2 = 2k^2$ implies n^2 is even): The proof line L14 is justified by an application of

ISLAND-TACTIC to L13, L6, and L8. OTTER fails to prove L14 with respect to these supports. Indeed, using OTTER to obtain a proof for L14 requires some further steps: (1) we have to unfold the definition of EVENP in L14, and (2) we have to speculate that n^2 is an integer as a lemma for L14, which is added as node L41 to the proof. L41 can then be closed by applying the tactic WELLSORTED with L6 as premise such that afterwards the problem has the following form:

```
L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2)) ISLAND-TACTIC: (L11 L6 L8)
L41 (L4 L5) |- (INT (POWER N 2))                          WELLSORTED: (L6)
L40 (L4 L5) |- (EXISTS-SORT ([DC-98774].                             OPEN
                 (= (POWER M 2) (TIMES 2 DC-98774))) INT)
L14 (L4 L5) |- (EVENP (POWER M 2))                          DEFNI: (L40)
```

After applying OTTER successfully to L40, TRAMP provides a proof that derives L40 in 5 steps from L13 and L41. A fully expanded subproof at the ND level consists of 11 steps. When the application of WELLSORTED is fully expanded, L41 is derived from L6 by a subproof consisting of 17 steps.

Closing the gap between L22 and its premises L21, L6, and L18 works similarly. The only difference is that instead of the lemma that n^2 is an integer, the lemma that k^2 is an integer has to be speculated. This lemma can be closed by an application of the tactic WELLSORTED to the node L18 with formula (INT K).

Proving L24 (since n and m are even they have 2 as a common divisor): Before we can exploit OTTER to obtain a proof for the gap between L24 and its supports L15, L23, L6, and L8 we have to unfold some defined concepts and speculate some lemmata. In L24, we have to unfold the defined concept COMMON-DIVISOR, which closes L24 and results in a new open node L59. In L59 we then have to unfold all occurrences of the defined concept DIVISOR, which closes L59 and creates a new open node L60. L60 inherits the supports of L24 via L59. Next, we have to unfold EVENP in the two support nodes L15 and L23, which creates the two new supports L61 and L62 for L60 that contain the formulae resulting from unfolding EVENP. Moreover, for L60 we have to speculate the two lemmata that $1 \neq 2$ and that 2 is an integer, which are introduced as nodes L63 and L64 in the proof.

The open node L60 and its supports can now be proved using OTTER. TRAMP translates OTTER's proof into an ND-level proof that consists of 16 steps. We already proved that 2 is an integer in L25. To prove the second lemma, $1 \neq 2$, is actually not as trivial as it may seem. The numbers 1 and 2 are defined concepts in ΩMEGA and they are more convenient representations of (S ZERO) and (S (S (ZERO))), where S is the successor function. After unfolding 1 and 2 we have to prove that (NOT (= (S ZERO) (S (S ZERO)))) holds. We can prove this statement with OTTER, but to do so we need to import the following axioms from the database into the proof: ZERO is a natural number, the successor of a natural number is again a natural

number, the successor function is injective, and ZERO has no predecessor. Then TRAMP provides as output an assertion-level proof that consists of 8 steps. When fully expanded, the subproof for L63 consists of 28 steps.

Proving L13 ($\sqrt{2}n = m$ implies $2n^2 = m^2$) and L21 ($m = 2k$ and $2n^2 = m^2$ imply $n^2 = 2k^2$): So far, we always used OTTER and TRAMP to expand applications of ISLAND-TACTIC. Indeed, automated theorem proving was the right choice for this task, since all subproofs essentially rely on some theorems or definitions, which — after being imported from the database — can be used by OTTER to derive a proof.

In contrast, the steps deriving L13 from L11, L6, and L8 as well as deriving L21 from L19, L13, L6, L8, and L18 represent algebraic computations, for which ATPs are not particularly well suited. In ΩMEGA, the tactic BYCOMPUTATION employs the computer algebra system MAPLE to check whether an equation containing arithmetic expressions follows from a set of other equations. To do so, BYCOMPUTATION passes all equations to MAPLE and calls MAPLE's is function to check whether the conclusion equation holds assuming the premise equations. This tactic succeeds when applied to L13 and L21. For example, it closes L13 as follows:

```
L11 (L5)    |- (= (TIMES (SQRT 2) N) M)                          ANDE: (L9)
L13 (L4 L5) |- (= (TIMES 2 (POWER N 2)) (POWER M 2))      BYCOMPUTATION: (L11)
```

Currently, the tactic BYCOMPUTATION can only be partially expanded into an ND-level proof, namely for computations with polynomials. We are working on extensions of SAPPER to cover more cases, such as the equation above. The two applications of BYCOMPUTATION that justify L13 and L21 are therefore only "verified" by MAPLE, but not automatically by an ND-level proof.[4]

Result: When fully expanded, the island proof consists of 282 nodes, where the nodes L13 and L21 are justified by the unexpanded tactic BYCOMPUTATION. Hence, ΩMEGA's checker verifies that the proof is correct modulo the correctness of these computations. Fully expanding and checking the proof takes about 300 seconds on a 1.8 GHz Pentium III machine with 512 MB RAM running LINUX.

2.5 Automation of Proof Tasks

As described in detail in Sections 2.3 and 2.4, the proof of the $\sqrt{2}$-problem is constructed in two phases: First an abstract proof is outlined, in which

[4] Applications of mathematical systems for simplification make their efficient computation available to the construction of the top-level proof plan. The time-consuming verification is left to the expansion mechanism.

islands are coupled by tactics without care for the detailed logical dependencies between them. Next, the gaps between the islands are closed in the second phase with detailed ND-proof segments. So far both tasks require fairly detailed user interaction: at the abstract level the user has to provide the islands (and to apply some tactics); and for the expansion into the logic level he has to speculate the *right* lemmata, import the *right* theorems, and unfold the *right* definitions up to the *right* level.

The main research in ΩMEGA is currently to better automate these tasks. To this end, we examine two approaches:

First, proof planning with MULTI [Meier, 2003] is the means to automatically create island proofs at a user-friendly level of abstraction. In Section 3 we shall show the knowledge-engineering process that generalizes the main ideas underlying the interactive proof into corresponding proof methods and control knowledge for MULTI. Using this knowledge MULTI is capable of automatically planning the proofs for theorems of the generalized $\sqrt[i]{l}$-problems, that is, whether $\sqrt[i]{l}$ is irrational. We also have expansion tactics for the employed methods such that the expansions of the proof plans for $\sqrt[i]{l}$-problems can be done automatically and we can in fact fully expand and verify them with our proof checker (again, modulo the computer algebra system computations).

Second, semi-automated agent-based reasoning with Ω-ANTS is a means to obtain logic-level proofs for the interactively generated island proof plans with fewer user interactions. Expanding and closing island gaps is often more challenging than in proof planning, since there is no knowledge immediately available. In general, the expansion of the tactic `ISLAND-TACTIC` corresponds to a completely new theorem, which may be solved by providing more specific islands. The idea is that after some hierarchical decomposition the gaps become so small that they can be filled in automatically. To obtain a better degree of automation for closing the island gaps, we are working on the following ideas:

- Ω-ANTS "agentifies" methods, tactics, calculus rules, and heterogeneous external reasoners in the sense that these search proactively for their respective applicability. It should be possible to link the `ISLAND-TACTIC` with appropriate Ω-ANTS agents such that these autonomously and cooperatively try to close the gaps in the background while the user works on the next island steps. In case Ω-ANTS cannot close a gap automatically, the user will be informed and he may rethink the island step or he may provide further knowledge that can be used by Ω-ANTS.

- We are currently also examining the Ω-ANTS mechanism as a mediator between a knowledge base and proof planning (first results are reported in [Benzmüller *et al.*, 2003]). The mediator supports the idea

of semantically guided retrieval of mathematical knowledge (theorems, definitions) from the database MBase.

- The speculation of suitable lemmata can be supported by a model generator. For instance, when applying the model generator Mace or Satchmo to the failing proof attempt with Otter for `L20` a counter-model is generated, in which 2 is not an integer. A general mechanism that employs model generation for the speculation of missing lemmata (such as the one asserting that 2 is an integer) is certainly possible and promising.

- A better support for unfolding definitions is to adapt Bishop and Andrew's selective unfolding mechanism [Bishop and Andrews, 1998] to our proof planning context.

3 PROOF PLANNING THE $\sqrt{2}$-PROBLEM

The user has to apply the island tactic eight times in the proof discussed in Section 2 (namely in steps 6, 7, 8, 11, 12, 13, 14, and 15). These are the crucial and creative steps that provide the essential idea of this proof.

Now, the question is: Can we find these creative steps automatically? The answer is yes, as we shall show in this section. However, while we can answer the question in the affirmative, not every reader may be convinced that this is really the final answer, as our solution touches upon a subtle point, which opens the Pandora Box of critical issues in the paradigm of proof planning [Bundy, 2002]. It is easy to write some specific methods, which perform just the steps in the interactively found proof and then call the proof planner Multi to fit the methods together into a proof plan for our problem. This, of course, shows nothing of substance: Just as we could write down all the definitions and theorems required for the problem in first-order predicate logic and hand them to a first-order prover such as Otter,[5] we would just hand-code the final solution into appropriate methods.

Instead, the goal of the game is to find *general* methods for a whole class of theorems within some theory that can solve not only this particular problem, but also all the other theorems in that class. While our approach essentially follows the proof idea of the interactively constructed proof for the $\sqrt{2}$-problem, it relies essentially on more general concepts such that we can solve, for example, $\sqrt[j]{l}$-problems for arbitrary natural numbers j and l. However, as we shall discuss in Section 3.4, this is certainly not the end of the story.

[5] As it was done when tackling the $\sqrt{2}$-problem with Otter; see [Wiedijk, 2002] for the original Otter case study and [Benzmüller *et al.*, 2002] for its replay with ΩMEGA.

3.1 The Knowledge Acquisition

In order to find a general approach for $\sqrt[j]{l}$-problems for arbitrary natural numbers j and l, we first analyzed proofs for statements such as $\sqrt{8}$, $\sqrt{(3 \cdot 3) - 1}$, or $\sqrt[3]{2}$. We found that some of the concepts and inference steps we used for $\sqrt{2}$ are particular to this problem and do not generalize whereas others do. Thus, the analysis led to some generalized concepts, theorems, and proof steps, which we encoded into methods and control rules, that together form one planner strategy for this kind of problems. We shall now discuss the acquired methods and control rules.

The essential ideas of the proof in Section 2 are as follows:

(1) Use the theorem `RAT-CRITERION` and construct an indirect proof.

(2) In order to derive the contradiction show that the two constants (existential variables) in `RAT-CRITERION`, which are supposed to have no common divisor, actually do have a common divisor d.

(3) In order to find a common divisor transform equations (for example, $\sqrt{2} \cdot n = m \longrightarrow 2 \cdot n^2 = m^2$), derive new divisor statements (for example, from $2 \cdot n^2 = m^2$ derive that m^2 has divisor 2, or from the statement that m^2 has divisor 2 derive that m has divisor 2), and derive from given divisor statements new representations of terms, which can be used again for equational transformations (for example, from the statement that m has divisor 2 derive that $m = 2 \cdot k$).

Note that we are particularly interested in prime divisors, since only for prime numbers d is it true that if d is a divisor of m^j then d is also a divisor of m. A corresponding theorem, which generalizes `SQUARE-EVEN`, is now available in ΩMEGA's database:

```
(th~deftheorem POWER-PRIME-DIVISOR (in integer)
  (conclusion (forall-sort (lam (n num)
                (forall-sort (lam (d num)
                  (forall-sort (lam (x num)
                    (implies (prime-divisor d (power x n))
                             (prime-divisor d x)))
                    INT))
                  INT))
                NAT)))
```

To realize idea (1) the planner MULTI has to decide to try an indirect proof, apply the theorem `RAT-CRITERION`, and derive $l \cdot n^j = m^j$ for integers m and n, which are supposed to have no common divisor. These steps are canonical for arbitrary $\sqrt[j]{l}$ problems. Hence, we could implement them all into one method. However, to avoid the well known problem of overfitting methods we decided to employ already existing methods from other domains: NotI-M (prove by contradiction), MAssertion-M (apply a theorem or an axiom from the theory), ExistsE-Sort-M (decompose existentially quantified formulae), AndE-M (decompose conjunctions).

The application of the methods ExistsE-Sort-M, AndE-M, and NotI-M do not need any further control, but the application of MAssertion-M has to be guided by selecting the theorem or axiom to be applied by the method. This is achieved by a control rule `apply-ratcriterion`, which determines that the theorem `RAT-CRITERION` should be used for MAssertion-M, whenever there is a formula $\sqrt[j]{l}$.

Idea (2) is realized with the method ContradictionCommonDivisor-M. When MULTI tries to apply the method it searches first for a proof line that states that two terms t_1, t_2 have no common divisor, and second for two proof lines that state that t_1 and t_2, respectively, have a divisor d. This method is not guided by control rules, but MULTI tries to apply it to some derived proof lines in each planning cycle.

Idea (3) of the proof technique is encoded into several collaborating methods: TransFormEquation-M, =Subst-m, PrimeFacsProduct-M, PrimeDivPower-M, and CollectDivs-M. The method TransFormEquation-M contains the knowledge about suitable equational transformations for our problem domain. It is applied to an equation and derives a new equation. For instance, TransFormEquation-M derives $l \cdot n^j = m^j$ from $\sqrt[j]{l} \cdot n = m$, or it derives $n^2 = 2 \cdot k^2$ from $2 \cdot n^2 = (2 \cdot k)^2$. The method =Subst-m performs equality substitutions.

PrimeFacsProduct-M and PrimeDivPower-M encapsulate the knowledge of how to derive divisor statements. PrimeFacsProduct-M is applied to equations $x = l \cdot y$ (or $l \cdot y = x$) and returns a proof line whose formula is a conjunction of statements that x has particular prime divisors. The method employs MAPLE to compute the prime divisors of l using MAPLE's function `with(numtheory,factorset)`. It derives that x has to have all prime divisors of l. For instance, from $2 \cdot n^2 = m^2$ PrimeFacsProduct-M derives that m^2 has the prime divisor 2, from $6 \cdot n^2 = m^2$ it derives that m^2 has the prime divisors 2 and 3. PrimeDivPower-M is applied to an assumption that states that y^j has prime divisor d and derives that y has prime divisor d.

For a term t CollectDivs-M searches for proof lines that state that t has some prime divisors. Then, it computes different possible representations of t based on the set of the prime divisors $\{p_1, \ldots, p_n\}$. That is, for each subset $\{p_{1'}, \ldots, p_{n'}\}$ of $\{p_1, \ldots, p_n\}$ it returns the proof line $t = p_{1'} \cdot \ldots \cdot p_{n'} \cdot c'$ for some integer c'.

TransFormEquation-M, PrimeFacsProduct-M and PrimeDivPower-M are applied whenever possible and no guidance is required. The application of the method CollectDivs-M, however, is guided by the control rule `apply-collectdivs`, which prefers CollectDivs-M with respect to a term t as soon as there are proof lines that state that t has some prime divisors. The application of =Subst-m is guided by the control rule `apply-=subst`, which states that, after an application of CollectDivs-M, the method =Subst-m should be applied in order to use the equations resulting from CollectDivs-M. When a method such as =Subst-m, PrimeFacsProduct-M, or PrimeDivPower-

M is applied to some premises, then the same method is afterwards applicable again to the same premises, deriving the same result. To avoid endless loops of such methods, we added the control rule `reject-loop`, which blocks the repeated application of a forward method to the same premises.

3.2 Applying MULTI *to the* $\sqrt{2}$*-Problem*

MULTI constructs now a proof plan as follows:

First, it applies the methods MAssertion-M, NotI-M, ExistsE-Sort-M, AndE-M, and TransFormEquation-M, in order to apply the theorem `RAT-CRITERION`, to establish an indirect proof, and to decompose existing existentially quantified formulae or conjunctions.[6]

```
L3 (L3)          |- (EXISTS-SORT ([DC-626].                                  HYP
                      (EXISTS-SORT ([DC-629].
                        (AND (= (TIMES (SQRT 2) DC-626) DC-629)
                             (NOT (EXISTS-SORT ([DC-633].
                                    (COMMON-DIVISOR DC-626 DC-629 DC-633))
                                    INT))))
                        INT))
                      INT)
L5 (L5)          |- (AND (INT CONST1)                                        HYP
                        (EXISTS-SORT ([DC-629].
                          (AND (= (TIMES (SQRT 2) CONST1) DC-629)
                               (NOT (EXISTS-SORT ([DC-633].
                                      (COMMON-DIVISOR CONST1 DC-629 DC-633))
                                      INT))))
                          INT))
L7 (L5)          |- (INT CONST1)                                  AndE-m: (L5)
L8 (L5)          |- (EXISTS-SORT ([DC-629].                       AndE-m: (L5)
                      (AND (= (TIMES (SQRT 2) CONST1) DC-629)
                           (NOT (EXISTS-SORT ([DC-633].
                                  (COMMON-DIVISOR CONST1 DC-629 DC-633))
                                  INT))))
                      INT)
L9 (L9)          |- (AND (INT CONST2)                                        HYP
                        (AND (= (TIMES (SQRT 2) CONST1) CONST2)
                             (NOT (EXISTS-SORT ([DC-633].
                                    (COMMON-DIVISOR CONST1 CONST2 DC-633))
                                    INT))))
L11 (L9)         |- (INT CONST2)                                  AndE-m: (L9)
L12 (L9)         |- (AND (= (TIMES (SQRT 2) CONST1) CONST2)       AndE-m: (L9)
                        (NOT (EXISTS-SORT ([DC-633].
                               (COMMON-DIVISOR CONST1 CONST2 DC-633))
                               INT)))
L13 (L9)         |- (= (TIMES (SQRT 2) CONST1) CONST2)           AndE-m: (L12)
L14 (L9)         |- (NOT (EXISTS-SORT ([DC-633].                 AndE-m: (L12)
                           (COMMON-DIVISOR CONST1 CONST2 DC-633))
                           INT)))
L15 (L9)         |- (= (TIMES 2 (POWER CONST1 2))    TRANSFORMEQUATION-M: (L13)
                      (POWER CONST2 2))
```

[6]Actually, MAssertion-M, which applies `RAT-CRITERION`, also introduces the open line `L1`, which is closed by the method Reflex-M. This line results from variable bindings internal to the theorem application process. It states that the variable `X1`, which corresponds to the universally quantified variable in `RAT-CRITERION`, has to be bound to the term `(SQRT 2)`. This binding has already been applied to the rest of the proof (e.g., in `L2`).

```
L10 (L9 L5 L3)   |- FALSE                                                    OPEN
L6 (L5 L3)       |- FALSE                                 Existse-Sort-m: (L8 L10)
L4 (L3)          |- FALSE                                  Existse-Sort-m: (L3 L6)
L2 ()            |- (NOT (EXISTS-SORT ([DC-626].                  NOTI-M: (L4)
                          (EXISTS-SORT ([DC-629].
                            (AND (= (TIMES (SQRT 2) DC-626) DC-629)
                                 (NOT (EXISTS-SORT ([DC-633].
                                        (COMMON-DIVISOR DC-626 DC-629 DC-633))
                                        INT))))
                                 INT))
                                INT))
L1 ()            |- (= X1 (SQRT 2))                                      REFLEX-M
SQRT2-NOT-RAT () |- (NOT (RAT (SQRT 2)))                  MAssertion-M: (L2)
```

Next, using the equation $2 \cdot const1^2 = const2^2$ in line L15[7] the methods PrimeFacsProduct-M and PrimeDivPower-M derive that $const2$ has the prime divisor 2.

```
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))    PRIMEFACS-PRODUCT-M: (L15)
L17 (L5 L9) |- (PRIME-DIVISOR 2 CONST2)                 PRIMEDIV-POWER-M: (L16)
```

Then, CollectDivs-M computes a representation for $const2$ with respect to line L17. Since CollectDivs-M introduces a new hypothesis in line L19 it reduces also the open line L10 to the new open line L20, which also contains the new hypothesis.

```
L19 (L19)          |- (AND (INT CONST3) (= CONST2 (TIMES 2 CONST3)))        HYP
L21 (L19)          |- (INT CONST3)                                    AndE-m: (L19)
L22 (L19)          |- (= CONST2 (TIMES 2 CONST3))                     AndE-m: (L19)


L20 (L19 L9 L5 L3) |- FALSE                                                OPEN
L10 (L9 L5 L3)     |- FALSE                                 COLLECTDIVS-M: (L20 L17)
```

Next, the methods =Subst-m and TransFormEquation-M derive with the new representation for $const2$ the equation in line L25.

```
L24 (L19 L9) |- (= (TIMES 2 (POWER CONST1 2))                  =subst-m: (L15 L22)
                  (POWER (TIMES 2 CONST3) 2))
L25 (L9 L19) |- (= (POWER CONST1 2)                    TRANSFORMEQUATION-M: (L24)
                  (TIMES 2 (POWER CONST3 2)))
```

Then, with respect to this equation the methods PrimeFacsProduct-M and PrimeDivPower-M derive that $const1$ has the prime divisor 2.

```
L26 (L5 L19 L9) |- (PRIME-DIVISOR 2 (POWER CONST1 2))
                                                    PRIMEFACS-PRODUCT-M: (L25)
L27 (L9 L19 L5) |- (PRIME-DIVISOR 2 CONST1)            PRIMEDIV-POWER-M: (L26)
```

Finally, ContradictionCommonDivisor-M closes the open line L20 and MULTI terminates with the final line:

```
L20 (L19 L9 L5 L3) |- FALSE         CONTRADICTIONCOMMONDIVISOR-M: (L14 L27 L17)
```

[7] Here, the automatically generated constants $const1$ and $const2$ replace the constants n and m, respectively, in the interactive proof given in Section 2.

