Freie Universität Berlin
Institut für Informatik

Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III

– Preprint –

See also the published version by AKA Verlag,
as part of the DISKI series, volume 345,

EAN/ISBN: 978-3-89838-739-2

ALEXANDER STEEN

Dissertation

zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat)
am Fachbereich Mathematik und Informatik der Freien Universität Berlin

2018
BERLIN, DEUTSCHLAND

Title	Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III
Author	Alexander Steen
School	Freie Universität Berlin, Berlin, Germany
Supervisor	Prof. Dr. habil. Christoph Benzmüller
Second Examiner	Prof. Dr. Geoff Sutcliffe (University of Miami)

This dissertation was submitted to the Department of Mathematics and Computer Science of Freie Universität Berlin on 07.06.2018 and defended on 11.07.2018. The doctoral commission consisted of Prof. Christoph Benzmüller (FU Berlin, Head of commission), Prof. Raul Rojas (FU Berlin), Prof. Geoff Sutcliffe (University of Miami) and Dr. Klaus Kriegel (FU Berlin).

The dissertation project was conducted within the "Leo-III" project funded by the German Research Foundation (DFG) under grant BE 2501/11-1, and the project "Consistent Rational Argumentation in Politics" funded by the Volkswagenstiftung. All source code related to work presented in this thesis is publicly available (under BSD-3 license) at `https://github.com/leoprover/Leo-III`. Supplemental material is available at `https://alexandersteen.de/phd/`.

Abstract

In this thesis the theoretical foundations and the practical components for implementing an effective automated theorem proving system for higher-order logic are presented. A primary focus of this thesis is the provision of evidence that a paramodulation-based proof calculus can effectively be employed for performant equational reasoning in Extensional Type Theory (higher-order logic). To that end, a sound and complete paramodulation calculus for extensional higher-order logic with Henkin semantics is presented. The completeness proof hereby unifies and simplifies existing abstract consistency techniques for a formulation of higher-order logic that is based on primitive equality as sole logical connective.

In the practically motivated main part of this thesis, the design and architecture of the new higher-order theorem prover Leo-III is presented. Leo-III is based on the above paramodulation calculus and implements additional practically motivated inference rules including equational simplification routines such as heuristic rewriting and support for reasoning with choice. The system encompasses a flexible mechanism for asynchronous cooperation with first-order reasoning systems, a powerful proof search procedure and a sophisticated and efficient set of underlying data structures. Pragmatic and practically significant features of Leo-III are discussed, including its native support for polymorphic higher-order logic and reasoning in higher-order quantified modal logics. An evaluation on a heterogeneous set of benchmark problems confirms that Leo-III is one of the most effective and versatile higher-order automated reasoning systems to date.

The dissertation project was conducted within the ”Leo-III” project funded by the German Research Foundation (DFG) under grant BE 2501/11-1, and the project ”Consistent Rational Argumentation in Politics” funded by the Volkswagenstiftung. All source code related to work presented in this thesis is publicly available (under BSD-3 license) at `https://github.com/leoprover/Leo-III`. Supplemental material is available at `https://alexandersteen.de/phd/`.

Acknowledgements

First of all I want to express my gratitude to Christoph Benzmüller who introduced me to the broad field of automated reasoning, in particular in higher-order logic and further expressive non-classical logics, during my master's studies. I want to thank him for accepting me as PhD student and thereby entrusting me with the development of the Leo-III theorem proving system, the successor-in-spirit to his own successful systems. Christoph's passionate teaching, visionary ideas and challenging research questions greatly stimulated my own personal development and created a warm and motivating working environment. His strategic foresight and tight inclusion into scientific processes allowed me to quickly get accustomed with the relevant research community and renowned scientists of the field. For his inspiring support, his supervision and his invested time, I want to express my heartfelt thanks.