This automatically generated proof plan required less than four seconds CPU time on a 1.8 GHz Pentium III machine with 512 MB RAM running LINUX. A considerable amount of time was required to call the external computer algebra system MAPLE twice within the applications of the method PrimeFacsProduct-M. The complete proof plan can then be passed to *P.rex*, which produces a natural language presentation of the proof as given in Fig. 3. This concludes the first phase, that is, the automated construction of a proof plan, and we shall now look at the second phase, that is, the expansion of this proof plan to an ND-level proof.

3.3 Expansion

All methods used in ΩMEGA have as part of their specification the knowledge on how to expand. A method contains a schematic proof segment that shows how to derive its conclusions from its premises using tactics and rules. When an application of the method is expanded, then the proof schema is instantiated and introduced into the proof plan justifying the conclusions of the method from its premises at a lower level of abstraction. Note that the methods only specify their direct expansion level, the recursive expansion is part of the overall expansion process.

As an example, consider the expansion of PrimeDivPower-M and PrimeFacsProduct-M, which demonstrate in particular how additional theorems and sort statements are found and treated in the expansion process.

The expansion schema for PrimeDivPower-M is as follows:

```
(DECL-CONTENT
    (S0 () (PRIME-DIVISOR N (POWER A M)))
    (S1 () (NAT M)                                          ("WELLSORTED" ()))
    (S2 () (INT N)                                          ("WELLSORTED" ()))
    (S3 () (INT A)                                        ("WELLSORTED" (IA)))
    (S4 () (IMPLIES (PRIME-DIVISOR N (POWER A M))
                    (PRIME-DIVISOR N A))     ("FORALLE-SORT*" (PPD S1 S2 S3)))
    (S5 () (PRIME-DIVISOR N A)                            ("IMPE" () (S4 S0))))
```

This proof segment represents a proof at tactic level. It shows how to derive line S5 (the conclusion of the method) from line S0 (the premise of the method). This proof segment uses the `Power-Prime-Divisor` theorem, whose incorporation into the proof plan during the expansion of an application of PrimeDivPower-M is specified in the method by a so-called expansion computation (see [Meier *et al.*, 2002a]). The `Power-Prime-Divisor` theorem is abbreviated in the proof segment as PPD and it is used to derive S4. To apply the theorem we have to establish some sort statements for m, n, and a in the lines S0, S1, and S2, respectively. Since m and n occur in the application part of the method, there is a concrete natural number and a concrete integer, respectively. We can prove these sort statements with

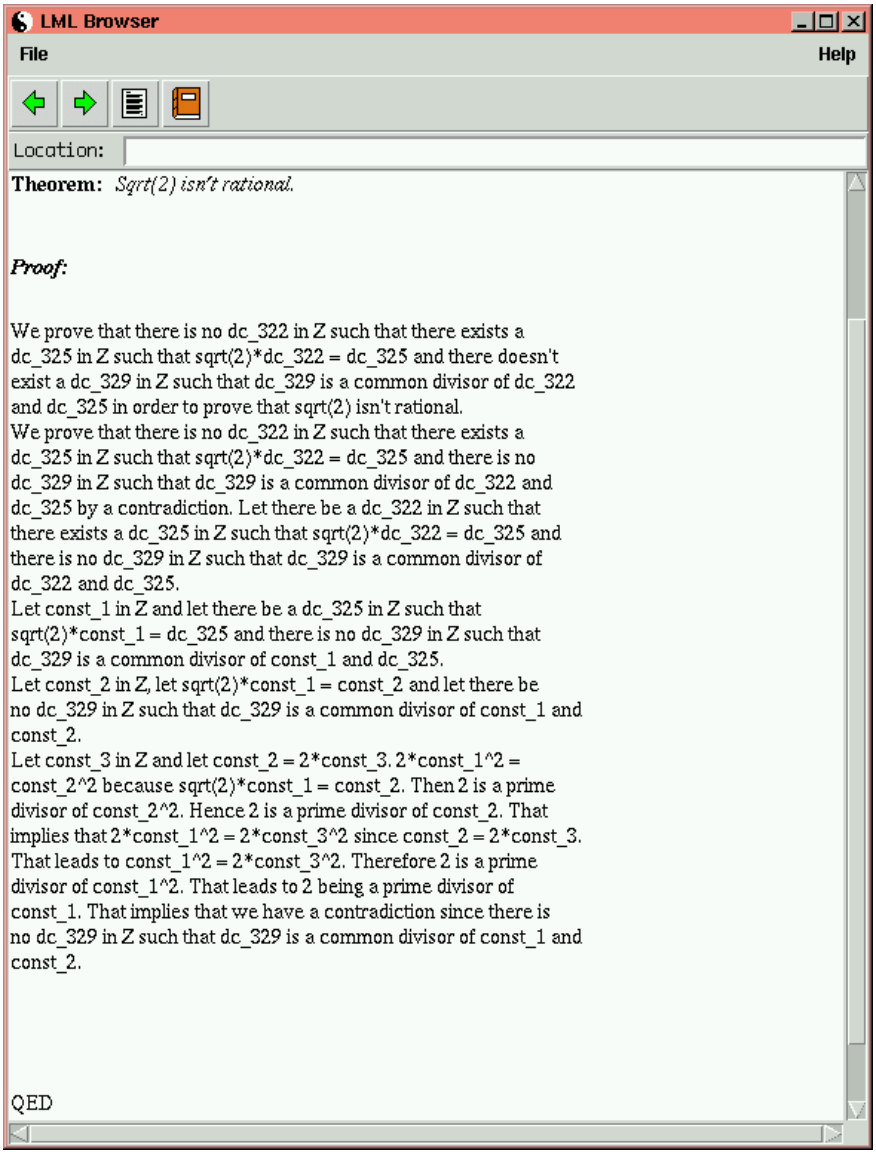


Figure 3. Natural language proof presentation of the planned proof by *P.rex*.

the tactic WELLSORTED. The variable a does not represent a concrete number, thus we have to derive from existing hypotheses that a is an integer. This derivation is part of the applicability check of the method and these hypotheses are represented above as IA in S3.

PrimeDivPower-M derives line L17. When it is expanded, the following proof segment is added to the proof plan:

```
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))    PRIMEFACS-PRODUCT-M: (L15)
L28 ()      |- (NAT 2)                                        WELLSORTED: ()
L29 ()      |- (INT 2)                                        WELLSORTED: ()
L30 (L9)    |- (INT CONST2)                                 WELLSORTED: (L11)
L31 (L9)    |- (IMPLIES                        FORALLE-SORT*: (PPD L28 L29 L30)
                 (PRIME-DIVISOR 2 (POWER CONST2 2))
                 (PRIME-DIVISOR 2 CONST2))
L17 (L5 L9) |- (PRIME-DIVISOR 2 CONST2)                        IMPE: (L31 L16)
```

Here, PPD stands for the Power-Prime-Divisor theorem. This theorem is now inserted into the proof plan as:

```
PPD (PPD) |-  (FORALL-SORT ([N].                                          THM
                (FORALL-SORT ([D].
                  (FORALL-SORT ([X]. (IMPLIES (PRIME-DIVISOR D (POWER X N))
                                              (PRIME-DIVISOR D X)))
                    INT))
                  INT))
                NAT)))
```

WELLSORTED and FORALLE-SORT* are tactics, whose applications can be expanded automatically. If all of this is done, the resulting logic-level proof segment derives L17 from L16 in 19 steps.

The proof schema of PrimeFacsProduct-M specifies that the conclusion of the formula, line S2, is derived by an application of the tactic EXPAND--PRIMEFACSPRODUCT to the premise of the method in line S1:

```
(DECL-CONTENT
    (S1 () (= A B))
    (S2 () CONJUNCTION        ("EXPAND-PRIMEFACSPRODUCT" () (S1 SORT-PREMS))))
```

Here, CONJUNCTION is a substitute for the actual conjunction of terms (PRIME-DIVISOR P X), which are computed during the application of the method. As opposed to the expansion of a method, which is stated declaratively, the expansion of a tactic is given by a LISP function.

The method PrimeFacsProduct-M derives line L16. When it is expanded, no new proof lines are added, but the justification of L16 is changed:

```
L11 (L9)    |- (INT CONST2)                                    AndE-m: (L9)
L7  (L5)    |- (INT CONST1)                                    AndE-m: (L5)
L15 (L9)    |- (= (TIMES 2 (POWER CONST1 2))         TRANSFORMEQUATION-M: (L13)
                 (POWER CONST2 2))
L16 (L5 L9) |- (PRIME-DIVISOR 2 (POWER CONST2 2))
                                  EXPAND-PRIMEFACS-PRODUCT: (L15 L11 L7)
```

In this case, lines L11 and L7 are mandatory premises for the tactic EXPAND-PRIMEFACSPRODUCT. They provide the necessary sort information.

The expansion of EXPAND-PRIMEFACSPRODUCT works as follows: The first premise of EXPAND-PRIMEFACSPRODUCT is an equation $x = l \cdot y$, and its conclusion is a conjunction of terms (PRIME-DIVISOR P X). Moreover, further premises of EXPAND-PRIMEFACSPRODUCT are necessary sort statements. First, for each conjunct (PRIME-DIVISOR P X) of the conclusion the definition of PRIME-DIVISOR is expanded.

```
(th~defdef prime-divisor (in integer)
  (definition (lam (x num) (lam (y num) (and (divisor x y) (prime x)))))
  (help "The predicate for prime divisors."))
```

This results in two new subgoals, respectively, namely that p is a prime number, formalized as (PRIME P), and that p is a divisor of x, formalized as (DIVISOR P X). By rewriting the term $l \cdot y$ to $p \cdot (r \cdot y)$ (r is computed from l and p by MAPLE), the expansion establishes that $x = p \cdot (r \cdot y)$ holds. Since this is essentially the definition of divisor in ΩMEGA's database, the expansion can derive (DIVISOR P X).

```
(th~defdef divisor (in integer)
  (definition
    (lam (x num)
         (lam (y num)
              (and (and (int x) (int y))
                   (exists-sort (lam (z num) (= y (times x z))) int)))))
  (help "The predicate for integer divisibility."))
```

The sort premises are needed, since the divisor definition requires that y, p, and $r \cdot x$ are of sort integer.

To rewrite $l \cdot y$ as stated above, the expansion has to establish that $l = p \cdot r$ holds, where l, p, and r are concrete numbers. This is currently justified by an application of the tactic BYCOMPUTATION, that is, it is verified by MAPLE. The same holds for the statements (PRIME P) for concrete numbers p. They are also justified by BYCOMPUTATION.

The expansion of EXPAND-PRIMEFACSPRODUCT in turn employs other tactics (e.g., tactics for definition expansion and equality substitution). The recursive expansion of all tactics that prove L16 results in a proof segment that derives L16 from L15, L11, and L7 in 75 steps at the logic level. The proof segment is verified except for the mentioned proof nodes justified by applications of BYCOMPUTATION.

The expansion of all other methods works similarly. Note that the methods CollectDivs-M and ContradictionCommonDivisor-M expand to proof segments that use the tactic OTTER. When this tactic is expanded, it employs OTTER in order to justify the conclusion of the tactic application from its premises and the resulting proof lines are inserted.

Completely expanded, the proof of the $\sqrt{2}$-problem consists of 753 steps. The proof is verified except for the nodes justified by the tactic

BYCOMPUTATION, which is used 6 times, that is, currently we trust MAPLE. Note that the expansion mechanism operates locally and schematically and therefore does not optimize the final proof for its size.

3.4 Discussion

In order to evaluate the appropriateness of our approach we suggest the following three criteria:

(1) How general and how rich in mathematical content are the methods and control rules?

(2) How much search is involved in the proof planning process?

(3) What kind of proof plans, that is, what kind of proofs, can we find?

These criteria should allow us to judge how general and how robust our solution is. The art of proof planning is to acquire domain knowledge that, on the one hand, comprises meaningful mathematical techniques and powerful heuristic guidance, and, on the other hand, is general enough to tackle a broad class of problems. For instance, as one extreme, we could have methods that encode ΩMEGA's ND calculus and we could run MULTI without any control. This approach would certainly be very general, but MULTI would fail to proof plan any interesting problems. As the other extreme case, we could cut a known proof into pieces, and code the pieces as methods. Guided by control rules that always pick the next right piece of the proof MULTI would assemble the methods again to the original proof without performing any search.

The amount of search and the variety of potential proof plans for a given problem are measures for the generality of the methods and also for the appropriateness for tackling the class of problems by planning. If tight control rules or highly specific methods restrict the search to just one branch in the search tree, then the resulting proof plans will merely instantiate a pattern. In this case, a single tactic or method that realizes the proof steps of the underlying pattern is more suitable than planning. The possibility of creating a variety of proof plans with the given methods and control rules is thus an important feature.

In the following, we shall discuss proof planning for $\sqrt[i]{l}$-problems with respect to these three criteria.

(1) The methods Notl-M, ExistsE-Sort-M, AndE-M, =Subst-m, and MAssertion-M of our approach encode logic-level steps (Notl-M, AndE-M, =Subst-m) or tactic steps very close to the logic-level (ExistsE-Sort-M, MAssertion-M). Thus, they are very general, but they do not encode specific domain knowledge, and they are in fact still in the spirit of Gerhard Gentzen's analysis of mathematical proofs [Gentzen, 1935].

PrimeFacsProduct-M and CollectDivs-M encode domain knowledge about integers. PrimeFacsProduct-M encodes the extraction of prime divisors for integers from equations on integers. CollectDivs-M computes different representations for integers, for which some divisors are known. The functionalities of these methods are currently rather restricted and focused on our actual problem domain (e.g., CollectDivs-M could deal also with divisors and not only with prime divisors, PrimeFacsProduct-M could handle a more general class of equations). However, suitably extended, these two methods will be useful for many problem classes dealing with integers.[8]

PrimeDivPower-M applies the theorem `POWER-PRIME-DIVISOR`. This could be done also by the MAssertion-M method. We encoded the application of this theorem into an extra method in order to hide the sort goals. Thus, PrimeDivPower-M is a very specific method just fitted to the particular needs of our problem class. The same holds for TransFormEquation-M, which performs exactly the equation transformations that we need to deal with our problem class.

ContradictionCommonDivisor-M is not specific to our domain and should also be useful for other problem classes dealing with divisors.

The control rules `apply-collectdivs`, `apply-=subst`, and `reject-loop` contain no particular domain knowledge, but force MULTI into the right search branch. Hence, they are useful for any search. Only `apply-ratcriterion` touches a subtle point: this control rule encodes the knowledge that the theorem `RAT-CRITERION` should be applied via the method MAssertion-M and is hence fitted to our particular problem domain. We are currently examining a mediator mechanism between our knowledge base and proof planning (first results are reported in [Benzmüller *et al.*, 2003]), which supports a semantically guided retrieval of theorems and definitions. This approach should replace particular control rules for theorem retrieval (such as `apply-ratcriterion`) by a more general mechanism.

(2) MULTI performs depth-first search, which typically involves backtracking from search branches with no solutions. In our domain and with the described methods and control rules, however, there is no backtracking. Rather, the search consists of all possible transformations and derivations for finding two prime divisors that yield the contradiction.

For instance, when tackling the $\sqrt{6}$-problem, MULTI derives from $6 \cdot n^2 = m^2$ that m has two prime divisors 2 and 3. With respect to these prime divisors it computes the following three representations of m: $m = 2 \cdot a$, $m = 3 \cdot b$, $m = 2 \cdot 3 \cdot c$, and each representation is used to substitute m in the equation $6 \cdot n^2 = m^2$.

[8] To extend the capabilities of the methods we would also have to extend the tactics for the expansion of the methods. Since this is not trivial for both methods, we instead decided for now to implement these "light" versions for which the expansion is fully specified and leave the generality to future work.

Not all proof plans we found are similar to the proof plan for the $\sqrt{2}$-problem. For instance, the proof plan for the $\sqrt{8}$-problem has the following steps: From $8 \cdot n^2 = m^2$ MULTI derives that m has the prime divisor 2. With respect to this prime divisor m can be written as $m = 2 \cdot a$. Substituting m in the initial equation and simplifying the equation yields $2 \cdot n^2 = a^2$. From this equation no prime divisors of n can be derived, which would then yield a contradiction with the prime divisors of m as in the proof plan for the $\sqrt{2}$-problem. Instead, MULTI derives that 2 is a prime divisor of a and computes a new representation for a: $a = 2 \cdot b$. Substituting a with respect to this equation then yields that $n^2 = 2 \cdot b^2$ from which MULTI can derive that n has the prime divisor 2 which yields the contradiction to the fact that m has also the prime divisor 2.

Although the proof plans for different problems can be different we found that they vary only very little. The main variations are with respect to the numbers of "detect prime divisors for c"—"represent c as $c = p \cdot c'$"—"substitute c in prior equations" cycles. In particular, the proof plans for every $\sqrt{p}$-problem, where p is a prime number, looks exactly as the proof plan for the $\sqrt{2}$-problem.

(3) The number of potential proof plans depends on the particular problem. For prime numbers p the proof plan is the same as for the $\sqrt{2}$-problem (modulo instantiations). For numbers l that have several prime divisors there are typically several proof plans that vary with respect to the common divisors.[9]

This concludes the actual descriptions of how the $\sqrt{2}$-problem (respectively its generalization, the $\sqrt[j]{l}$-problem) was solved with ΩMEGA.

What general lessons can we learn from small, albeit typical mathematical challenges of this kind?

1. The devil is in the detail, that is, it is always possible to hide the crucial creative step in some small pre-programmed step and to pretend a level of generality that has not actually been achieved. To evaluate a solution *all* tactics, methods, theorems and definitions have to made explicit.

   In this paper, we have tried to strike a balance and to provide enough information to judge the strength (and weakness) of the current state of the art in the new paradigm of proof planning without providing too many details.

[9]For instance, when tackling $\sqrt{6}$ MULTI derives from $6 \cdot n^2 = m^2$ that m has the prime divisors 2 and 3 and hence m can be represented as $m = 2 \cdot 3 \cdot a$. From this representation MULTI derives that $n^2 = 6 \cdot a^2$. This equation yields that n has the prime divisors 2 and 3. Now, MULTI can use both 2 and 3 to derive a contradiction.

2. The enormous distance between the well-known (top-level) proof of the Pythagorean School, which consists of about a dozen single proof steps in comparison to the final proof at the ND level with 753 is striking.

   This is, of course, not a new insight. While mathematics can *in principle* be reduced to purely formal logic-level reasoning as demonstrated by Russell and Whitehead as well as the Hilbert School, nobody would actually want to do so *in practice* as the influential Bourbaki group showed: only the first quarter of the first volume in the several dozen volume set on the foundation of mathematics starts with elementary, logic-level reasoning and then proceeds with the crucial sentence [Bourbaki, 1968]: "No great experience is necessary to perceive that such a project [of complete formalization] is absolutely unrealizable: the tiniest proof at the beginning of the theory of sets would already require several hundreds of signs for its complete formalization."

3. Finally and more to the point of the concrete contribution of this paper: Now that we can prove theorems in the $\sqrt[3]{1}$-problem class, the skeptical reader may still ask *So what?* Will this ever lead to a *general* system for mathematical assistance?

   We have demonstrated in [Melis and Siekmann, 1999; Melis, 1998] that the class of ϵ-δ-proofs for limit theorems can indeed be solved with a few dozen mathematically meaningful methods and control rules. Similarly, the domain of group theory with its class of residue theorems can be formalized with even less [Meier and Sorge, 2000; Meier *et al.*, 2001; Meier *et al.*, 2002b], and the crucial general observation is that these methods correspond to the kind of mathematical knowledge a freshman would have to learn to master this level of professionalism.

   Is the same true for $\sqrt[3]{1}$-problems? The unfortunate answer is probably *No!* Imagine the subcommittee of the United Nations in charge of the maintenance of the global mathematical knowledge base in a hundred years from now. Would they accept the entry of our methods, tactics and control rules for the $\sqrt[3]{1}$-problems? Probably not!

4 MATHEMATICAL KNOWLEDGE ENGINEERING

Mathematical knowledge is preserved in books and monographs, but the art of doing mathematics [Polya, 1973; Hadamard, 1944] is passed on by word of mouth from generation to generation. The methods and control rules of the proof planner correspond to important mathematical techniques and

"ways to solve it", and they make this implicit and informal mathematical knowledge explicit and formal[10].

The theorems about $\sqrt[j]{l}$-problems are shown by contradiction, that is, the planner derives a contradiction from the equation $l \cdot n^j = m^j$, where n and m are integers with no common divisor. However, these problems belong to the more general class to determine whether two complex mathematical objects $\mathcal{X}$ and $\mathcal{Y}$ are equal. A general mathematical principle for comparison of two complex objects is to look at their characteristic properties, for example, their normal forms or some other uniform notation in the respective theory. If there is a characteristic property that distinguishes the two objects then they cannot be equal.

In order to tackle $\sqrt[j]{l}$-problems we have to find a property that distinguishes the integers $l \cdot n^j$ and m^j. Since each integer can be represented as a product of prime numbers, the normal forms in the integer theory are products of primes. If $P[i]$ denotes the prime product of i, then we can write $l \cdot n^j$ and m^j as $P[l] \cdot P[n^j]$ and $P[m^j]$. For instance, for $l = 2$, $j = 2$ this is $2 \cdot P[n^2]$ and $P[m^2]$. A characteristic property distinguishing these two integers is the quantity of prime numbers in the normal form. For $P[n^2]$ and $P[m^2]$ we do not know the exact quantity, but we do know for both that the quantity is even, since each occurrence in $P[x]$ is duplicated in $P[x^2]$. The quantity of prime numbers for 2 is 1, thus, we know that the quantity of prime numbers in $2 \cdot P[n^2]$ is odd, whereas the quantity of prime numbers in $P[m^2]$ is even. Thus, they cannot be equal. Note that for this argument the premise that n and m have no common divisor is not necessary.

The use of normal forms and characteristic properties as in the above argument is a common mathematical principle. For example, in polynomial rings over finite fields irreducible polynomials play the role of prime numbers. That is, each polynomial can be expressed as a product of irreducible polynomials. For instance, the irreducible polynomials of grade 1 in $\mathbb{F}_3[x]$ are $x+1$ and $x+2$. Can there be two polynomials $n[x]$ and $m[x]$ in $\mathbb{F}_3[x]$ such that $(x+1) \cdot n[x]^2 = (x+2) \cdot m[x]^2$? The answer is no, and the argument is essentially the same as for integers and their prime number representations. Whereas the quantity of occurrences of the irreducible polynomial $x + 1$ in a normal form representation of $(x + 2) \cdot n[x]^2$ has to be even, it is odd in a normal form representation of $(x + 1) \cdot m[x]^2$.

A similar argument is used in *set theory*, where the reasoning about two sets can sometimes be reduced to their cardinality, that is, a uniform representation of the two sets under scrutiny.

Hence, we have now a general principle and argument used in set theory, number theory and polynomial rings (and probably also other areas of mathematics), from which the argument for the irrationality of $\sqrt{2}$ is just a special instance. We are currently working on methods, tactics and con-

[10] In the sense of coding it into some representational formalism.

trol rules to mechanize this more general approach, and to demonstrate its feasibility in much larger and diverse fields of mathematics.

The general idea here is to represent objects via normal forms and to use projection in order to rewrite these representations until a distinguishing property becomes obvious. For our problem class at hand a distinguishing property can be computed as follows: Compute the prime product normal form of l. This can be done with a computer algebra system. Let $l = (p_1)^{o_1} \cdot \ldots \cdot (p_k)^{o_k}$, that is, o_i is the quantity of occurrences of the prime p_i. Then, there exists an $o \in \{o_1, \ldots, o_k\}$, which is not divisible by j, since otherwise $\sqrt[j]{l}$ would be rational (again the divisibility for each o_i can be checked with a computer algebra system). Let p be the prime number corresponding to o. For an integer x let $\delta_p(x)$ be the number of occurrences of p in the prime product of x. δ_p is a homomorphism with respect to multiplication and addition, that is, $\delta_p(x \cdot y) = \delta_p(x) + \delta_p(y)$. Moreover, we know that $\delta_p(x^y) = y \cdot \delta_p(x)$. The application of δ_p to both sides and repeated applications of these equations rewrite the initial equation as follows:

$$\begin{array}{lll} & l \cdot n^j & = & m^j \\ \Rightarrow & \delta_p(l \cdot n^j) & = & \delta_p(m^j) \\ \Rightarrow & \delta_p(l) + \delta_p(n^j) & = & \delta_p(m^j) \\ \Rightarrow & o + j \cdot \delta_p(n) & = & j \cdot \delta_p(m) \end{array}$$

Now, it is clearly visible that the left hand side of the equation is not divisible by j whereas the right hand side is.

This is a far more general approach and the corresponding methods are certainly more likely candidates for an entry into the international knowledge base on mathematics in the centuries to come.

We are now working on formalizing these methods in rather general terms and then instantiate them with appropriate parameters to the domain in question (number theory, set theory, or polynomial rings) — and the crucial creative step of the system MULTI is then to find the instantiation by some general heuristics.

Acknowledgments

We thank Claus-Peter Wirth and Volker Sorge for their generous support in writing this paper and many fruitful discussions on the proof planning topics raised in this paper. Moreover, we are grateful to the anonymous reviewers for their thought-provoking comments.

This work has been supported partially by the EU training network CALCULEMUS (HPRN-CT-2000-00102) and partially by the Sonderforschungsbereich 378 of the Deutsche Forschungsgemeinschaft DFG.

BIBLIOGRAPHY

[Allen *et al.*, 2000] S. Allen, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. The Nuprl open logical environment. In McAllester [2000].

[Andrews *et al.*, 1996] P. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[Bartle and Sherbert, 1982] R. Bartle and D. Sherbert. *Introduction to Real Analysis.* Wiley, 2nd edition, 1982.

[Baumgartner and Furbach, 1994] P. Baumgartner and U. Furbach. PROTEIN, a PROver with a Theory INterface. In Bundy [1994], pages 769–773.

[Benzmüller and Kohlhase, 1998] C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [1998].

[Benzmüller and Sorge, 1998] C. Benzmüller and V. Sorge. A blackboard architecture for guiding interactive proofs. In Giunchiglia [1998].

[Benzmüller and Sorge, 2000] C. Benzmüller and V. Sorge. Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving. In Kerber and Kohlhase [2000].

[Benzmüller *et al.*, 1999] C. Benzmüller, M. Bishop, and V. Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5:188–207, 1999.

[Benzmüller *et al.*, 2002] C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Irrationality of $\sqrt{2}$ — a case study in ΩMEGA. Seki-Report SR-02-03, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.

[Benzmüller *et al.*, 2003] C. Benzmüller, A. Meier, and V. Sorge. Bridging theorem proving and mathematical knowledge retrieval. In *Festschrift in Honour of Jörg Siekmann's 60s Birthday*, LNAI. Springer, 2003. To appear.

[Benzmüller, 1999] C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving.* PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.

[Bishop and Andrews, 1998] M. Bishop and P. Andrews. Selectively instantiating definitions. In Kirchner and Kirchner [1998].

[Bledsoe, 1990] W. Bledsoe. Challenge problems in elementary calculus. *Journal of Automated Reasoning*, 6:341–359, 1990.

[Bourbaki, 1968] N. Bourbaki. *Theory of sets.* His Elements of mathematics, 1. Paris, Hermann Reading Mass., Addison-Wesley, 1968.

[Bundy *et al.*, 1990] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In M. Stickel, editor, *Proceedings of the 10th Conference on Automated Deduction*, number 449 in LNCS, pages 647–648, Kaiserslautern, Germany, 1990. Springer.

[Bundy, 1988] A. Bundy. The use of explicit plans to guide inductive proofs. In Lusk and Overbeek [1988], pages 111–120.

[Bundy, 1994] A. Bundy, editor. *Proceedings of the 12th Conference on Automated Deduction*, number 814 in LNAI. Springer, 1994.

[Bundy, 2002] A. Bundy. A critique of proof planning. In *Computational Logic: Logic Programming and Beyond*, number 2408 in LNCS, pages 160–177. Springer, 2002.

[Char *et al.*, 1992] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *First leaves: a tutorial introduction to Maple V.* Springer, 1992.

[Cheikhrouhou and Siekmann, 1998] L. Cheikhrouhou and J. Siekmann. Planning diagonalization proofs. In Giunchiglia [1998], pages 167–180.

[Cheikhrouhou and Sorge, 2000] L. Cheikhrouhou and V. Sorge. $\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, Monastir, Tunisia, 22–24 March 2000.

[Church, 1940] A. Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[Coq Development Team, 1999-2003] Coq Development Team. *The Coq Proof Assistant Reference Manual.* INRIA 1999-2003. See `http://coq.inria.fr/doc/main.html`.

[de Nivelle, 1999] H. de Nivelle. Bliksem 1.10 user manual. Technical report, Max-Planck-Institut fr Informatik, 1999.

[Doris, 2001] The Doris system is available at `http://www.cogsci.ed.ac.uk/~jbos/doris/`.

[Drummond, 1994] M. Drummond. On precondition achievement and the computational economics of automatic planning. In *Current Trends in AI Planning*, pages 6–13. IOS Press, 1994.

[Fiedler, 2001a] A. Fiedler. *P.rex*: An interactive proof explainer. In Goré et al. [2001].

[Fiedler, 2001b] A. Fiedler. Dialog-driven adaptation of explanations of proofs. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.

[Fiedler, 2001c] A. Fiedler. *User-adaptive proof explanation*. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R. R. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Franke and Kohlhase, 2000] A. Franke and M. Kohlhase. System description: MBase, an open mathematical knowledge base. In McAllester [2000].

[Ganzinger, 1999] H. Ganzinger, editor. *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI. Springer, 1999.

[Gebhard, 1999] H. Gebhard. Beweisplanung für die Beweise der Vollständigkeit verschiedener Resolutionskalküle in ΩMEGA. Master's thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1999.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, 39:176–210, 572–595, 1935.

[Giunchiglia, 1998] F. Giunchiglia, editor. *Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA '98)*, number 1480 in LNAI. Springer, 1998.

[Gordon and Melham, 1993] M. Gordon and T. Melham. *Introduction to HOL – A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[Goré *et al.*, 2001] R. Goré, A. Leitsch, and T. Nipkow, editors. *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI. Springer, 2001.

[Hadamard, 1944] J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Dover Publications, New York, USA; edition 1949, 1944.

[Hillenbrand *et al.*, 1999] Th. Hillenbrand, A. Jaeger, and B. Löchner. System description: Waldmeister — improvements in performance and ease of use. In Ganzinger [1999], pages 232–236.

[Huang, 1994] X. Huang. Reconstructing Proofs at the Assertion Level. In Bundy [1994], pages 738–752.

[Ireland and Bundy, 1996] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

[Kerber and Kohlhase, 2000] M. Kerber and M. Kohlhase, editors. *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, 2000.

[Kirchner and Kirchner, 1998] C. Kirchner and H. Kirchner, editors. *Proceedings of the 15th Conference on Automated Deduction*, number 1421 in LNAI. Springer, 1998.

[Kirchner and Ringeissen, 2000] H. Kirchner and C. Ringeissen, editors. *Frontiers of combining systems: Third International Workshop, FroCoS 2000*, volume 1794 of *LNAI*. Springer, 2000.

[Kohlhase and Franke, 2001] M. Kohlhase and A. Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems*, 32(4):365–402, September 2001.

[Lusk and Overbeek, 1988] E. Lusk and R. Overbeek, editors. *Proceedings of the 9th Conference on Automated Deduction*, number 310 in LNCS, Argonne, Illinois, USA, 1988. Springer.

[Manthey and Bry, 1988] R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In Lusk and Overbeek [1988], pages 415–434.

[McAllester, 2000] D. McAllester, editor. *Proceedings of the 17th Conference on Automated Deduction*, number 1831 in LNAI. Springer, 2000.

[McCune, 1994] W. W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[McCune, 1997] W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[Meier and Sorge, 2000] A. Meier and V. Sorge. Exploring properties of residue classes. In Kerber and Kohlhase [2000].

[Meier *et al.*, 2001] A. Meier, M. Pollet, and V. Sorge. Classifying Isomorphic Residue Classes. In R. Moreno-Diaz, B. Buchberger, and J.-L. Freire, editors, *A Selection of Papers from the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, number 2178 in LNCS, pages 494–508. Springer, 2001.

[Meier *et al.*, 2002a] A. Meier, E. Melis, and M. Pollet. Towards extending domain representations. Seki Report SR-02-01, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2002.

[Meier *et al.*, 2002b] A. Meier, M. Pollet, and V. Sorge. Comparing Approaches to the Exploration of the Domain of Residue Classes. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, 34(4):287–306, October 2002. Steve Linton and Roberto Sebastiani, eds.

[Meier, 2000] A. Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In McAllester [2000].

[Meier, 2003] A. Meier. *Proof Planning with Multiple Strategies.* PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2003.

[Melis and Meier, 2000] E. Melis and A. Meier. Proof planning with multiple strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagivand, and P. Stuckey, editors, *First International Conference on Computational Logic (CL-2000)*, number 1861 in LNAI, pages 644–659, London, UK, 2000. Springer.

[Melis and Siekmann, 1999] E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, 1999.

[Melis *et al.*, 2000] E. Melis, J. Zimmer, and T. Müller. Integrating constraint solving into proof planning. In Kirchner and Ringeissen [2000].

[Melis, 1996] E. Melis. Island planning and refinement. Seki-Report SR-96-10, Department of Computer Science, Saarland University, Saarbrücken, Germany, 1996.

[Melis, 1998] Erica Melis. AI-techniques in proof planning. In H. Prade, editor, *Proceedings of the 13th European Conference on Artifical Intelligence*, pages 494–498, Brighton, UK, 1998. John Wiley & Sons, Chichester, UK.

[Newell and Simon, 1963] A. Newell and H. Simon. GPS: A program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–296. McGraw-Hill, New York, NY, 1963.

[Nipkow *et al.*, 2002] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Number 2283 in LNCS. Springer, 2002.

[Owre *et al.*, 1996] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in LNCS, pages 411–414, New Brunswick, NJ, 1996. Springer.

[Paulson, 1994] L. Paulson. *Isabelle: A Generic Theorem Prover.* Number 828 in LNCS. Springer, 1994.

[Polya, 1973] G. Polya. *How to Solve it.* Princeton University Press, 1973.

[Riazanov and Voronkov, 2001] A. Riazanov and A. Voronkov. Vampire 1.1 (system description). In Goré et al. [2001].

[Richardson *et al.*, 1998] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λClam. In Kirchner and Kirchner [1998].

[Schönert and others, 1995] M. Schönert et al. *GAP – Groups, Algorithms, and Programming*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

[Siekmann *et al.*, 1999] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11:326–342, 1999.

[Siekmann *et al.*, 2002] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with ΩMEGA. In Voronkov [2002], pages 143–148.

[Sorge, 2000] V. Sorge. Non-Trivial Computations in Proof Planning. In Kirchner and Ringeissen [2000].

[Sorge, 2001] V. Sorge. Ω-ANTS — *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, 2001.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C. Suttner, and T. Yemenis. The TPTP problem library. In Bundy [1994].

[Vo *et al.*, 2003] B. Quoc Vo, C. Benzmüller, and S. Autexier. Assertion application in theorem proving and proof planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. (poster description).

[Voronkov, 2002] A. Voronkov, editor. *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in LNAI. Springer, 2002.

[Weidenbach *et al.*, 1999] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, Th. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In Ganzinger [1999], pages 378–382.

[Weld, 1994] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[Wiedijk, 2002] F. Wiedijk. The fifteen provers of the world. Unpublished Draft, 2002.

[Zhang and Zhang, 1995] J. Zhang and H. Zhang. SEM: A system for enumerating models. In C. S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 298–303, Montreal, Canada, 1995. Morgan Kaufmann, San Mateo, California, USA.

[Zimmer and Kohlhase, 2002] J. Zimmer and M. Kohlhase. System description: The Mathweb Software Bus for distributed mathematical reasoning. In Voronkov [2002], pages 138–142.

Mathematical Domain Reasoning Tasks in Natural Language Tutorial Dialog on Proofs*

Christoph Benzmüller and **Quoc Bao Vo**
Saarland University, Saarbrücken, Germany
chris|bao@ags.uni-sb.de
http://ags.uni-sb.de/~chris|bao

Abstract

We study challenges that are imposed to mathematical domain reasoning in the context of natural language tutorial dialog on mathematical proofs. The focus is on proof step evaluation:

(i) How can mathematical domain reasoning support the resolution of ambiguities and underspecified parts in proof steps uttered by a student?

(ii) How can mathematical domain reasoning support the evaluation of a proof step with respect to the criteria *soundness*, *granularity*, and *relevance*?

Introduction

The final goal of the DIALOG project[1] is a natural tutorial dialog on mathematical proofs between a student and an assistance system for mathematics. Natural language (NL) tutorial dialog on mathematical proofs is a multi-disciplinary scientific challenge situated between (i) advanced NL processing, (ii) flexible tutorial dialog, and (iii) dynamic, abstract level mathematical domain reasoning (MDR[2]). There is still relatively few data available that can guide research in this area. We, therefore, approached the project by using a methodology with a strong initial emphasis on empirical investigations and a top-down modeling of the over-all architecture followed by refinements of the architecture, down to implementation.

First a relevant corpus has been collected and analyzed in the DIALOG project. The phenomena that have been identified through corpus analysis demonstrate, for instance, the need for deep semantical analysis, the importance of a tight integration of NL processing and MDR, and the relevancy of dynamic, abstract-level proof development techniques supporting human-oriented MDR. In particular, the explicit abstract-level representation of proof steps (logically sound or unsound) as uttered by the students is a crucial prerequisite for their subsequent analysis by MDR means in a tutorial dialog setting. Additionally, from a logical point of view, proof steps are highly underspecified (e.g. logically relevant references are left implicit) causing an additional challenge for bridging the gap between NL analysis and MDR.

In this paper we focus on the challenges imposed to MDR:

(i) How can MDR support the resolution of ambiguities and underspecified parts in proof steps uttered by a student?

(ii) How can MDR support the evaluation of a student proof step with respect to the criteria *soundness*, *granularity*, and *relevance*?

In the next section we present an example dialog from our DIALOG corpus and point to some revealed phenomena. We then discuss the MDR challenges from a general viewpoint. Subsequently we present our first concrete approach to solve these challenges. Finally, we discuss some related work and conclude the paper.

Phenomena and Challenges

A Wizard-of-Oz experiment (Dahlbäck, Jönsson, & Ahrenberg 1993) has been performed in the DIALOG project in order to obtain a corpus of tutorial dialogs on mathematical proofs. Twenty four subjects with varying background in humanities and sciences participated in this experiment. Their prior mathematical knowledge ranged from little to fair. The experiment employed typed user and tutor (wizard) input as opposed to spoken language. This experiment and the corpus obtained is discussed in more details in (Wolska *et al.* 2004). The complete corpus comprises 66 recorded dialogs containing on average 12 turns and is available from the DIALOG web-page[3]. It contains 1115 sentences in total, of which 393 are student sentences. An example dialog is shown in Fig. 1.

Investigation of the corpus resulted in an overwhelming list of key phenomena raising interesting and novel research challenges (Benzmüller *et al.* 2003). This was not expected, in particular, because of the simplicity of the mathematical domain (naive set theory) chosen for this experiment. Many

*We thank Manfred Pinkal, Jörg Siekmann, Ivana Kruijff-Korbayová, Armin Fiedler, Magdalena Wolska, Helmut Horacek, Serge Autexier, Dimitra Tsovaltzis, Marvin Schiller, and Mark Buckley.

[1]The DIALOG project is a collaboration between the Computer Science and Computational Linguistics departments of Saarland University as part of the Collaborative Research Center on *Resource-Adaptive Cognitive Processes*, SFB 378 (`http://www.coli.uni-saarland.de/projects/sfb378/`).

[2]We use 'MDR' in the remainder as an abbreviation for both 'mathematical domain reasoning' and 'mathematical domain reasoner'; the precise meaning will be clear in each context.