In the context of my studies at Freie Universität Berlin, I want to thank in particular Heinz Schweppe for his support and advice; Marcel Kyas for many joyful memories and his sustainable teaching; Lutz Prechelt for his unique ability to stimulate critical reflection; and to everyone else who gave lectures on computer science and mathematics to me. In want to thank my colleagues Oliver Wiese, Nadja Scharf, Jonas Cleve, Max Willert and many others with whom I spent many lunch breaks and had fruitful discussions. Concerning the Leo-III project I want to thank Tomer Libal from whom I learned about higher-order unification and Hans-Jörg Schurr, Maximilian Haslbeck and Tobias Gleißner who were wonderful to work with.

I want to thank Geoff Sutcliffe and Stephan Schulz who spent a lot of time creating integral parts of the automated theorem prover community (e.g. TPTP and the first-order system E, respectively) from which I learned a lot. There were many hours in which I studied the source code of the E prover and I think no teaching book on automated theorem proving could have ever taught me as much. Furthermore I'd like to thank Jasmin Blanchette, Chad Brown, Martin Suda, Cezary Kaliszyk and many others who welcomed me to the research community.

I am thankful for my friends; in particular Max Wisniewski, Marco Träger and Paul Podlech, with whom I spent a lot of time and had countless inspiring discussions. Above everyone else I want to thank my wife Giulia for her understanding and her emotional and professional support during my studies and beyond. I am furthermore indebted to my family Franziska and Pascal and in particular to my parents Karin and Bodo for their patience and their support. It was my parents who gave me my first personal computer when I was a child, which greatly influenced my interests and certainly stimulated my fascination for computer science.

— To my niece Ida.

QUOD SCRIPSI, SCRIPSI.

Contents

1. Introduction

1.1. Motivation

God necessarily exists — This is the final conclusion of Gödel's Ontological Argument which was part of his Nachlass and published post-mortem by Sobel [Sob87, Göd70, Sco72]. Ontological Arguments in general try to verify the existence of a ”God-like individual” and have a long tradition, going back at least to 1078 where Anselm of Canterbury tried to formulate a rational argument for the existence of God [Ans78]. In contrast to many previous formulations by various authors, Gödel worked on a sound and formal, that is mathematically rigorous, argument that can successfully stand a critical analysis of mathematicians and logicians, notwithstanding that the argument's assumptions themselves may be controversial among philosophers or theologians. Gödel's formulation elegantly incorporates modalities [BvBW06] and higher-order quantification [End15], i.e. operators expressing concepts of necessity and possibility, and quantification over predicates and sets, respectively, and is considered a masterpiece argument of metaphysics. Gödel's Ontological Argument certainly is one of the most discussed proofs of modern times and regularly a topic of academic philosophy classes; and, in particular, accepted as a sound mathematical proof.

Because of this it is even more surprising that only recently (around 2014) an automated deduction system found a major flaw in Gödel's argument that had been overlooked for decades by philosophers and mathematicians [BWP14]: The higher-order automated theorem prover LEO-II [BPST15] verified that the argument of Gödel is in fact inconsistent, hence allowing every conceivable statement as valid consequence (including obviously contradictory conclusions such as "every formula is a tautology", "truth and falsehood coincide", etc) [BWP16].[1] The results of these studies not only constitute new knowledge to the field of theoretical philosophy and demonstrate the capabilities of contemporary reasoning systems but also – and even more importantly – illustrate the limitations of human capabilities to safely maneuver within complex reasoning tasks such as non-trivial arguments formulated in a higher-order quantified modal logic as used by

[1] The example of Gödel's Ontological argument presented here serves mainly as a prominent motivating example, but itself constitutes an interesting and diverse field of research. Details on the results of the formal study of the argument and various variants of it can be found in the relevant literature [BWP14, BWP16].

Gödel in his Ontological Argument.