[3]`http://www.ags.uni-sb.de/~chris/dialog/`

T1: Bitte zeigen Sie: $K((A \cup B) \cap (C \cup D)) = (K(A) \cap K(B)) \cup (K(C) \cap K(D))$! *[Engl.: Please show: $K((A \cup B) \cap (C \cup D)) = (K(A) \cap K(B)) \cup (K(C) \cap K(D))$!]*

S1: nach deMorgan-Regel-2 ist $K((A \cup B) \cap (C \cup D)) = (K(A \cup B) \cup K(C \cup D))$. *[Engl.: by deMorgan-Rule-2 $K((A \cup B) \cap (C \cup D)) = (K(A \cup B) \cup K(C \cup D))$ holds.]*

T2: Das ist richtig! *[Engl.: This is correct!]*

S2: $K(A \cup B)$ ist laut deMorgan-1 $K(A) \cap K(B)$ *[Engl.: $K(A \cup B)$ is $K(A) \cap K(B)$ according to deMorgan-1]*

T3: Das stimmt auch. *[Engl.: That is also right.]*

S3: und $K(C \cup D)$ ist ebenfalls laut deMorgan-1 $K(C) \cap K(D)$ *[Engl.: and $K(C \cup D)$ is also $K(C) \cap K(D)$ according to deMorgan-1]*

T4: Auch das stimmt. *[Engl.: That also is right.]*

S4: also folgt letztendlich: $K((A \cup B) \cap (C \cup D)) = (K(A) \cap K(B)) \cup (K(C) \cap K(D))$. *[Engl.: hence follows finally: $K((A \cup B) \cap (C \cup D)) = (K(A) \cap K(B)) \cup (K(C) \cap K(D))$.]*

T5: Das stimmt ... *[Engl.: This is correct ...]*

Figure 1: An example dialog. **T** and **S** mark tutor (i.e. wizard) and student turns respectively. 'K' refers to the 'set complement' relation. German has been the language of choice.

of the identified phenomena are relevant not only for the tutorial NL dialog context but have a much wider impact for NL interactions in human-oriented theorem proving. This paper focuses on phenomena that are relevant for MDR:

Notion of Proof. For analyzing the notion of human-oriented mathematical proofs, primarily shaped-up textbook proofs have been investigated in the deduction systems community (Zinn 2004). The DIALOG corpus provides an important alternative view on it, since textbook proofs neither reveal the actual dynamics of proof construction nor do they show the weaknesses and inaccuracies of the student's utterances, i.e., the student's proof step directives. The corpus also illustrates the style and logical granularity of human-constructed proofs. The style is mainly declarative, for example, the students declaratively described the conclusions and some (or none) of the premises of their inferences. This is in contrast to the procedural style employed in many proof assistants where proof steps are invoked by calling rules, tactics, or methods, i.e., some proof refinement procedures.

The hypothesis that assertion level reasoning (Huang 1994) plays an essential role in this context has been confirmed. The phenomenon that assertion level reasoning may by highly underspecified in human-constructed proofs, however, is a novel finding (Autexier *et al.* 2003).

Underspecification is a well known phenomenon in linguistic analysis. The corpus reveals that underspecification also occurs in the content and precision of mathematical utterances (proof step specification) and thus carries over to MDR. Interestingly underspecification also occurs in shaped-up textbook proofs but has only very recently been addressed (Zinn 2004). To illustrate the underspecification aspect we use example utterance **S4** in Fig. 1: Utterance **S4** is logically strongly underspecified. Here, it is neither mentioned from what assertion(s) in the discourse this statement exactly follows nor how these assertions are used. However, such detailed information is typically required in proof assistants to execute the student's proof step directive, i.e., to 'understand' and 'logically follow' the student's argumentation.

Proof Step Evaluation (PSE) is an interesting novel application for theorem proving systems. A (next) proof step uttered by a student within a tutorial context has to be analyzed with respect to the following criteria:

Soundness: Can the proof step be reconstructed by a formal inference system and logically and tutorially verified?

Granularity: Is the 'argumentative complexity' or 'size' of the proof step logically and tutorially acceptable?

Relevance: Is the proof step logically and tutorially useful for achieving the goal?

Resolution of underspecification and PSE motivate a specific module supporting these tasks in tutorial NL dialog on proofs; in the remainder we call such a module *proof manager (PM)*.

MDR Challenges from a General Viewpoint

Ambiguity and Underspecification Resolution The corpus reveals that ambiguities may arise at different phases of processing between the linguistic analysis and MDR. Consider, for instance, the following student utterance:

S: A enthaelt B [*Engl.*: A contains B]

In this utterance 'enthaelt' ('contains') is ambiguous as it may refer to the set relations 'element-of' and 'subset-of'. The ambiguity arises during linguistic analysis. It can be resolved, for instance, by type-checking provided that type information on A and B is available: if both symbols are of the same 'set type' then 'enthaelt' means 'subset-of'. However, type checking cannot differentiate between '$\subset$' and '$\subseteq$' as potential readings. The phenomenon is even better illustrated by the following two utterances in which important bracketing information is missing ('K' refers to the 'set complement' operation and 'P' to the 'Power set' operation):

S': $P((A \cup C) \cap (B \cup C)) = PC \cup (A \cap B)$

S": $K((A \cup C) \cap (B \cup C)) = KC \cup (A \cap B)$

In **S'** type information (if available) can be employed to rule out the reading $P(C) \cup (A \cap B)$ for the term to the right. However, type information is not sufficient to differentiate between the readings $K(C) \cup (A \cap B)$ and $K(C \cup (A \cap B))$ in **S"**. Here only MDR can detect that the first reading leads to a logically wrong statement and the second reading to a correct one. As we cannot assume that the domain model statically represents all correct mathematical statements this calls for dynamic MDR support in the resolution of ambiguities that, as given here, may arise during linguistic analysis. Now consider the following slight modification (wrt. reference to deMorgan rule) of utterances **T1** and **S1** from Fig. 1.

T1: Please show : $K((A \cup B) \cap (C \cup D)) = (K(A) \cap K(B)) \cup (K(C) \cap K(D))$

S1': by the deMorgan rule we have $K((A \cup B) \cap (C \cup D)) = (K(A \cup B) \cup K(C \cup D))$.

S1' does not lead to an ambiguity during linguistic analysis. It nevertheless leads to an ambiguity in the domain reasoner since the suggested proof step is highly underspecified from

Proof State	**Some Student Utterances**
(A1) $A \wedge B$. (A2) $A \Rightarrow C$. (A3) $C \Rightarrow D$. (A4) $F \Rightarrow B$. (G) $D \vee E$.	(a) From the assertions follows D. (b) B holds. (c) It is sufficient to show D. (d) We show E.

Figure 2: PSE example scenario: (A1)-(A4) are assertions that have been introduced in the discourse and that are available to prove the proof goal (G). (a)-(d) are examples for possible proof step directives of the student in this proof situation.

a proof construction viewpoint: **S1'** can be obtained directly from the deMorgan rule $\forall X, Y.K(X \cap Y) = K(X) \cup K(Y)$ (denoted as `deMorgan-2`) by instantiating X with $(A \cup B)$ and Y with $(C \cup D)$. Alternatively it could be inferred from **T1** when applying deMorgan rule $\forall X, Y.K(X \cup Y) = K(X) \cap K(Y)$ (denoted as `deMorgan-1`) from right to left to the subterms of **T1**: $K(A) \cap K(B)$ and $K(C) \cap K(D)$. Differentiating between such alternatives could be crucial in tutoring mathematical proofs.

Proof Step Evaluation: PSE supports the dynamic step-by-step analysis (with criteria soundness, granularity, relevance) of the proof constructed by the student. All three criteria have a pure logical dimension and additionally a tutorial dimension. For instance, a proof step may be formally relevant by pure logical means but it may be considered as not relevant when additional tutorial aspects are taken into account. On the other hand, a student utterance which is sufficiently close to a valid next proof step may be considered tutorially relevant while being logically irrelevant. In this paper we mainly focus on the logical dimension; the hypothesis is that their solution is one important prerequisite for solving the general PSE problem involving also the tutorial dimension. Much further research in this direction is clearly needed. The PSE challenge will now be further illustrated using the artificially simplified example in Fig. 2.

Soundness: Determining whether an uttered proof step is sound requires that the MDR can represent, reconstruct and validate the uttered proof step (including all the justifications used by the student) within the MDR's representation of the proof state. Consider, for instance, utterance (a) in Fig. 2: Verification of the soundness of this utterance boils down to adding D as a new assertion to the proof state and to proving that: **(P1)** $(A \wedge B), (A \Rightarrow C), (C \Rightarrow D), (F \Rightarrow B) \vdash D$. Solving this proof task confirms the logical soundness of utterance (a). If further explicit justifications are provided in the student's utterance (e.g. a proof rule) then we have to take them into consideration and, for example, prove **(P1)** modulo these additional constraints. Soundness is a fairly tractable criterion for which different techniques are readily available (Zinn 2004). PSE with respect to the criteria *granularity* and *relevance*, however, is novel and challenging.

Granularity evaluation requires analyzing the 'complexity' or 'size' of proofs instead of asking for the mere existence of proofs. For instance, evaluating utterance (a) above boils down to judging the complexity of the generated proof task **(P1)**. Let us, for example, use Gentzen's natural deduction (ND) calculus as the proof system $\vdash$. As a first and naive logical granularity measure, we may determine the number of $\vdash$-steps in the smallest $\vdash$-proof of the proof task for the proof step utterance in question; this number is taken as the argumentative complexity of the uttered proof step. For example, the smallest ND proof for utterance (a) has '3' proof steps: we need one 'Conjunction-Elimination' step to extract A from $A \wedge B$, one 'Modus Ponens' step to obtain C from A and $A \Rightarrow C$, and another 'Modus Ponens' step to obtain D from C and $C \Rightarrow D$. On the other hand, the smallest ND proof for utterance (b) requires only '1' step: B follows from assertion $A \wedge B$ by 'Conjunction-Elimination'. If we now fix a threshold that tries to capture, in this sense, the 'maximally acceptable size of an argumentation' then we can distinguish between proof steps whose granularity is acceptable and those which are not. This threshold may be treated as a parameter determined by the tutorial setting. However, the ND calculus together with naive proof step counting doesn't always provide a cognitively adequate basis for granularity analysis. The reason is that two intuitively very similar student proof steps (such as **(i)** from $A = B$ and $B = C$ infer $A = C$ and **(ii)** from $A \Leftrightarrow B$ and $B \Leftrightarrow C$ infer $A \Leftrightarrow C$) may actually expand into base-level ND proofs of completely different size. Also related literature has pointed out that standard ND calculus does not adequately reflect human-reasoning (Rips 1994). This problem could become even worse if we chose a machine-oriented calculus such as resolution. Two important and cognitively interesting questions thus concern the appropriate choice of a proof system $\vdash$ and ways to measure the 'argumentative complexity' of a proof step.

Relevance. Relevance asks questions about the usefulness and importance of a proof step with respect to the original proof task. For instance, in utterance (c) the proof goal $D \vee E$ is refined to the new proof goal D using backward reasoning, i.e., the previously open goal $D \vee E$ is closed and justified by a new goal. Answering the logical relevance question in this case requires to check whether a proof can still be generated in the new proof situation. In our case, the task is thus identical to proof task **(P1)**. A backward proof step that is not relevant according to this criterion is (d) since it reduces to the proof task: **(P2)** $(A \wedge B), (A \Rightarrow C), (C \Rightarrow D), (F \Rightarrow B) \vdash E$ for which no proof can be generated. Thus, (d) is a sound refinement step that is not relevant. This simple approach appears plausible but needs to be refined. The challenge is to exclude detours and to take tutorial aspects into account (in a tutorial setting we are often interested in teaching particular styles of proofs, particular proof methods, etc.). This also applies to the more challenging forward reasoning case to identify that, for instance, utterance (b) describes a non-relevant proof step.

Relevance and granularity are interesting, ambitious and important challenges for tutoring of proofs. To address these problems, it's not sufficient to merely establish the existence of proofs but the system has to construct proofs with particular properties. It may be the case that evaluating different criteria requires different 'suitable' theorem provers.

Moreover, the system also needs to closely mirror and reflect reasoning steps as they are typically performed by humans. Generally, the system will need to adapt to the capabilities of individual students and the requirements of varying tutorial settings.

PSE in the DIALOG Demonstrator

We have implemented a demonstrator version of a PM which provides dynamic support for resolution of underspecification and PSE based on heuristically guided abstract-level MDR realized on top of the ΩMEGA-CORE framework (Autexier 2003). The PM has been integrated into the overall demonstrator of the DIALOG project in which it communicates with other components of the system including the linguistic analyzer, the dialog manager, the tutorial manager, and the NL generator. More information on the role of the PM in the DIALOG demonstrator system and on its interplay with other modules is given in (Buckley & Benzmüller 2005). Note that we do not address tutoring aspects directly in the PM. Instead the result of the PM's proof step analysis is passed to the *tutorial manager* which then proposes a tutoring move to the dialog manager of the overall system. Tutoring aspects of the DIALOG project are discussed in (Fiedler & Tsovaltzi 2003).

The complete system has been applied to several example dialogs from the DIALOG corpus and it has been demonstrated in the course of the evaluation of the DIALOG project that the system is particularly able to support variations of the dialog presented in Fig.1 (which we will use for illustration purposes). However, our system is currently only applicable to a very restricted subset of example proofs in naive set theory. For these examples the PM's computation costs are acceptable. It remains to be seen whether this is still the case when moving to less elementary mathematical problem domains.

Proof Step Representation and Resolution of Underspecification. The PM needs to "understand" the incoming student proof step and to fit it into the current proof context.

In our implementation, the student proof step is first formatted into a tuple ⟨ LABEL, TYPE, DIR, FORMULA, JUSTIFICATION-LIST ⟩: LABEL provides a reference to this proof step. TYPE indicates whether the student proof step is, for example, an *inference* step, a *variable assignment*, or a *local hypothesis introduction* (these are the options we currently support). Given the proof step type *inference*, DIR indicates the direction of this step as linguistically extracted from the student's utterance. The alternatives are *forward*, *backward*, *sideward*, and *closing*. For instance, when the student asserts that "ϕ follows from ψ and θ" and if we know that ψ and θ are the two premises of the current proof task, then the input analyzer should be able to assign forward inference to DIR. FORMULA is the asserted formula in this proof step, e.g., the ϕ from above. JUSTIFICATION-LIST contains all the information the student uses to justify FORMULA.

In our current approach, all of these fields except from FORMULA can be left underspecified (i.e. empty). LABEL can in general be easily generated by referring to FORMULA or by NL references such as "the previous proof step", "your second proof step", etc. The other fields are usually more ambitious to determine. Before we proceed with describing our solution to underspecification resolution, we elaborate the JUSTIFICATION-LIST. JUSTIFICATION-LIST is a list $(J_1, \ldots, J_n)$ of justifications J_i (for $0 \leq i \leq n$). When $n = 0$ then JUSTIFICATION-LIST is underspecified. Each justification J_i is a tuple ⟨NAME, FORM, SUBST⟩: NAME refers to an assertion. It can be the label of a previous proof step or of an assertion in a mathematical knowledge base, for example, '*deMorgan-2*'. FORM is a formula used to justify the asserted proof step. For instance, instead of referring to deMorgan-2, the student may say: "Since $\overline{A \cap (B \cup C)} = \overline{A} \cup \overline{B \cup C}$, from $\Phi[\overline{A \cap (B \cup C)}]$ we obtain $\Phi[\overline{A} \cup \overline{B \cup C}]$." SUBST is an explicitly mentioned instantiation of variables the student has applied in the proof step.

All justifications fields can be left underspecified. The field SUBST has been introduced mainly for the purpose of exhaustively capturing the student input in our representation. Given an underspecified justification ⟨NAME, FORM, SUBST⟩, FORM is generally equivalent to $dereference(\text{NAME}) + \text{SUBST}$. Assume, for example, that we already have information on FORM := $\overline{A \cap (B \cup C)} = \overline{A} \cup \overline{B \cup C}$. The PM can determine a possible assertion which has been used (e.g. *deMorgan-2*) together with the substitution the student has applied (here $[A \mapsto X, (B \cup C) \mapsto Y]$). In fact, in most proof step utterances in the DIALOG corpus the student justifies her proof step with a reference to the employed assertion NAME and by specifying the inferred formula FORMULA: For instance, a student may say: "By *deMorgan-2*, we have Φ". Unification and heuristically guided theorem proving is employed in the PM to support the analysis and completion of different combinations of given and missing information in justifications. Problematic cases typically arise when the student leaves the justification for her proof step underspecified altogether.

The proof step representation language presented here is the one that has been implemented in the PM. In the meantime this language has been further developed in theory (Autexier *et al.* 2003).

Example 1 The underspecified proof step **S1** in the example dialog (see Fig. 1) is represented in the PM as follows:[4]

```
(input (label 1_1)
  (formula (= (C (N (U a b) (U c d)))
              (U (C (U a b)) (C (U c d)))))
  (type ?)
  (direction ?)
  (justifications
    (just   (reference deMorgan-2)
            (formula ?)
            (substitution ?))))
```

Our PM employs the ΩMEGA-CORE calculus (Autexier 2003) as a sound and complete base framework (for classical higher-order reasoning) to support resolution of underspecification and PSE. The internal proof representation of the

[4] `C`, `N`, and `U` stand for `complement`, `intersection`, and `union`, respectively. `?` denotes underspecification.

PM is based on *task structures* which are defined on top of the ΩMEGA-CORE calculus; for more details on this proof representation framework we refer to (Hübner *et al.* 2004).

In some sense, tasks resemble and generalize sequents in sequent calculi. Proof construction in this "ΩMEGA-CORE + tasks"-framework employs and generalizes well-known techniques in tableau-based theorem proving (cf. (Hähnle 2001) and the references therein) and the matrix method (Andrews 1981; Bibel 1983). See also (Vo, Benzmüller, & Autexier 2003) for further details.

We present two example strategies employed by the PM to relate the student proof step to the PM's internal representation of the current proof state and to formally reconstruct it in order to determine missing information.

Justify by a unifiable premise: The system looks for subterms of the premises of the present task and for subterms of the available assertions in a knowledge base which are unifiable to the student proof step. Such a justification may require further conditions to be discharged. These conditions are extracted with the help of the ΩMEGA-CORE framework and they form additional proof obligations which are analyzed by an automated theorem prover.

Justify by equivalence transformation and equality reasoning: This case is a generalization of the above one in the sense that the asserted formula does only follow via equivalence transformation and equality reasoning from the premises and assertions available in the proof state. For this strategy we employ a specifically adapted tableau-based reasoner implemented within the ΩMEGA-CORE framework.

Example 1 (contd.) Our simple example illustrates the above strategies:

1. The asserted formula in the student proof step is unifiable at top-level with the `deMorgan-2` rule. Thus, we recompute a *forward* proof step:

   $$\overline{(A \cup B) \cap (C \cup D)} = (\overline{A \cup B}) \cup (\overline{C \cup D})$$

   is obtained by `deMorgan-2` using the *substitution*:

   $$[X \mapsto (A \cup B); Y \mapsto (C \cup D)]$$

2. On the other hand, our system is able to identify the discrepancies between the asserted formula and the goal formula of the current proof task. Identifying a possible *backward* reasoning step the system thus carries out the following transformation:

   $$\overline{(A \cup B) \cap (C \cup D)} = (\overline{A} \cap \overline{B}) \cup (\overline{C} \cap \overline{D})$$

   is reduced to the new goal formula

   $$\overline{(A \cup B) \cap (C \cup D)} = (\overline{A \cup B}) \cup (\overline{C \cup D})$$

   by rewriting the subterms: $(\overline{A} \cap \overline{B})$ and $(\overline{C} \cap \overline{D})$ with the subterms $(\overline{A \cup B})$ and $(\overline{C \cup D})$, respectively, using the rule `deMorgan-1`.

For the initially underspecified input proof step representation we have thus computed two possible fully specified logical interpretations.

Proof Step Evaluation The PM is now facing the problem of evaluating both identified proof step interpretations along the PSE criteria. Note that soundness has already been partly addressed during the above phase, since we were able to reconstruct the underspecified proof step in at least one way in the current proof state.

Employing heuristically guided theorem proving techniques, our PM finally identifies the following ratings and annotations for our two proof step interpretations:[5]

1.
```
(evaluation
  (reference ...)
  (formula (= (C (N (U a b) (U c d)))
              (U (C (U a b)) (C (U c d)))))
  (substitution ((x (U a b) y (U c d))))
  (direction FORWARD)
  (justification DeMorgan-2)
  (soundness 1)
  (relevance 0.9)
  (granularity 1))
```
2.
```
(evaluation
  (reference ...)
  (formula (= (C (N (U a b) (U c d)))
              (U (C (U a b)) (C (U c d)))))
  (substitution ...)
  (direction BACKWARD)
  (justification (((C (U c d)) . (N (C c) (C d)))
                 ((C (U a b)) . (N (C a) (C b)))))
  (soundness 1)
  (relevance 0.9)
  (granularity 0.5))
```

The overall system then determines a preference for interpretation (1.) since it shares the justification used by the student, *viz.* the rule `deMorgan-2`. Furthermore, the former inference is considered to be granularly more appropriate than the latter. This is because the former employs only one application of the rule `deMorgan-2` while the latter applies the rule `deMorgan-1` twice. As discussed in the previous section, this is generally an over-simplified way to determine the relative granularity of a proof step. A more precise, separate soundness investigation in the PSE phase would also rule out interpretation (2.), provided that the students explicit reference to deMorgan-2 is taken into account.

Further Proof Management Tasks It is important that the system and the student share a mutual understanding about the situation they are confronting. And we have already motivated that the system should be capable of adequately representing the context and the situation in which the student is currently operating and reasoning about. Generally, we consider different classes of situations. Two examples are:

Problem-solving situations: In these situations, alternative problem solving strategies are considered to tackle the problem, e.g. looking for similar problems whose solutions

[5] The ellipses indicate that the field refers to some internal representation which is left out to save space. Note also that the *relevance* rating for both interpretations is 0.9 to allow a margin for error unless the proof step is found to be used in *every* possible proofs in which case the *relevance* rating will be 1.

are known, finding a lemma whose application could bridge the gap between the premises and the goal, searching for applicable proving methods such as *proof by induction*, *diagonalization proof*, etc.

Proof situations: Once a student proof step has been identified as related to an available proof situation in the maintained proof history, a new *current proof situation* is computed and updated into the proof history. The current proof situation consists of the "relevant" proof fragments which have been identified up to this point.

The tasks of reconstructing theorem prover-oriented proof fragments from the student proof steps, organizing the relevant proof fragments into (partial) proofs, keeping track of the proof history and other relevant information for future backtracking, etc. are all handled by the PM. It's also important to note that while the problems of resolving underspecification and PSE have been discussed separately, they are solved in combination since they are mutually dependent.

In general, judging the student's utterances in a mathematics tutoring scenario is a very complex task addressing many AI problems including NL understanding, plan recognition, ambiguity resolution, step-wise proof construction, management of proofs, etc. In our first implementation of the PM, we clearly had to make several simplifications which can later be generalized if future experiments indicate the need for this. We give some examples:

Granularity and the Direction of Inference: If the direction of an inference is not made explicit by the student, the PM tries to determine it by considering the granularity of the proof justifying a forward reasoning step and the granularity of the proof justifying a backward directed goal reduction step; cf. our example from before. If the former is considered to be more difficult than the latter, the system conjectures that this proof step is a forward proof step; otherwise, it is considered to be a backward proof step.

Student Modeling: The granularity of a proof step is relative to the student's knowledge and expertise in the domain under consideration. In the present implementation, the student model and other relevant information have not been taken into account when appraising the student proof step.

Related Work

Empirical findings in the area of intelligent tutoring show that flexible natural language dialog supports active learning (Moore 1993). In the DIALOG project, therefore, the focus has been on the development of solutions allowing flexible dialog. However, little is known about the use of natural language in dialog settings in formal domains, such as mathematics, due to the lack of empirical data.

Input analysis in dialog systems is for most domains commonly performed using shallow syntactic analysis combined with keyword spotting; slot-filling templates, however, are not suitable in our case. Moreover, tight interleaving of natural and symbolic language makes key-phrase spotting difficult because of the variety of possible verbalizations. Statistical methods are employed in tutorial systems to compare student responses with a domain-model built from preconstructed gold-standard answers (Graesser *et al.* 2000). In our context, such a static domain-modeling solution is impossible because of the wide quantitative and qualitative range of acceptable proofs, i.e., generally, our set of gold-standard answers is even infinite.

Related work with regard to interpreting mathematical texts is (Zinn 2004) which analyzes comparably complete, carefully structured textbook proofs, and relies on given text-structure, typesetting and additional information that identifies mathematical symbols, formulae, and proof steps. With respect to our goal of ambiguity and underspecification resolution, (Bos 2003) provides an algorithm for efficient presupposition and anaphora resolution which uses state-of-the-art traditional automated theorem provers for checking consistency and informativeness conditions.

Recent research into dialog modeling has delivered a variety of approaches more or less suitable for the tutorial dialog setting. For instance, scripting is employed in Autotutor (Person *et al.* 2000) and knowledge construction dialogs are implemented in Geometry Tutor (Matsuda & VanLehn 2003). Outside the tutorial domain, the framework of Information State Update (ISU) has been developed in the EU projects TRINDI[6] and SIRIDUS[7] (Traum & Larsson 2003), and applied in various projects targeting flexible dialog. An ISU-based approach with several layers of planning is used in the tutorial dialog system BEETLE (Zinn *et al.* 2003).

Finally, the dialogs in our corpus reveal many challenges for human-oriented theorem proving. Traditional automated theorem provers (e.g. OTTER and Spass) work on a very fine-grained logic level. However, interactive proof assistants (e.g. PVS, Coq, NuPRL, Isabelle) and in particular proof planners (e.g. OMEGA and λClam) support abstract-level reasoning. The motivation for abstract-level reasoning is twofold: (a) to provide more adequate interaction support for the human and (b) to widen the spectrum of mechanizable mathematics. Proof assistants are usually built bottom-up from the selected base-calculus; this often imposes constraints on the abstract-level reasoning mechanisms and the user-interface.

Conclusion

We have identified novel challenges and requirements to MDR in the context of tutorial NL dialogs on mathematical proofs. For instance, we must be able to explicitly represent and reason about ambiguous and underspecified student proof steps in the PM. The represented proof steps may be unsound, of unacceptable granularity or not relevant. The analysis of these criteria is then the task of PSE. Generally, resolution of underspecification and PSE are mutually dependent. Except for pure logical soundness validation of proof steps, none of these requirements can currently be easily supported within state-of-the-art theorem provers. Thus, novel and cognitively interesting challenges are raised to the deduction systems community.

PSE can principally be supported by different approaches — including ones that avoid dynamic theorem proving as

[6] http://www.ling.gu.se/research/projects/trindi/

[7] http://www.ling.gu.se/projekt/siridus/

presented in this paper. We list some alternative approaches according to increasing difficulty:

1. We could statically choose one or a few 'golden proofs' and match the uttered partial proofs against them.
2. We first generate from the initially chosen golden proofs larger sets modulo, for instance, (allowed) re-orderings of proof steps and match against this extended set.
3. We dynamically support PSE with heuristically guided abstract-level MDR.
4. We interpret the problem as challenge to proof theory and try to develop a proper proof theoretic approach to differentiate between 'tutorially good proofs and proof steps' and 'tutorially less good proofs and proof steps' in the space of all proofs for a given problem.

The space of all proofs that solve a proof problem is generally infinite which is one reason why a static modeling of finitely many 'golden solutions' as in approaches (1) and (2) is generally insufficient in our context. Approach (3) is our currently preferred choice and a first, still rather naive, approach to the logical dimension of this challenge has been presented in this paper. Much further research is clearly needed. Approach (4) is the approach we want to additionally investigate in the future; some relevant related work in proof theory to capture a notion of good proofs is presented in (Dershowitz & Kirchner 2003).

For (3) we have developed a heuristically guided MDR tool that is capable of representing, constructing and analyzing proofs at the assertion level. In the first place these proofs maybe sound or non-sound. For naive set theory (our mathematical domain of choice so far) this tool has been able to reconstruct and represent student proofs at the same level of argumentative complexity as given in the DIALOG corpus. We conjecture that this is a basic requirement for PSE in tutorial settings. We have also shown how (in the same mathematical domain) our PM resolves ambiguities and underspecification in the student input and how it evaluates the student input along the three major dimensions of *soundness*, *relevance*, and *granularity*. The application of our approach to more challenging mathematical domains and its evaluation therein is future work.

References

Andrews, P. B. 1981. Theorem proving via general matings. *J. of the ACM* 28(2):193–214.

Autexier, S.; Benzmüller, C.; Fiedler, A.; Horacek, H.; and Vo, B. Q. 2003. Assertion-level proof representation with underspecification. *ENTCS* 93:5–23.

Autexier, S. 2003. *Hierarchical Contextual Reasoning*. Ph.D. Dissertation, Saarland University, Germany.

Benzmüller, C.; Fiedler, A.; Gabsdil, M.; Horacek, H.; Kruijff-Korbayová, I.; Pinkal, M.; Siekmann, J.; Tsovaltzi, D.; Vo, B. Q.; and Wolska, M. 2003. Tutorial dialogs on mathematical proofs. In *Proc. of the IJCAI 03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*.

Bibel, W. 1983. *Automate Theorem Proving*. Friedr. Vieweg.

Bos, J. 2003. Implementing the the binding and accomodation theory for anaphora resolution and presupposition projection. *Computational Linguistics*.

Buckley, M., and Benzmüller, C. 2005. System description: A dialog manager supporting tutorial natural language dialogue on proofs. In *Proc. of the ETAPS Satellite Workshop on User Interfaces for Theorem Provers (UITP)*.

Dahlbäck, N.; Jönsson, A.; and Ahrenberg, L. 1993. Wizard of oz studies – why and how. *Knowledge-Based Systems* 6(4):258–266.

Dershowitz, N., and Kirchner, C. 2003. Abstract saturation-based inference. In *Proc. of LICS 2003*. Ottawa, Ontario: ieee.

Fiedler, A., and Tsovaltzi, D. 2003. Automating hinting in an intelligent tutorial dialog system for mathematics. In *Proc. of the IJCAI 03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*.

Graesser, A.; Wiemer-Hastings, P.; Wiemer-Hastings, K.; Harter, D.; and Person, N. 2000. Using latent semantic analysis to evaluate the contributions of students in autotutor. *Interactive Learning Environments* 8.

Hähnle, R. 2001. Tableaux and related methods. In Robinson, A., and Voronkov, A., eds., *Handbook of Automated Reasoning*, volume I. Elsevier Science. chapter 3, 101–176.

Huang, X. 1994. Reconstructing Proofs at the Assertion Level. In *Proc. of CADE-12*, number 814 in LNAI, 738–752. Springer.

Hübner, M.; Autexier, S.; Benzmüller, C.; and Meier, A. 2004. Interactive theorem proving with tasks. *ENTCS* 103(C):161–181.

Matsuda, N., and VanLehn, K. 2003. Modelling hinting strategies for geometry theorem proving. In *Proc. of the 9th International Conference on User Modeling*.

Moore, J. 1993. What makes human explanations effective? In *Proc. of the 15th Annual Conference of the Cognitive Science Society*.

Person, N. K.; Graesser, A. C.; Harter, D.; Mathews, E.; and the Tutoring Research Group. 2000. Dialog move generation and conversation management in AutoTutor. In *Building Dialog Systems for Tutorial Applications—Papers from the AAAI Fall Symposium*. North Falmouth, MA: AAAI press.

Rips, L. J. 1994. *The psychology of proof*. MIT Press, Cambridge, Mass.

Traum, D. R., and Larsson, S. 2003. The information state approach to dialogue management. In van Kuppevelt, J., and Smith, R., eds., *Current and New Directions in Discourse and Dialogue*. Kluwer. `http://www.ict.usc.edu/~traum/Papers/traumlarsson.pdf`.

Vo, Q. B.; Benzmüller, C.; and Autexier, S. 2003. Assertion application in theorem proving and proof planning. In *Proc. of IJCAI-03*. IJCAI/Morgan Kaufmann.

Wolska, M.; Vo, B. Q.; Tsovaltzi, D.; Kruijff-Korbayová, I.; Karagjosova, E.; Horacek, H.; Gabsdil, M.; Fiedler, A.; and Benzmüller, C. 2004. An annotated corpus of tutorial dialogs on mathematical theorem proving. In *Proc. of LREC*.

Zinn, C.; Moore, J. D.; Core, M. G.; Varges, S.; and Porayska-Pomsta, K. 2003. The be&e tutorial learning environment (beetle). In *Proc. of Diabruck, the 7th Workshop on the Semantics and Pragmatics of Dialogue*.

Zinn, C. 2004. *Understanding Informal Mathematical Discourse*. Ph.D. Dissertation, University of Erlangen-Nuremberg.

An Agent-based Architecture for Dialogue Systems*

Mark Buckley and Christoph Benzmüller
{markb|chris}@ags.uni-sb.de

Dept. of Computer Science, Saarland University

Abstract. Research in dialogue systems has been moving towards reusable and adaptable architectures for managing dialogue execution and integrating heterogeneous subsystems. In this paper we present a formalisation of ADMP, an agent-based architecture which supports the development of dialogue applications. It features a central data structure shared between software agents, it allows the integration of external systems, and it includes a meta-level in which heuristic control can be embedded.

1 Introduction

Research in dialogue systems has been moving towards reusable and adaptable architectures for managing dialogue execution and integrating heterogeneous subsystems. In an architecture of this type, different theories of dialogue management can be formalised, compared and evaluated. In this paper we present a formalisation of ADMP[1], an architecture which uses software agents to support the development of dialogue applications. It features a central data structure shared between agents, it allows the integration of external systems, and it includes a meta-level in which heuristic control can be embedded.

We have instantiated the system to support dialogue management. Dialogue management involves maintaining a representation of the state of a dialogue, coordinating and controlling the interplay of subsystems such as domain processing or linguistic analysis, and deciding what content should be expressed next by the system. ADMP applies the information state update (ISU) approach to dialogue management [1]. This approach uses an information state as a representation of the state of the dialogue, as well as update rules, which update the information state as the dialogue progresses. The ISU approach supports the formalisation of different theories of dialogue management.

The framework of our research is the DIALOG project[2], which investigates flexible natural language dialogue in mathematics, with the final goal of natural tutorial dialogue between a student and a mathematical assistance system. In

* This work was supported by the DAAD (German Academic Exchange Service), grant number A/05/05081 and by the DFG (Deutsche Forschungsgemeinschaft), Collaborative Research Centre 378 for Resource-adaptive Cognitive Processes.

[1] The Agent-based Dialogue Management Platform

[2] http://www.ags.uni-sb.de/dialog/

the course of a tutorial session, a student builds a proof by performing utterances which contain proof steps, thereby extending the current partial proof. The student receives feedback from the DIALOG system after each proof step. This feedback is based on the computations and contribution of numerous systems, such as a domain reasoner or a natural language analysis module. The integration of these modules and the orchestration of their interplay as well as the selection of a next dialogue move which generates the feedback is the task of the dialogue manager.

The work presented in this paper is motivated by an initial prototype dialogue manager for the DIALOG demonstrator [2]. After its development we were able to pinpoint some features which we consider necessary for the DIALOG system, and which the platform presented here supports. The overall design of ADMP is influenced by the design of Ω-Ants [3], a suggestion mechanism which supports interactive theorem proving and proof planning. It uses societies of software agents, a blackboard architecture, and a hierarchical design to achieve concurrency, flexibility and robust distributed search in a theorem proving environment.

Although ADMP has been developed to support dialogue systems, it can be seen as a more general architecture for collaborative tasks which utilise a central data store. For example, we have used ADMP to quickly implement a lean prototype resolution prover for propositional logic.

Our work is related to other frameworks for dialogue management such as TrindiKit, a platform on top of which ISU based dialogue applications can be built. TrindiKit provides an information state, update rules and interfaces to external modules. Another such framework is Dipper [4], which uses an agent paradigm to integrate subsystems.

This paper is structured as follows. In Section 2 we give an overview of the DIALOG project and the role a dialogue manager plays in this scenario. Section 3 outlines the architecture of ADMP. Section 4 presents the formalisation of the system, and Section 5 concludes the paper.

2 The DIALOG Project

The DIALOG project is researching the issues involved in automating the tutoring of mathematical proofs through the medium of flexible natural language. In order to achieve this a number of subproblems must be tackled. An *input analyser* [5] must perform linguistic analysis of utterances. These typically contain both natural language and mathematical expressions and exhibit much ambiguity. In addition to the linguistic analysis the input analyser delivers an underspecified representation of the proof content of the utterance. Domain reasoning is encapsulated in a *proof manager* [6], which replays and stores the status of the student's partial proof. Based on the partial proof, it must analyse the correctness, relevance and granularity of proof steps, and try to resolve ambiguous proof steps. Pedagogical aspects are handled by a *tutorial manager* [7], which decides when and how to give which hints.

These three modules, along with several others such as a natural language generator, collaborate in order to fully analyse student utterances and to compute system utterances. Their computation must be interleaved, since they work with shared information, and this interplay is orchestrated by the dialogue manager. Fig. 1 shows the modules involved in the DIALOG system.

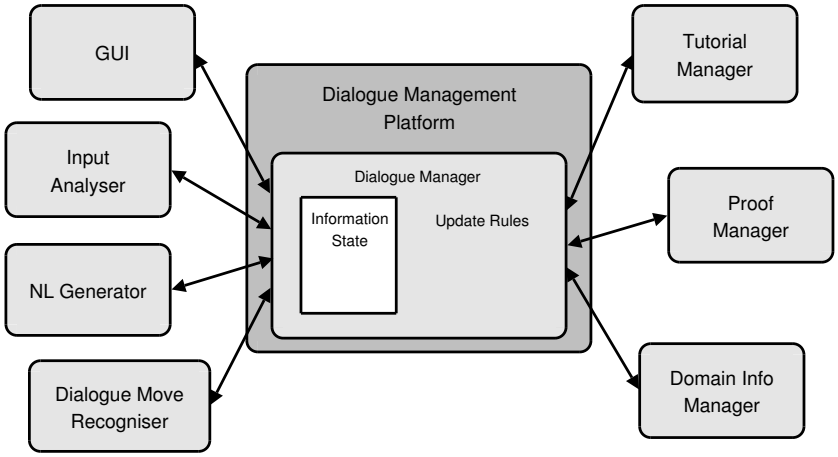


Fig. 1. The DIALOG system.

We illustrate how the system works with an example from the DIALOG corpus [8] in Fig. 2, where K stands for the complement operation and U is the universal set. The student has made a correct step (`Stud1`) and thus has begun building a partial proof of the theorem. Later in the proof he makes an irrelevant step (`Stud2`). We now consider how the modules of the system interact to generate the response in (`Sys2`).

The student's utterance first goes to the input analyser, which calculates its linguistic meaning and underspecified proof content. The proof content in this case is a representation of the content of the formula which is given in the utterance. This is then taken up by the proof manager. In the given proof context it assigns the proof step category `correct`, since the formula holds under the assumptions, but also `irrelevant`, because the step does not lead to a proof, or at least does not appear in a proof of the theorem. Simultaneously the *dialogue move recogniser* uses the linguistic meaning to determine the function of the utterance in the dialogue, and encodes this function in a dialogue move [9]. A dialogue move is a multi-dimensional abstraction of the functions of an utterance in a dialogue, such as *question* or *assertion*. In the example the utterance is a *domain contribution* , since it asserts a new piece of information in the theorem proving task.

A *domain information manager* determines what domain information was addressed by the proof step by accessing static mathematical knowledge re-

`Sys0:` Bitte zeigen Sie: Wenn $A \subseteq K(B)$, dann $B \subseteq K(A)$
Please show that: If $A \subseteq K(B)$ *then* $B \subseteq K(A)$

`Stud1:` $U/A = K(A)$
$U/A = K(A)$

`Sys1:` Das ist richtig! Und weiter?
That's correct! And now?

...

`Stud2:` also ist $A \not\subseteq B$
Therefore it holds that $A \not\subseteq B$

`Sys2:` Nein, das bringt nichts. Wissen Sie, was Sie mit der wenn-dann-Beziehung anfangen müssen?
No, that doesn't help. Do you know what to do with the if-then relation?

Fig. 2. Excerpt from the DIALOG corpus, session soc20k.

sources. The tutorial manager uses a combination of these results to add hinting information to the dialogue moves. In this case it decides to explicitly indicate the inappropriateness (*"No"*) and irrelevance (*"that doesn't help"*) of the step. Furthermore, a combination of a student and tutor model result in an explicit hint, namely to draw the student's attention to dissolving the if-then relation which is the head of the theorem.

In general, the result of each module's computation is a contribution of content to some system dialogue move. The final step is that a *natural language generator* generates the utterances constituting the system's response in (`Sys2`) from these dialogue moves. Since a module's computations depend only on information stored in a subset of the information state, their execution order is only partially constrained. This means that many computations can and should take place in parallel, as in the case of the proof manager and dialogue move recogniser in the example above.

DIALOG is an example of a complex system in which the interaction of many non-trivial components takes place. This interaction requires in turn non-trivial control to facilitate the distributed computation which results in the system response. This control function resides in the dialogue manager. As shown in Fig. 1, the dialogue manager forms the hub of the system and mediates all communication between the modules. It furthermore controls the interplay of the modules.

We realised a first DIALOG demonstrator in 2003. It includes a dialogue manager built on top of Rubin [10], a commercial platform for dialogue applications. This dialogue manager integrates each of the modules mentioned above and controls the dialogue. It provides an information state in which data shared between modules is stored, input rules which can update the information state based on input from modules, and interfaces to the system modules.

However, we identified some shortcomings of this first dialogue manager for the demonstrator, and these have formed part of the motivation for the development of ADMP:

- The modules in the system had no direct access to the information state, meaning they could not autonomously take action based on the state of the dialogue.
- The dialogue manager was static, and neither dialogue plans nor the interfaces to modules could be changed at runtime.
- There was also no way to reason about the flow of control in the system.

ADMP solves these problems by using a software agent approach to information state updates and by introducing a meta-level. The meta-level is used to reason about what updates should be made, and provides a place where the execution of the dialogue manager can be guided.

3 Architecture

The central concepts in the architecture of ADMP are information states and update rules, and these form the core of the system. An information state consists of slots which store values, and can be seen as an attribute-value matrix. It is a description of the state of the dialogue at a point in time, and can include information such as a history of utterances and dialogue move, the results of speech recognition or a representation of the beliefs of dialogue participants. Update rules encode transitions between information states, and are defined by a set of preconditions, a list of sideconditions, and a set of effects. Preconditions constrain what information states satisfy the rule, sideconditions allow arbitrary functions to be called within the rule, and effects describe the changes that should be made to the information state in order to carry out the transition that the rule encodes.

An update rule is embodied by an update rule agent, which carries out the computation of the transition that the update rule encodes. These check if the current information state satisfies the preconditions of the rule. When this is the case, they compute an *information state update* representing the fully instantiated transition. An information state update is a mapping from slotnames in the information state to the new values they have after the update is executed. We introduce information state updates as explicit objects in ADMP in order to be able to reason about their form and content at the meta-level.

As an example, we consider the information state in (1), a subset of the information state of the DIALOG system[3]. Here the user's utterance is already present in the slot `user_utterance`, but the linguistic meaning in the slot `lm` has not yet been computed. The slot `lu` stores a representation of the proof content of the utterance, and `eval_lu` stores its evaluated representation.

(1)
$$\text{IS} \begin{bmatrix} \texttt{user\_utterance} & \text{"also ist } A \not\subseteq B\text{"} \\ \texttt{lm} & \text{""} \\ \texttt{lu} & \text{""} \\ \texttt{eval\_lu} & \text{""} \end{bmatrix}$$

[3] In general an information state will contain richer data structures such as XML objects, but for presentation we restrict ourselves here to strings.

The update rule in (2) represents transitions from information states with a non-empty `user_utterance` slot to information states in which the `lm` and `lu` slots have been filled with the appropriate values.

$$(2)\quad \frac{\{\texttt{non\_empty(user\_utterance)}\}}{\{\texttt{lm} \rightarrow \texttt{p}\ ,\ \texttt{lu} \rightarrow \texttt{q}\}} \quad \begin{array}{l} < \texttt{r} := \texttt{input\_analyser(user\_utterance)}, \\ \quad \texttt{p} := \textit{extract\_lm}(\texttt{r}), \\ \quad \texttt{q} := \textit{extract\_lu}(\texttt{r}) > \end{array}$$

In ADMP's update rule syntax this rule is defined as:

```
(3) (ur~define-update-rule
     :name "Sentence Analyser"
     :preconds ((user_utterance :test #'ne-string))
     :sideconds ((r :function input_analyser
                    :slotargs (user_utterance))
                 (p :function extract-lm :varargs (r))
                 (q :function extract-lu :varargs (r))
                 )
     :effects ((lm p) (lu q))
    )
```

The precondition states that the slot `user_utterance` must contain a non-empty string. When this is the case, the rule can fire. It carries out its sideconditions, thereby calling the function `input_analyser`, which performs the actual computation and calls the module responsible for the linguistic analysis of utterances. Rule (2) thus represents the input analyser. The result of this computation is an object containing both the linguistic meaning of the utterance and an underspecified representation of the proof content. The functions *extract_lm* and *extract_lu* access the two parts and store them in the variables `p` and `q`, respectively. The information state update that the rule computes maps the slot name `lm` to the linguistic meaning of the utterance and the slot name `lu` to its proof content.

Rule (4) represents the proof manager, and picks up the proof content of the utterance in the slot `lu`.

$$(4)\quad \frac{\{\texttt{non\_empty(lu)}\}}{\{\texttt{eval\_lu} \rightarrow \texttt{r}\}} \quad < \texttt{r} := \texttt{pm\_analyse(lu)} >$$

The proof manager augments the information in `lu` by attempting to resolve underspecification and assign correctness and relevance categories, and the resulting update maps `eval_lu` to this evaluated proof step. A similar update rule forms the interface to the dialogue move recogniser, which uses the linguistic meaning of the utterance in `lm` to compute the dialogue move it represents. Since these two computations are both made possible by the result of the update from the input analyser, they can run in parallel.

Fig. 3 shows the architecture of ADMP. On the left is the information state. Update rules have in their preconditions constraints on some subset of the information state slots and are embodied by update rule agents, which are shown here next to the information state. When an update rule agent sees that the preconditions of its rule hold, the rule is applicable and can fire. The agent then executes each of the sideconditions of the rule, and subsequently computes the

information state update that is expressed by the rule's effects. The resulting information state update is written to the update blackboard, shown in the middle of the diagram.

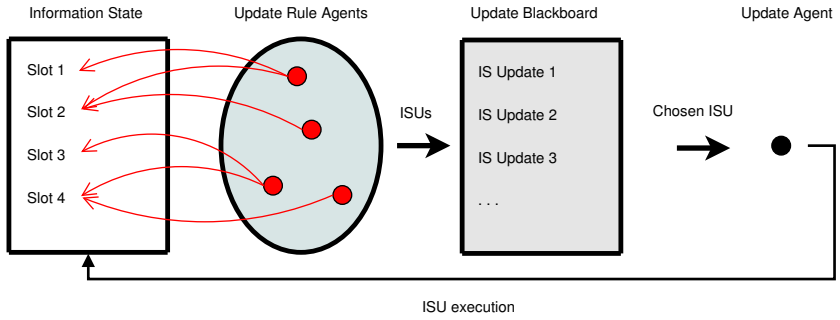


Fig. 3. The architecture of ADMP.

The update blackboard collects the proposed updates from the update rule agents. These agents act in a concurrent fashion, so that many of them may be simultaneously computing results; some may return results quickly and some may perform expensive computations, e.g. those calling external modules. Thus the set of entries on the update blackboard can grow continually. On the far right of the diagram is the update agent, which surveys the update blackboard. After a timeout or some stimulus it chooses the heuristically preferred update (or a combination of updates) and executes it on the current information state. This completes a transition from one information state to the next.

Finally the update agent resets the update rule agents. Agents for whom the content of the slots in their preconditions has not changed can continue to execute since they will then be computing under essentially the same conditions (i.e. the information that is relevant to them is the same). Agents for whom the slots in the preconditions have changed must be interrupted, even if their preconditions still happen to hold. This is because they are no longer computing within the correct current information state.

4 A Formal Specification of ADMP

We now give a concise and mathematically rigorous specification of ADMP. We introduce the concepts and terminology necessary to guarantee the well-definedness of information states and update rules, and we give an algorithmic description of the update rule agents and the update agent.

Information States and Information State Updates First, we fix some data structures for the slot names and the slot values of an information state.

In our scenario it is sufficient to work with strings in both cases (alternatively we could work with more complex data structures). Let $\mathcal{A}$ and $\mathcal{B}$ be alphabets. We define the *language for slot names* as $\mathit{SlotId} := \mathcal{A}^*$ and the *language for slot values* as $\mathit{SlotVal} := \mathcal{B}^*$. In our framework we want to support the checking of certain properties for the values of single slots. Thus we introduce the notion of a Boolean test function for slot values. A *Boolean test function* is a function $f \in \mathcal{BT} := \mathit{SlotVal} \to \{\top, \bot\}$.

Next, we define information state slots as triples consisting of a slot name, a slot value, and an associated Boolean test function. The set of all possible *information state slots* is $\mathit{Slots} := \mathit{SlotId} \times \mathcal{BT} \times \mathit{SlotVal}$. Given an information state slot $u = (s, b, v)$, the slot name, the test function, and the slot value can be accessed by the following projection functions: $\mathit{slotname}(u) := s$, $\mathit{slotfunc}(u) := b$ and $\mathit{slotval}(u) := v$.

Information states are sets of information state slots which fulfil some additional conditions. Given $r \subseteq \mathit{Slots}$, we call r a *valid information state* if $r \neq \emptyset$ and for all $u_1, u_2 \in r$ we have $\mathit{slotname}(u_1) = \mathit{slotname}(u_2) \Rightarrow u_1 = u_2$. We define $\mathcal{IS} \subset \mathcal{P}(\mathit{Slots})$ to be the set of all valid information states. The set of all slot names of a given information state $r \in \mathcal{IS}$ can be accessed by a function $\mathit{slotnames} : \mathcal{IS} \to \mathcal{P}(\mathit{SlotId})$ which is defined as follows

$$\mathit{slotnames}(r) = \{s \in \mathit{SlotId} \mid \exists\, u \in r \,.\, \mathit{slotname}(u) = s\}$$

We define a function $\mathit{read} : \mathcal{IS} \times \mathit{SlotId} \to \mathit{SlotVal}$ to access the value of a slot in an information state where $\mathit{read}(r, s) = \mathit{slotval}(u)$ for the unique $u \in r$ with $\mathit{slotname}(u) = s$.

In our framework information states are dynamically updated, i.e. the values of information state slots are replaced by new values. Such an *information state update* is a mapping from slots to their new values. The set of all valid information state updates μ is denoted by $\mathcal{ISU}$, the largest subset of $\mathcal{P}(\mathit{SlotId} \times \mathit{SlotVal})$ for which the following restriction holds: $\forall (s_1, v_1), (s_2, v_2) \in \mu \,.\, s_1 = s_2 \Rightarrow v_1 = v_2$ for all $\mu \in \mathcal{ISU}$. We define $\mathcal{ISU}_\bot := \mathcal{ISU} \cup \{\bot\}$. An information state update $\mu \in \mathcal{ISU}$ is *executable* in an information state $r \in \mathcal{IS}$ if the slot names addressed in μ actually occur in r and if the new slot values suggested in μ fulfil the respective Boolean test functions, i.e.

$$\mathit{executable}(r, \mu) \text{ iff } \forall (s, v) \in \mu \,.\, \exists\, u \in r \,.\, \mathit{slotname}(u) = s \wedge \mathit{slotfunc}(u)(v) = \top$$

We overload the function *slotnames* from above and analogously define it for information state updates. Information state updates are executed by a function $\mathit{execute\_update} : \mathcal{IS} \times \mathcal{ISU} \to \mathcal{IS}$. Given an information state $r \in \mathcal{IS}$ and an information state update $\mu \in \mathcal{ISU}$ we define

$$\mathit{execute\_update}(r, \mu) = \begin{cases} r & \text{if not } \mathit{executable}(r, \mu) \\ r^- \cup r^+ & \text{otherwise} \end{cases}$$

where

$$r^- := (r \setminus \{(s, b, v) \in r | s \in \mathit{slotnames}(\mu)\}$$
$$r^+ := \{(s', b', v') \mid (s', v') \in \mu \wedge \exists u \in r \,.\, s' = \mathit{slotname}(u) \wedge b' = \mathit{slotfunc}(u)\}$$

Update Rules Update rules use the information provided in an information state to compute potential information state updates. They consist of preconditions, sideconditions and effects.

The preconditions of an update rule identify the information state slots that the rule accesses information from. For each identified slot an additional test function is provided which specifies an applicability criterion. Intermediate computations based on information in the preconditions are performed by the sideconditions of the update rules. For this, a sidecondition may call complex external modules, such as the linguistic analyser or the domain reasoner. The results of these side-computations are bound to variables in order for them to be accessible to subsequent sideconditions and to pass them over from the sideconditions to the effects of a rule. We now give a formal definition of each part in turn.

Let $s \in \mathit{SlotId}$ and $b \in \mathcal{BT}$. The tuple (s, b) is called an *update rule precondition*. The set of all update rule preconditions is denoted by $\mathcal{C} := \mathit{SlotId} \times \mathcal{BT}$. We define projection functions $\mathit{pc\_slotname} : \mathcal{C} \rightarrow \mathit{SlotId}$ and $\mathit{pc\_testfunc} : \mathcal{C} \rightarrow \mathcal{BT}$ such that $\mathit{pc\_slotname}(pc) = s$ and $\mathit{pc\_testfunc}(pc) = b$ for all $pc = (s, b)$. An information state $r \in \mathcal{IS}$ *satisfies* an update rule precondition $pc = (s, b)$ if the function b applied to the value of the slot in r named s returns $\top$, i.e. $\mathit{satisfies}(r, pc)$ iff $\exists u \in r \,.\, \mathit{pc\_testfunc}(pc)(\mathit{slotval}(u)) = \top \wedge \mathit{slotname}(u) = \mathit{pc\_slotname}(pc)$. We overload the predicate *satisfies* and define it for sets of preconditions $\mathcal{C}' \subseteq \mathcal{C}$ and information states $r \in \mathcal{IS}$ as follows: $\mathit{satisfies}(r, \mathcal{C}')$ holds if each precondition in $\mathcal{C}'$ is satisfied by r.

Let $v \in \mathit{Var}$ be a variable where Var is a set of variables distinct from the languages $\mathcal{A}^*$ and $\mathcal{B}^*$, let $(v_1 \ldots v_m) \in \mathit{Var}^m$ be an m-tuple of variables, let $(s_1 \ldots s_n) \in \mathit{SlotId}^n$ be an n-tuple of slot names, and let $f : \mathit{SlotVal}^n \rightarrow \mathit{SlotVal}^m \rightarrow \mathit{SlotVal}$ be a function[4] $(n, m \geq 0)$. A *single sidecondition* is thus given by the quadruple $(v, (s_1, \ldots, s_n), (v_1, \ldots, v_m), f)$. The set of all single sideconditions is denoted by $\mathcal{D} := \mathit{Var} \times \mathit{SlotId}^n \times \mathit{Var}^m \times (\mathit{SlotVal}^n \rightarrow \mathit{SlotVal}^m \rightarrow \mathit{SlotVal})$.

Given the set $\mathcal{D}$ of single sideconditions sc_i, the sideconditions of an update rule are now modelled as lists $l := < sc_1, \ldots, sc_n >$, $n \geq 0$. We further provide projection functions $\mathit{sc\_var} : \mathcal{D} \rightarrow \mathit{Var}$, $\mathit{sc\_slottuple} : \mathcal{D} \rightarrow \mathit{SlotId}^n$, $\mathit{sc\_slotnames} : \mathcal{D} \rightarrow \mathcal{P}(\mathit{SlotId})$, $\mathit{sc\_vartuple} : \mathcal{D} \rightarrow \mathit{Var}^m$, $\mathit{sc\_varnames} : \mathcal{D} \rightarrow \mathcal{P}(\mathit{Var})$ and $\mathit{sc\_func} : \mathcal{D} \rightarrow (\mathit{SlotVal}^n \rightarrow \mathit{SlotVal}^m \rightarrow \mathit{SlotVal})$, such that for all $sc = (v, (s_1, \ldots, s_n), (v_1, \ldots, v_m), f) \in \mathcal{D}$ it holds that $\mathit{sc\_var}(sc) = v$, $\mathit{sc\_slottuple}(sc) = (s_1, \ldots, s_n)$, $\mathit{sc\_slotnames}(sc) = \{s_1, \ldots, s_n\}$, $\mathit{sc\_vartuple}(sc) = (v_1, \ldots, v_m)$, $\mathit{sc\_varnames}(sc) = \{v_1, \ldots, v_m\}$ and $\mathit{sc\_func}(sc) = f$.

A *sidecondition list* l is called valid if two conditions hold: for all $sc_i, sc_j \in l$ with $i \neq j$ we must have $\mathit{sc\_var}(sc_i) \neq \mathit{sc\_var}(sc_j)$ and for all $sc_i \in l$ we must have $\mathit{sc\_varnames}(sc_i) \subseteq \{v | \exists\ sc_j \in l \,.\, j < i \wedge v = \mathit{sc\_var}(sc_j)\}$. The set of all valid sidecondition lists is denoted as $\mathcal{D}_l$.

Let $s \in \mathit{SlotId}$ and $v \in \mathit{Var}$ be a variable. The tuple (s, v) is called an *update rule effect*. The set of all update rule effects is denoted by $\mathcal{E} := \mathit{SlotId} \times \mathit{Var}$.

[4] We assume the right-associativity of $\rightarrow$.

We provide projection functions $e\_slotname : \mathcal{E} \to SlotId$ and $e\_var : \mathcal{E} \to Var$ such that $e\_slotname((s, v)) = s$ and $e\_var((s, v)) = v$.

Let $\mathcal{U}$ be a set of rule names (distinct from $\mathcal{A}^*$, $\mathcal{B}^*$, and Var). An *update rule* is a quadruple $\nu \in \mathcal{UR} := \mathcal{U} \times \mathcal{P}(\mathcal{C}) \times \mathcal{D}_l \times \mathcal{P}(\mathcal{E})$. An update rule $\nu = (n, c, d, e) \in \mathcal{UR}$ is *well-defined* w.r.t. the information state r if

1. the slotnames mentioned in the preconditions actually occur in r, i.e, for all $pc \in c$ we have $pc\_slotname(pc) \in slotnames(r)$,
2. each slot that is accessed by a sidecondition function has been mentioned in the preconditions, i.e., $(\bigcup_{d_i \in d} sc\_slotnames(d_i)) \subseteq \{s \in SlotId \mid \exists\ pc \in c\ .\ pc\_slotnames(pc) = s\}$,
3. the variables occurring in the effects have been initialised in the sideconditions, i.e., $\{v \in Var \mid \exists\ e_i \in e\ .\ e\_var(e_i) = v\} \subseteq \{v \in Var \mid \exists\ sc \in d\ .\ sc\_var(sc) = v\}$, and
4. the slotnames in the effects refer to existing slots in the information state r, i.e., $\{s \in SlotId \mid \exists e_i \in e\ .\ e\_slotname(e_i) = s\} \subseteq slotnames(r)$.

Let $\nu = (n, c, d, e) \in \mathcal{UR}$ be an update rule and $r \in \mathcal{IS}$ be an information state. ν is called *applicable* in r if ν is well-defined w.r.t. r and $satisfies(r, c)$ holds. This is denoted by $applicable(r, \nu)$.

Update Rule Agents Update rule (software) agents encapsulate the update rules, and their task is to compute potential information state updates. The suggested updates are not immediately executed but rather they are passed to an update blackboard for heuristic selection. Update rule agents may perform their computations in a distributed fashion.

An update rule agent embodies a function $execute\_ur\_agent : \mathcal{UR} \to (\mathcal{IS} \to \mathcal{ISU}_\perp)$. The function $execute\_ur\_agent(\nu)$ takes an update rule ν and returns a function (lambda term) representing the computation that that rule defines. The new function can then be applied to a given information state in order to compute a suggestion for how to update this information state. For each update rule we obtain a different software agent.

We introduce a macro `sc_evaluate` which abbreviates the retrieval of the values in the variables and slotnames in the body of sidecondition and the computation of the value which is to be stored in the sidecondition's variable. We use **function_call** to apply a function to the arguments which follow it and **value_of** to retrieve the value stored in a variable.

$$
\begin{array}{l}
\texttt{sc\_evaluate}(sc)\ = \\
\quad \textbf{let}\ (s_1, \ldots, s_n) := sc\_slottuple(sc) \\
\quad\quad \textbf{let}\ (v_1, \ldots, v_m) := sc\_vartuple(sc) \\
\quad\quad\quad \textbf{let}\ (t_1, \ldots, t_m) := (\textbf{value\_of}(v_1), \ldots, \textbf{value\_of}(v_m)) \\
\quad\quad\quad\quad \textbf{function\_call}(sc\_func(sc), (read(r, s_1), \ldots, read(r, s_n)), (t_1, \ldots, t_m))
\end{array}
$$

We now define $execute\_ur\_agent$ as

$$
\begin{array}{l}
execute\_ur\_agent(\nu = (n, c, d, e)) = \\
\quad \lambda r \;.\; \textbf{if}\ applicable(r, \nu) \\
\qquad \textbf{then} \\
\qquad\quad \textbf{let}\ < sc_1, \ldots, sc_n > := d \\
\qquad\qquad \textbf{let}\ sc\_var(sc_1) := \texttt{sc\_evaluate}(sc_1) \\
\qquad\qquad\quad \textbf{let}\ sc\_var(sc_2) := \texttt{sc\_evaluate}(sc_2) \\
\qquad\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad \textbf{let}\ sc\_var(sc_n) := \texttt{sc\_evaluate}(sc_n) \\
\qquad\qquad\qquad\quad \{(s, v) | \exists (s, sc\_var(sc_i)) \in e \;.\; v = \textbf{value\_of}(sc\_var(sc_i))\} \\
\qquad \textbf{else}\ \bot
\end{array}
$$

Update Blackboard and Update Agent An *update blackboard* is modelled as a set of information state updates $w \in \mathcal{UB} := \mathcal{P}(\mathcal{ISU})$, and stores proposed updates to the current information state. The *update agent* investigates the entries on the update blackboard, heuristically chooses one of the proposed information state updates and executes it. We assume a user-definable function $choose : \mathcal{UB} \rightarrow \mathcal{ISU}$ which realises the heuristic choice based on some heuristic ordering criterion $>_{\mathcal{UB}} : \mathcal{ISU} \times \mathcal{ISU}$. A simple example of a partial ordering criterion $>_{\mathcal{UB}}$ is

$$\mu_1 >_{\mathcal{UB}} \mu_2 \ \text{ iff } \ slotnames(\mu_2) \subseteq slotnames(\mu_1)$$

In fact, *choose* may be composed of several such criteria, and clearly the overall behaviour of the system is crucially influenced by them. The update agent now embodies a function $update\_agent : \mathcal{UB} \times (\mathcal{UB} \rightarrow \mathcal{ISU}) \times \mathcal{IS} \rightarrow \mathcal{IS}$ which is defined as

$$update\_agent(w, choose, r) = execute\_update(r, choose(w))$$

5 Conclusion

In this paper we have presented a formalisation of ADMP, a platform for developing dialogue managers using the information state update approach. We were motivated by the need to integrate many complex and heterogeneous modules in a flexible way in a dialogue system for mathematical tutoring. These modules must be able to communicate and share information with one another as well as to perform computations in parallel.

ADMP supports these features by using a hierarchical agent-based design. The reactive nature of the update rule agents allows for the autonomous concurrent execution of modules triggered by information in the information state. This furthermore obviates the need for a strict pipeline-type control algorithm often seen in dialogue systems, since agents can execute without being explicitly called. Interfacing the dialogue manager with system modules is also simplified by using

the agent paradigm, because adding a new module involves only declaring a new update rule. Finally, the meta-level provides a place where overall control can take place if needed.

ADMP thus allows the formalisation of theories of dialogue in the information state update approach, offering the functionality of related systems like TrindiKit and Dipper. However by introducing an explicit heuristic layer for overall control it allows reasoning about the execution of the dialogue manager which these two systems do not support.

An instantiation of ADMP is achieved by declaring an information state, a set of update rules which operate on the information state, and a *choose* function, whereby a developer can fall back to a default function such as suggested in the previous section. A user-defined *choose* function should compute valid $\mathcal{ISU}$s, also in the case where $\mathcal{ISU}$s from the update blackboard are merged. As an example, a conservative merge strategy would simply reject the merging of pairs of $\mathcal{ISU}$s whose slotname sets intersect. Update rule agents and the update agent are automatically generated from the update rule declarations.

We have recently implemented ADMP and given an instantiation for the DIALOG system which uses eleven update rules and requires no declaration of control structure. We have also shown that we can implement a propositional resolution prover in ADMP with four agents and five information state slots, which corresponds to just 40 lines of code. Extensions such as a set of support strategy can be realised simply by adding agents, possibly at runtime.

We foresee as future work the extension of our agent concept to include for instance resource sensitivity, and the investigation of further default heuristics for the dialogue scenario. Other interesting work is to turn the specification given in this paper into a formalisation within a higher-order proof assistant such as ISABELLE/HOL, HOL or OMEGA and to verify its properties.

References

1. Traum, D., Larsson, S.: The information state approach to dialogue management. In van Kuppevelt, J., Smith, R., eds.: Current and new directions in discourse and dialogue. Kluwer (2003)
2. Buckley, M., Benzmüller, C.: A Dialogue Manager supporting Natural Language Tutorial Dialogue on Proofs. Electronic Notes in Theoretical Computer Science (2006) To appear.
3. Benzmüller, C., Sorge, V.: Ω-Ants – An open approach at combining Interactive and Automated Theorem Proving. In Kerber, M., Kohlhase, M., eds.: 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000), AK Peters (2000)
4. Bos, J., Klein, E., Lemon, O., Oka, T.: Dipper: Description and formalisation of an information-state update dialogue system architecture. In: Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue, Sapporo, Japan (2003)
5. Horacek, H., Wolska, M.: Interpreting Semi-Formal Utterances in Dialogs about Mathematical Proofs. Data and Knowledge Engineering Journal **58**(1) (2006) 90–106

6. Benzmüller, C., Vo, Q.: Mathematical domain reasoning tasks in natural language tutorial dialog on proofs. In Veloso, M., Kambhampati, S., eds.: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, Pennsylvania, USA, AAAI Press / The MIT Press (2005) 516–522
7. Tsovaltzi, D., Fiedler, A., Horacek, H.: A Multi-dimensional Taxonomy for Automating Hinting. In Lester, J.C., Vicari, R.M., Paraguaçu, F., eds.: Intelligent Tutoring Systems, 7th International Conference (ITS 2004). Number 3220 in LNCS, Springer (2004) 772–781
8. Benzmüller, C., Fiedler, A., Gabsdil, M., Horacek, H., Kruijff-Korbayová, I., Pinkal, M., Siekmann, J., Tsovaltzi, D., Vo, B.Q., Wolska, M.: A Wizard-of-Oz experiment for tutorial dialogues in mathematics. In: Proceedings of the AIED Workshop on Advanced Technologies for Mathematics Education, Sidney, Australia (2003) 471–481
9. Allen, J., Core, M.: Draft of DAMSL: Dialogue act markup in several layers. DRI: Discourse Research Initiative, University of Pennsylvania (1997)
10. Fliedner, G., Bobbert, D.: A framework for information-state based dialogue (demo abstract). In: Proceedings of the 7th workshop on the semantics and pragmatics of dialogue (DiaBruck), Saarbrücken (2003)

DiaWOz-II - A Tool for Wizard-of-Oz Experiments in Mathematics*

Christoph Benzmüller[1], Helmut Horacek[1], Ivana Kruijff-Korbayová[2], Henri Lesourd[1], Marvin Schiller[1], and Magdalena Wolska[2]

[1] Dept. of Computer Science
[2] Dept. of Computational Linguistics
Saarland University, Germany
{chris, horacek, henri, schiller}@ags.uni-sb.de,
{korbay, magda}@coli.uni-sb.de

Abstract. We present DiaWOz-II, a configurable software environment for Wizard-of-Oz studies in mathematics and engineering. Its interface is based on a structural *wysiwyg* editor which allows the input of complex mathematical formulae. This allows the collection of dialog corpora consisting of natural language interleaved with non-trivial mathematical expressions, which is not offered by other Wizard-of-Oz tools in the field. We illustrate the application of DiaWOz-II in an empirical study on tutorial dialogs about mathematical proofs, summarize our experience with DiaWOz-II and briefly present some preliminary observations on the collected dialogs.

Key words: Dialog systems, natural language dialog in mathematics, tutoring systems, Wizard-of-Oz experiments

1 Introduction

For the development of natural language dialog systems, experiments in the Wizard-of-Oz (WOZ) paradigm are a valued source of dialog corpora.[3]

Existing environments for WOZ experiments, even those for the domain of mathematics tutoring, generally operate in domains that either require only simple mathematical formulae (with operators like + and ×), or they separate the mathematical objects (geometric figures or equations) from the tutorial dialog (such as in the Wooz tutor [2], for example). In this paper we present our WOZ

* This work has been funded by the DFG Collaborative Research Center on Resource-Adaptive Cognitive Processes, SFB 378 (http://www.coli.uni-saarland.de/projects/sfb378/).

[3] A Wizard-of-Oz experiment [1] serves to test the usability of a hypothetical software system. The system is (partially) simulated by a human expert, the *wizard*. Typically, a mediator software partially implements the functionality of the simulated system.

environment DiaWOz-II which, in contrast to that, enables the collection of dialogs where natural language text is interleaved with mathematical notation, as is typical for (informal) mathematical proofs. The interface components of DiaWOz-II are based on the *what-you-see-is-what-you-get* scientific text editor TEXMACS[4] [3]. DiaWOz-II provides one interaction window for the user and one for the wizard, together with additional windows displaying instructions and domain material for the user, and additional notes and pre-formulated text fragments for the wizard. All of these windows allow for copying freely from one to the other. Furthermore, our DiaWOz-II allows the wizard to annotate user dialog turns with their categorization. DiaWOz-II is also connected to a spell-checker for checking both the user's and the wizard's utterances.

This paper is organized as follows: In Sect. 2 we motivate the design of our system. In Sect. 3.1 we describe the TEXMACS *wysiwyg* editor, on which the interface of DiaWOz-II is based. The DiaWOz-II system is discussed in detail in Sect. 3. In Sect. 4 we discuss the application of DiaWOz-II in a recently completed series of experiments. Section 5 concludes the paper.

2 Design Aspects

General Requirements for WOZ Tools. We list some general requirements we considered in the development of DiaWOz-II:

Plausibility and Comfort. For WOZ experiments, it is crucial to maintain the user's belief that he is interacting with a fully artificial system. Therefore, the software that mediates between wizard and student should enable the wizard to conceal his human identity. This is not a trivial pursuit, since it is common sense that "people are flexible, computers are rigid (or consistent), people are slow at typewriting, computer output is fast" [4]. Thus, the WOZ tool is required to enable the wizard to respond to the participant quickly and comfortably and in a plausible way.

Suitability for Book-keeping. The main goal of WOZ experiments is the analysis of the interactions between the subjects and the simulated system. Therefore, the WOZ tool is required to record the dialogs using a representation format suitable for further processing and analysis.

Flexibility and Simplicity. The WOZ tool should be easily adjustable, so that it can be used under different experimental conditions and in different domains. Adjustments to the software should not significantly add to the complexity of carrying out a series of experiments, a process which by itself poses enough challenges.

Tool Integration. The WOZ tool should support the integration of other software components, for example, modules that already realize single parts of the simulated overall system.

[4] `www.texmacs.org`

Specific Requirements for DiaWOz-II. DiaWOz-II has been developed for application in the DIALOG project [5], which investigates the use of natural language dialog for teaching mathematical proofs. The particular research foci of the DIALOG project are natural language analysis, domain reasoning for mathematics, and tutorial aspects of mathematics tutoring.

In 2003, we carried out a first empirical study [6] in the WOZ paradigm in which we collected a corpus of tutorial dialogs on mathematical proofs in German. The study concentrated on the comparison between three tutoring strategies, namely the *Socratic*, *didactic* and the *minimal feedback* strategies. For this purpose, we developed the DiaWoZ [7] environment, the predecessor of DiaWOz-II. DiaWoZ supports complex dialog specifications, which were needed in order to specify a particular hinting algorithm used in the *Socratic* tutoring condition. DiaWoZ allows keyboard-to-keyboard interaction between the wizard and the student. The interfaces consist mainly of a text window with the dialog history and a menu bar providing mathematical symbols. Furthermore, the wizard can assign dialog state transitions and speech act categories to student turns w.r.t. the underlying dialog model. The DiaWoZ interface allowed free mixing of natural language text with mathematical symbols. Still, there was room for improvement with respect to the *plausibility and comfort* criterion postulated above. For example, the experiment participants suggested the use of the keyboard instead of the mouse for inserting mathematical symbols.

The first study motivated a second series of experiments [8], which we briefly describe in Sect. 4. In contrast to the first study, the more recent study imposes less constraints on the wizards' tutoring and assumes a rather simple dialog model. However, in comparison to the first study, the second study is more focused on linguistic phenomena and mathematical domain reasoning in tutorial dialogs and the interplay between these two.

Related Work. A variety of WOZ tools and dialog system toolkits already exist. Examples are the simulation environment ARNE [4], the SUEDE prototyping tool for speech user interfaces [9] and MD-WOZ [10].

In the domain of mathematics, a WOZ simulation of the ALPS environment [11] and the Wooz tutor [2] should be mentioned. In the case of ALPS, the Synthetic Interview (SI) method is used, i.e. the student formulates free-form questions in a chat window, and receives a video clip with an answer. In the ALPS system, these video clips are pre-recorded, stored in a database, and retrieved as answers for the questions from the user, whereas in the WOZ simulation of ALPS, the wizard's responses are spontaneous. The ALPS tutor is designed to be an algebra tutor. Typical problems in the domain of ALPS are for example the computation of area and perimeter of geometric figures.

The Wooz tutor is also a tool for keyboard-to-keyboard interaction in the domain of algebra. It offers a chat window displaying the tutorial dialog, a dedicated window displaying the problem statement and a dedicated editor for editing equations. A typical problem given to the participants is "please factor $11x^2 - 11x + 6$".

The interfaces of these two systems are not intended for mixing natural language input with the mathematical notation employed for proving theorems, which we investigate in the DIALOG project. For our dialog system we aim for an interface that allows flexible and easy input for mathematical formulae and natural language text. This requirement is addressed by the interface in DiaWOz-II.

3 The DiaWOz-II System

We decided to build a new WOZ tool rather than trying to improve the existing DiaWoZ system. An important motivation was to use TEXMACS [3] as a platform for the new system in order to benefit from its typesetting abilities, its configurable GUI and its event-handling as a building block for the creation of a more lightweight software.

DiaWOz-II is realized as a classical client-server architecture, and consists of a server and two client interfaces for the student and the tutor respectively. The architecture allows keyboard-to-keyboard interaction between the student and the tutor. Furthermore, the server fulfills other central functions, namely the recording of the interaction in a log file, controlling turn-taking between the dialog participants, and providing an interface to a spell-checker. We first describe TEXMACS and its role in DiaWOz-II before we further elaborate on each of these aspects in turn.

3.1 TEXmacs

TEXMACS is a scientific text editor with strong support for mathematical typesetting which is inspired by TEX and *GNU emacs*. The internal representation of a TEXMACS-document is well organized in a tree-like structure. TEXMACS provides two alternative editing modes: (i) a *wysiwyg interface* that allows to directly manipulate the typeset document and (ii) a *source mode* that provides a view of the internal document representation in the underlying, structured TEXMACS markup language. This language supports the definition of *macros*, which are generally easy to read and understand. It is also worth noting that the standard TEXMACS markup language inherits many usual LATEX constructs, in such a way that for LATEX-literate persons, starting to use TEXMACS is usually straightforward. Thus, extending the markup (namely, defining new kinds of tags together with how these newly defined tags must be typeset) can be done in a very convenient way using macros. For more sophisticated behavior, for example, the implementation of an interactive application, one can use *Scheme*, the standard TEXMACS scripting language.

TEXMACS fulfills the *plausibility and comfort* requirement introduced in Sect. 2 by offering various advanced modes of input for mathematical symbols, and in particular it enables LATEX commands. Using TEXMACS also fulfills the *flexibility and simplicity* requirement, since it can be reconfigured with little effort.

The TEXMACS editor has already been adapted as an interface to a diversity of external tools, most of which are computer algebra systems. However, using TEXMACS as an interface for a (simulated) tutoring system is novel.

3.2 TEXmacs as Base Component of DiaWOz-II

A TEXMACS application as employed in DiaWOz-II has the overall structure shown in Fig. 1. Such an application consists of (i) a set of TEXMACS *macros* which implement the *visualization* of the different parts of the user interface (i.e. what are their shapes, their locations, the text attributes (e.g. color, font, ...), etc.), and (2) a set of *Scheme scripts*, which implement the mechanism which interprets the *events* (e.g., a mouse click, a key press, etc.) and *modifies* the interface accordingly.

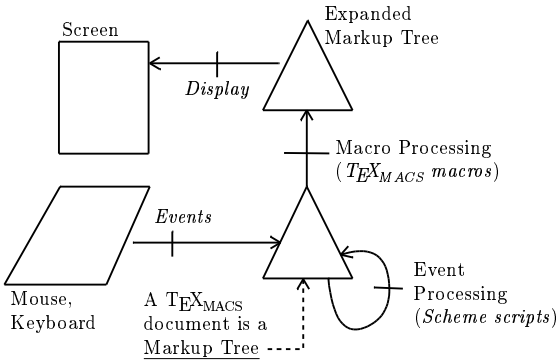


Fig. 1. Structure of a TEXMACS application

Macros. A very basic example of a TEXMACS macro that can be used to turn a part of the document into *italics underlined* text is (cf. [12] for more details on the macro language):

```
<underlined-italics|x> => <with|font-shape|italic|<underline|<arg|x>>>
```

The left-hand side of this expression defines the use of the macro (i.e., the *non-expanded* markup, as it can be found in a TEXMACS document file) and the right-hand side its expansion. Given this macro definition, the TEXMACS markup fragment `<underlined-italics|This is italics underlined text.>` is first rewritten by the macro processor as `<with|font-shape|italic|<underline|This is italics underlined text.>>` and then displayed in TEXMACS as *This is italics underlined text.*

Processing the Markup Using *Scheme*. The event processor can be extended by plugins written as *Scheme* scripts. These scripts can manipulate the internal markup tree that represents the user interface, typically as a reaction to an event (e.g., mouse, keyboard, network, etc.). As a reaction to the changes in the markup, the macros are reevaluated, and the display is then updated.

3.3 Student and Wizard Interfaces

The dialog system simulated by DiaWOz-II is presented to the student as a window, referred to as the interaction window. It consists of menu bars and a text field, as shown in Fig. 2. The dialog history and the prompt for the current input are displayed in the same text field, separated by a horizontal bar at the bottom in Fig. 2. The utterances from the tutor and the student are displayed in

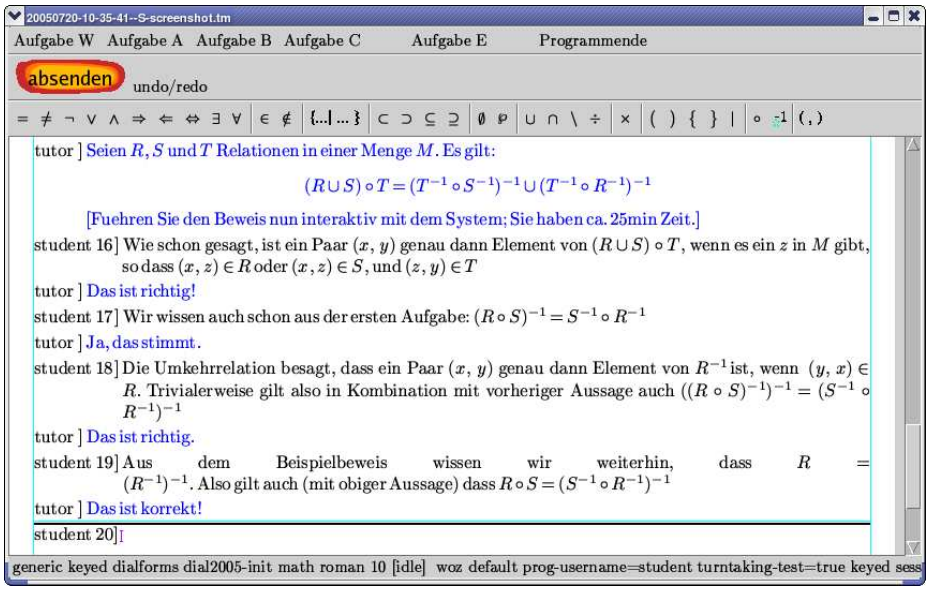


Fig. 2. Interaction window of the student interface

different colors for better readability. The student can send messages by pressing the "absenden" (submit) button. Upon submitting, the message becomes part of the dialog history. The answers by the tutor are accompanied by an acoustic signal.

In a second window, which is independent of the interaction window, supplementary study material with mathematical concepts and definitions is displayed.

The wizard's interface, as shown in Fig. 3, is conceptually similar to the student's interface. In addition, the wizard is asked to categorize each student turn w.r.t. three dimensions: correctness, granularity and relevance; the wizard fills out the fields of a small table referring to the three dimensions by making choices in pull-down menus, or directly by typing. The wizard's button for sending messages is only enabled once all the fields have been filled. If the student's utterance does not represent a mathematical statement the wizard fills in default values (`N/A`).

We now turn in more detail to the methods for inserting mathematical symbols in DiaWOz-II, which are made available by TEXMACS. Mathematical symbols (e.g., $\emptyset$) can be created by using LATEX commands (e.g., `\emptyset`) or by using additional commands defined when designing the interface (e.g., the command `\emptyset` in German language, i.e. `\leeremenge`). These commands are also made available in the menu bar. DiaWOz-II also allows for structured commands, e.g. commands that create pairs of brackets for pair (□, □) and for set notation {□ | □}. An example is the macro *paar* (German for *pair)*:

```
<paar|left|right> => ( <arg|left> , <arg|right> )
```

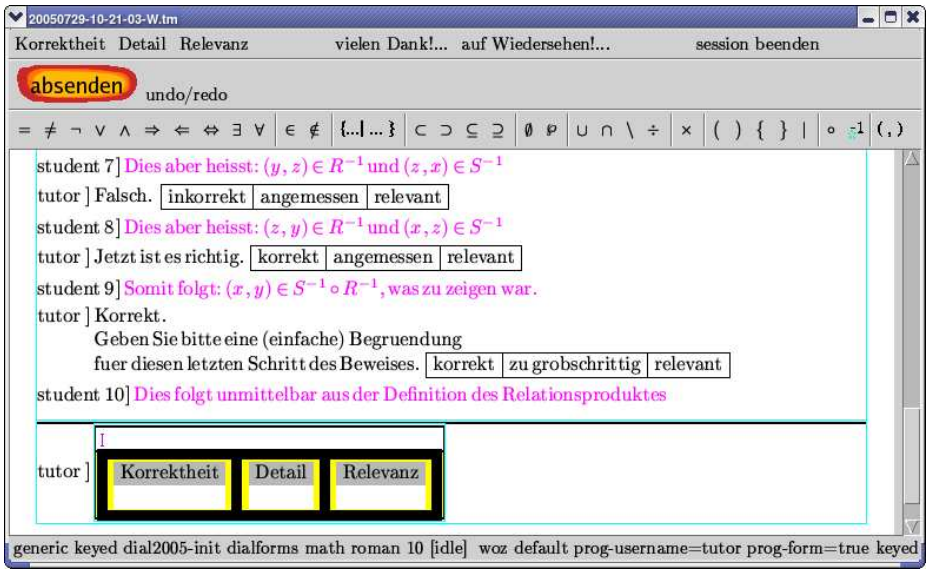


Fig. 3. The interaction window of the wizard interface

Invoking `\paar` with the arguments x and y yields the formula (x, y). The two arguments need not necessarily be provided when invoking the macro, their respective placeholders can be also filled in interactively and modified later. Macros can be nested, and most importantly, they avoid missing parentheses when the user writes expressions using the pair notation. The set of macros provided with DiaWOz-II can be easily extended with further TEXMACS macros.

TEXMACS furthermore makes it possible to distinguish between mathematical symbols created via the menu bar and via LATEX commands, even if they appear to be the same at the typesetting level.

Using structured building blocks for constructing mathematical formulae via macros is similar to the MATHS TILES approach [13]. MATHS TILES are graphical tiles that can contain text, diagrammatic shapes and *sockets*, which are placeholders where other MATHS TILES can be inserted to form composite objects. TEXMACS has the advantage over MATHS TILES that it already includes by default a large set of macros for constructing formulae, such as a large number of macros that represent LATEX commands.

3.4 The Server

The central capabilities of DiaWOz-II reside in the server. Its main task is to pass the dialog contributions back and forth between the student and the wizard interface. Furthermore, it provides the following other central services:

Log-file Mechanism. All dialogs are recorded in a log-file in DiaWOz-II. The log-file format is based on the representation format of TEXMACS, which is a

structured, extensible and open document format. Naturally, the annotations performed by the wizard for each student turn are also stored in the log-file.

Spell-Checking. Spelling mistakes by the wizard can be a giveaway of human simulation. Therefore, our server (optionally) integrates a spell-checker. If spell-checking is activated, a message from the wizard is only passed on by the server if it passes the spell-checker, otherwise the wizard is asked to correct the message. The student's input is also spell-checked. Messages exceeding a threshold of spelling errors are refused (i.e. not passed on to the wizard). The underlying rationale is that it would be implausible that an automated system could deal with such misspelled input.
We currently employ the spell-checker GNU Aspell[5] with the standard German dictionary provided with Aspell together with an extra dictionary of mathematical jargon. The latter was compiled from the introductory mathematics materials and gradually extended during the experiments.

Turn-Taking Control. DiaWOz-II imposes strict turn-taking on the student: once the student makes a turn, the sending of new messages is disabled (i.e., the dedicated button for "sending" is deactivated and displayed in a darker shade) until the tutor provides a response. Without this constraint, it might become unclear to which turn of the student an answer from the wizard belongs. However, the tutor is allowed to barge in at any time, which enables him to offer support or prompt if the student appears to be inactive.

3.5 Implementation

Figure 4 illustrates the architecture of DiaWOz-II. In order to customize the client interfaces, we have

- adapted the menu bars and buttons to the needs of our application and
- restricted the editing facilities so that the student can only type in a designated text area with all other TEXMACS functionalities disabled (for example, inserting an image, or editing the dialog history).

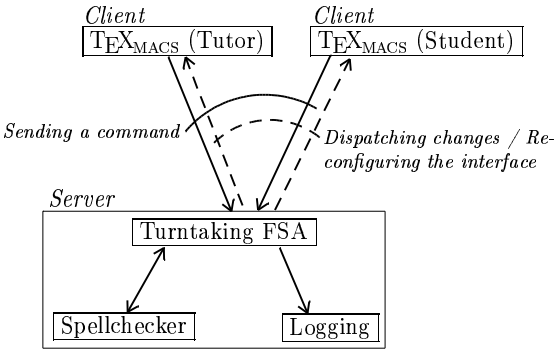


Fig. 4. System architecture

On the server side, turn-taking is controlled by a finite-state automaton. A message received by the server is written to the log-file and sent to a spell-checker. If it passes, it is broadcast to the clients. If it does not pass, it is

[5] `http://aspell.sourceforge.net/`

sent back to the sender for correction. Disallowing the student from sending new messages until the wizard makes a turn is technically realized by server messages to the student's client to reconfigure the client's interface (i.e. enable/disable the interface's elements according to the current state).

The combination of macros and *Scheme* provided in TeXmacs has turned out to be very useful for our development of DiaWOz-II. In particular, the amount of code we wrote (a dozen of *Scheme* files of approximately 100 Kb in total) is relatively small considering the implemented functionality, and it remained manageable over time (as opposed to the previous version of DiaWoZ that consisted of about 200 Kb Java code spread among 70 files). The environment enabled also people who are not professional software developers to participate in developing the system. Thus, TeXmacs has proven to be a good choice for our WOZ software, both from the point of view of the level of functionality it offers (word processing with LaTeX-like mathematical typesetting in a customizable editor) as well as from the point of view of prototyping and extending the software. The combination of the *Scheme* programming language with the large set of features already provided by TeXmacs allows for a lightweight, inclusive software development process.

4 An Empirical Study Using DiaWOz-II

Exploiting the DiaWOz-II system, we carried out a series of experiments in July 2005. In this study (see [8]), we collected a corpus of tutorial dialogs in German on mathematical proofs in the domain of binary relations. The collected data serves to investigate linguistic phenomena related to the mixing of mathematical formulae and natural language, underspecification phenomena, qualitative aspects of proof steps and mutual dependencies between natural language analysis and non-trivial mathematical domain reasoning.

4.1 Method

Thirty-seven students from Saarland University participated in the experiments. They were instructed to solve proof exercises collaboratively with a computer system that was described to them as a natural language dialog system on mathematics. This system was simulated with the DiaWOz-II software and four experts[6], who took the role of the wizard in turn (the set-up is shown in Fig. 5).

The wizards were given general instructions on the *Socratic* style of tutoring (cf. [14]), which is characterized by the use of questions to elicit information from the student. The tutors were instructed to reject utterances outside the mathematical domain and to respond in a uniform manner. Apart from that, the wizards were not restricted in the verbalization of their answers to the students.

[6] The experts consisted of the lecturer of a course *Foundations of Mathematics*, a maths teacher, and two maths graduates with teaching experience.

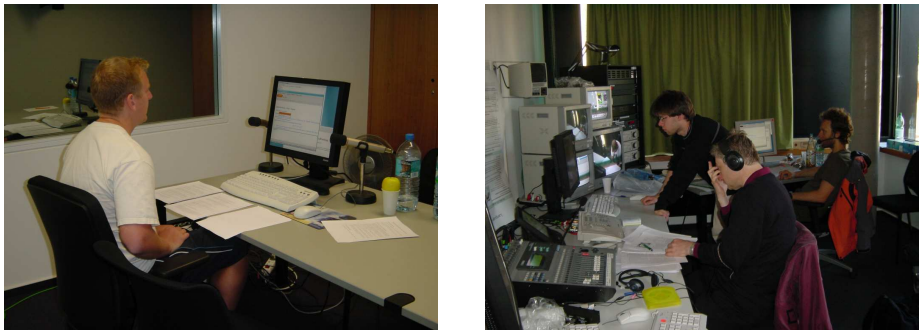


Fig. 5. An experiment in progress: The participant (left) and the wizard, experimenter and research assistant in the control room (right)

This allowed us to investigate the use of mathematical language without possibly influencing it by a-priori restrictions, even if more restrictions might have contributed to making the simulated system appear even more machine-like. In addition to the interaction window of DiaWOz-II, the tutors were provided with a second TeXmacs window in which they could save text and formulae for re-use.

The exercises were taken from the domain of relations, and were centered around the concepts of relation composition and relation inverse. Because of the advanced character of the exercises, the participants were required to have taken part in at least one mathematics course at university level. First, the subjects were required to fill out a questionnaire, asking about previous experiences with dialog systems and mathematics background. Subjects were also given study material with the mathematical definitions that were required to solve the exercises which was studied for approximately 25 minutes. The materials were handed out on paper and were also available as a TeXmacs document on the screen. This helped to achieve a uniform (and thus plausible) appearance of the system. Prior to the tutoring session, the students received a short introduction to the interface, during which the different modes of input for mathematical symbols – as menu items, as LaTeX commands or via commands in German language – and the copy & paste facility were demonstrated.

The largest part of the two-hour experimental session was allotted to the interaction between the student and the simulated system. In addition to the log-files recorded by DiaWOz-II, screen recordings were made. Furthermore, the participants were encouraged to "think aloud" and they were audio-recorded and filmed. This comprehensive collection of data not only documents the text of the tutorial dialogs, but also allows us to analyze how the participants used the interface and the study material.

At the end of the experimental session, the participants were required to fill out a second questionnaire asking about their verdict on the usability of the system, how difficult they found the exercises, and suggestions for improvements of the system.

S33: Nach Aufgabe W ist
$(S \circ (S \cup R)^{-1})^{-1} = ((S \cup R)^{-1})^{-1} \circ S^{-1}$
By Exercise W
$(S \circ (S \cup R)^{-1})^{-1} = ((S \cup R)^{-1})^{-1} \circ S^{-1}$
holds
T34: Das ist richtig! *That is correct!*
S34: Dies ist nach Theorem 1 gleich $(S \cup R) \circ S^{-1}$
This is by Theorem 1 equal to
$(S \cup R) \circ S^{-1}$
T35: Das ist auch richtig!
That is also correct!
S35: Ein Element (a,b) ist genau dann in dieser Menge, wenn es ein $z \in M$ gibt mit $(a,z) \in S \cup R$ und $(z,b) \in S^{-1}$
An element (a,b) is in this set exactly when there is a $z \in M$ *with* $(a,z) \in S \cup R$ *und* $(z,b) \in S^{-1}$
T36: Das ist korrekt! *That is correct!*
S36: Also $(b,z) \in S$. *Therefore* $(b,z) \in S$.
T37: Auch das ist korrekt!
That is also correct!
S37: DAmit [sic] kann aber nicht die geforderte Gleichheit bewiesen werden.
But with this the given equality cannot be proven.
T38: Das haben Sie gut erkannt! Was fehlt Ihnen dazu?
You have made a good observation! What is missing?

S25: $(R \cup S) \circ S = \{(x,y) \mid \exists\, z(z \in M \wedge ((x,z) \in R \vee (x,z) \in S) \wedge (z,y) \in S\}$
T28: Diese Beziehung hat Relevanz fuer die Aufgabe. Sie muessen sich also etwas dabei gedacht haben, als Sie diese vermuteten. Koennen Sie begruenden, wie sie mit der Aufgabe zusammenhaengt?
This relation is relevant for the exercise. You must have considered something as you proposed it. Can you justify how it is related to the exercise?
T29: Ihre Formel ist richtig.
Your formula is correct.
T30: Koennen Sie die Behauptung unter zusaetzlichen Voraussetzungen beweisen?
Can you prove the expression under additional assumptions?
S26: $\{(x,y) \mid \exists\, z(z \in M \wedge ((x,z) \in R \vee (x,z) \in S) \wedge (z,y) \in S\} = \{(x,y) \mid \exists\, z(z \in M \wedge (y,z) \in S \wedge ((x,z) \in R \vee (x,z) \in S))\} \Leftrightarrow ((y,z) \in S \wedge (z,y) \in S)\}$
T31: Auf der rechten Seite ist z nicht spezifiziert. Meinten Sie vielleicht $\Leftrightarrow$ oder etwas Aehnliches [sic] statt $\wedge$?
On the right side z is not specified. Do you perhaps mean $\Leftrightarrow$ *or something similar instead of* $\wedge$*?*

Fig. 6. Excerpts of dialogs in the two conditions: VM-group (left) and FM-group (right). English translations are given in italics. S_ and T_ indicate student and wizard turns, respectively.

4.2 Discussion

The experiments resulted in a large and diverse corpus of dialogs. During a session, a participant made on average 24 dialog turns, excluding those that were rejected for bad spelling. We briefly discuss how DiaWOz-II fulfilled its role, how the participants coped with the interface. Furthermore, the collected data hints at a potential influence of the interface features in combination with the reading material on the resulting tutorial dialogs.

Observations from the Corpus. An example of two dialog fragments from the experiment is given in Fig. 6. These dialogs were obtained under two different modes of presentation of the study material: formal (FM) vs. verbose (VM). Note

that the dialogs clearly differ in the employed mathematical style and that in Fig. 6 (right), the mathematical operations performed by the student can be characterized as term rewriting steps, i.e. a subformula of a term is replaced by an equivalent subformula. Also note that in Fig. 6 (right), the student uses no natural language. Even though all subjects were informed before the interaction that the system can handle a combination of natural language and formula input, we observed great variations in the amount of natural language used by the subjects.

Corpus analysis reveals differences in the use of natural language and mathematical expressions that was at least partially influenced by the mode of presentation of the study material. The group presented with the verbose material tended to use more natural language than the formal material group and the dialog turns of the VM-subjects contained more, but shorter, mathematical expressions. The formal material group tended to use more and longer formulas overall, and less natural language. More details on the differences in language production between the two conditions can be found in [15].

The copy & paste facilities provided by DiaWOz-II allowed copying definitions from the study material into the dialog contributions, and allowed copying previously uttered formulae for constructing new formulae. We observed that many subjects constructed larger and larger formulae with several levels of nesting. No such phenomenon was observed in the first study [6]. Even though the predecessor DiaWoZ software used in this study allowed copy & paste, this feature was not explained to the users and discovered only by some. Furthermore, in the first study the introduction material was only presented on paper, so that students could not copy from there as was possible in the second study. Another difference is the mathematical domain itself - the proofs concerning relations in the second experiment series require considerably longer formulae than those concerning naive set theory in the first experiment.

Usability of DiaWOz-II. The students were required to fill out post-experiment questionnaires, which among other things asked questions about the interface.

Students were asked if they had problems while using the interface, and to qualify their answer by a rating on a five-point scale between one (no problems) and five (great problems). Their ratings[7] (median 2, average 2.14, standard deviation 0.85) indicate that the participants generally had little trouble using the DiaWOz-II interface.

Even though a direct comparison between DiaWoZ and DiaWOz-II would require an experiment on its own (the two reported experiments involved different mathematical domains and different requirements imposed on the participants), these ratings are not far from those obtained in the first series of experiments

[7] The ratings from thirty-six participants are distributed as follows: A rating of 1 was assigned by 7 participants, a rating of 2 by 21 participants, a rating of 3 by 4 participants and a rating of 4 by 4 participants. No participant gave a rating of 5.

Table 1. Most frequent comments on the DiaWOz-II interface (number of participants indicated in brackets)

Positive Comments	
– Variety of formula input methods[1] (7)	– Interface is simple to use/clear (5)
– LaTeX commands available[1] (6)	– Questions can be formulated in NL (4)
– Math symbols in menu[1] (5)	
[1] In total, 20 subjects mentioned at least one positive aspect w.r.t. to formula input.	
Negative Comments	
– TeX$_{MACS}$-specific problems (14)	– Interface delay (10)
– Bad screen size/font size (8)	– Sending messages not via return key (6)
– No direct keyboard shortcuts for math symbols available (3)	

with DiaWoZ. There, students had also been asked the same question, where they indicated a rating of 1.59 on average and a median of 1.

A small number of participants commented to the experimenter that they suspect a human teacher. However, comments by other subjects indicated that these were convinced of having interacted with an automated system.

Participants were asked to give comments about the system in general and the interface in particular, which are summarized in Table 1. The fact that the input facilities of DiaWOz-II were positively mentioned by numerous participants can be contrasted with the first series of experiments, where eight of the seventeen participants complained that the sole input method for mathematical symbols via the menu bar required constant switching between the mouse and the keyboard for inputting mathematical formulae.

A serious criticism concerned the speed of the system. This refers to two aspects: (1) the fact that the students had to wait for the answers from the system, and (2) the behavior of the interface itself. The waiting times consisted in the time spent by the tutor to read the dialog contributions from the students and to write an answer (even with the help of pre-formulated answers), but also the message-passing between the client, the server and the spell-checker. An important fact was that the wizards were sometimes challenged by the size of formulae created by the students, which made checking them particularly time-consuming. The insufficient speed attributed to the system's interface refers to a small but noticeable delay when typing symbols in DiaWOz-II. This delay is not experienced when using a standard TeXmacs, but results from the extra mechanism that protects the dialog history from being edited mentioned above. Another criticism concerns the window layout. For the experiment we used a screen capturing software and a low screen resolution to save disk space, which was commented on negatively by the subjects.

In summary, the questionnaires show that the input methods for mathematical text available in DiaWOz-II were well received by many participants, but that other mainly technical difficulties remain. A possible improvement proposed by some of the participants is an option for the user to withdraw a message after

it is sent, in case the user himself becomes aware of a minor error and wants to correct it himself.

5 Conclusion

We have presented DiaWOz-II, our mediator software for WOZ experiments based on the *wysiwyg* editor TEXMACS. DiaWOz-II allows various modes of input for mathematical symbols, such as LATEX commands, customized commands and menu items, and editing facilities that allow for the creation of complex formulae. Furthermore, DiaWOz-II inherits high quality typesetting from TEXMACS. One purpose of this paper is to advocate DiaWOz-II to the AI community for similar WOZ studies in domains such as engineering, physics, economics, etc. where mathematical input in combination with natural language plays a crucial role.

We also briefly addressed the set-up and some results of a series of experiments conducted with DiaWOz-II. The corpus we obtained is important to guide our research in the DIALOG project. It is currently being evaluated and can be obtained from `http://www.ags.uni-sb.de/~dialog` (see [8] for a preliminary analysis). We have observed that the capabilities of DiaWOz-II for editing and copying mathematical formulae introduced artifacts into some of the tutorial dialogs that we collected, which we did not observe in the previous, similar experiment. These manifest themselves in a term-rewriting style of proving mathematical theorems leading to unnecessarily large and nested formulae. This hints at the importance of incorporating didactic knowledge into tutoring systems in our field (as simulated by DiaWOz-II) which prevent students from abusing such a system's features in a technology-driven manner, and to help the students to use these features purposefully and with moderation.

As a part of our ongoing work, we are combining the dialog specification mechanism from DiaWoZ with the DiaWOz-II system to obtain an environment that reflects our expertise gained with both systems. The DiaWOz-II system can be downloaded from `http://www.ags.uni-sb.de/~dialog/diawoz2`.

Acknowledgments. We would like to thank all of the members of the Dialog team for their input and comments on initial drafts of this paper, and of course for their contributions to DiaWOz-II and the experiments.

References

1. Fraser, N.M., Gilbert, G.N.: Simulating speech systems. Computer Speech and Language (5) (1991) 81–99
2. Kim, J.H., Glass, M.: Evaluating dialogue schemata with the Wizard of Oz computer-assisted algebra tutor. In: Intelligent Tutoring Systems. (2004) 358–367
3. Hoeven, J.v.d.: GNU TeXmacs: A free, structured, wysiwyg and technical text editor. In Flipo, D., ed.: Le document au XXI-ième siècle. Volume 39-40., Metz (2001) 39–50 Actes du congrès GUTenberg.

4. Dahlbäck, N., Jönsson, A., Ahrenberg, L.: Wizard of Oz studies – Why and how. Knowledge-Based Systems **6**(4) (1993) 258–266
5. Benzmüller, C., Fiedler, A., Gabsdil, M., Horacek, H., Kruijff-Korbayová, I., Pinkal, M., Siekmann, J., Tsovaltzi, D., Vo, B.Q., Wolska, M.: Tutorial dialogs on mathematical proofs. In: Proceedings of the IJCAI Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems, Acapulco (2003) 12–22
6. Benzmüller, C., Fiedler, A., Gabsdil, M., Horacek, H., Kruijff-Korbayová, I., Pinkal, M., Siekmann, J., Tsovaltzi, D., Vo, B.Q., Wolska, M.: A Wizard of Oz experiment for tutorial dialogues in mathematics. In: Proceedings of AI in Education (AIED 2003) Workshop on Advanced Technologies for Mathematics Education, Sydney, Australia (2003) 471–481
7. Fiedler, A., Gabsdil, M., Horacek, H.: A tool for supporting progressive refinement of wizard-of-oz experiments in natural language. In Lester, J.C., Vicari, R.M., Paraguaçu, F., eds.: Intelligent Tutoring Systems — 7th International Conference (ITS 2004). Number 3220 in LNCS, Springer (2004) 325–335
8. Benzmüller, C., Horacek, H., Lesourd, H., Kruijff-Korbayová, I., Schiller, M., Wolska, M.: A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material. In: Proceedings of International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy, ELDA (2006) To Appear.
9. Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N., Wang, A.: Suede: a wizard of oz prototyping tool for speech user interfaces. In: UIST. (2000) 1–10
10. Munteanu, C., Boldea, M.: MDWOZ: A Wizard of Oz environment for dialog systems development. In: Proceedings 2nd International Conference on Language Resources and Evaluation, Athens, Greece (2000) 1579–82
11. Anthony, L., Corbett, A.T., Wagner, A.Z., Stevens, S.M., Koedinger, K.R.: Student question-asking patterns in an intelligent algebra tutor. In Lester, J.C., Vicari, R.M., Paraguaçu, F., eds.: Intelligent Tutoring Systems. Volume 3220 of Lecture Notes in Computer Science., Springer (2004) 455–467
12. Hoeven, J.v.d., et al.: The TeXmacs manual. http://www.texmacs.org/tmweb/manual/web-manual.en.html (1999-2006)
13. Billingsley, W., Robinson, P.: Towards an intelligent online textbook for discrete mathematics. In: Proceedings of the 2005 International Conference on Active Media Technology, Takamatsu, Japan (2005) 291 – 296
14. Rosé, C.P., Moore, J.D., VanLehn, K., Albritton, D.: A comparative evaluation of socratic versus didactic tutoring. In: 23rd Annual Conference of the Cognitive Science Society, Edinburgh, Scotland (2001)
15. Wolska, M., Kruijff-Korbayová, I.: Factors influencing input styles in tutoring systems: the case of the study-material presentation format. In: Proceedings of the ECAI-06 Workshop on Language-enabled Educational Technology. (2006) To Appear.