There are two key points that rendered the above discovery possible: Firstly, the existence of automated deduction systems for higher-order quantified logics. Although there exist different technical translation processes that allow the employment of first-order reasoning tools on higher-order problems [Ker94, MP08], these approaches are either incomplete with respect to Henkin semantics or ineffective for practical and complete automation of higher-order logics within first-order systems.[2] In the experiments of Benzmüller and Woltzenlogel Paleo, the higher-order reasoner LEO-II was in fact the only reasoning system that contributed the key higher-order inferences that were essential for the detection of the inconsistency in Gödel's argument [BWP14]. None of the existing powerful first-order theorem proving systems, even when combined with well-established translation mechanisms, could infer the necessary higher-order instances required for the proof. Secondly, the availability of techniques for automating quantified modal logics were vital for the success [BWP16]. In the experiments of Benzmüller and Woltzenlogel Paleo, a translation process is used that reduces second-order quantified modal logic formulas into equivalent formulas of classical higher-order logic using a shallow semantical embedding approach [BP13a]. This in turn allows the employment of powerful deduction systems for classical logics, such as LEO-II as described further above, in the context of modal logic.

Of course, Gödel's Ontological Argument is only one example for the use of higher-order formalisms. Boolos pointed out that there exists a family of simple theorems of first-order logic that cannot practically be proven within any first-order calculus as the number of necessary inference steps grows similar to the Ackermann function [Boo87]. In a higher-order calculus, however, there exist simple and short proofs for these theorems that essentially make use of the expressiveness of the term language for formulating a suitable induction principle [BB07]. In the context of computer science, the expressiveness of higher-order logic can e.g. be used to concisely formulate correctness specifications of software. Furthermore there exist various language extensions for higher-order logic that allow an even more versatile employment in computer science [Far08].

[2] Note that automation of higher-order logic is usually studied with respect to so-called Henkin semantics [Hen50] which is essentially as expressive as first-order logic. However, aspects of extensionality and comprehension complicate automation of relevant classes of higher-order problems within first-order reasoning systems. Nevertheless, first-order reasoning systems can often be fruitfully exploited by higher-order deduction systems for discharging higher-order proof obligations that are essentially first-order or require only few higher-order reasoning steps [BN10b]. As a prominent example, encoding of higher-order language features for cooperation with first-order reasoners is implemented by the Sledgehammer [BBP13] system within the interactive proof assistant Isabelle/HOL [NWP02] for semi-automatic discharging of (simple) higher-order goals.

The practical evidence that quantified modal logics can effectively be modeled as a fragment of classical higher-order logic furthermore suggests that higher-order logic can serve as an *universal meta-logic* [Ben17b]. This claim is further substantiated by more recent evaluations [BOR12, GSB17, SB18] and the observation that there exist analogous reductions for numerous further non-classical logics [BS16, SB16, Ben17a, Ben17c, BFP18], for many of which there exist none or only few specialized reasoning systems. A strong foundation for the automation of higher-order logic and an effective implementation of a corresponding deduction system thus enables computer-assisted reasoning in even more, practically relevant, logical systems and application areas. This observation has been a core inspiration of this thesis project.

The described situation is, however, in contrast to the current state of higher-order logic automation: Although there is increasing interest in higher-order reasoning and there has been significant progress, including the development of new systems and substantial improvement of existing ones [Ben15a], the theoretical and practical foundations of higher-order automated reasoning are still not as mature and well-developed as their first-order counterpart [BM14]. This is due to the fact that a significant amount of research in automated deduction focused on first-order logic and even more restricted systems (such as SAT techniques [Kro09]) and, as a consequence, many powerful proof calculi and practically relevant implementation techniques exist mainly for such systems [RV01]. One particular family of representatives in the context of first-order logic with equality are superposition-based calculi that have proven an effective basis for reasoning systems and provide a powerful notion of redundancy [BG90, NR92, BG94]. Superposition calculi make use of so-called term orderings that impose a partial ordering on the terms of first-order logic and additionally incorporate further properties such as totality on ground terms and stability under substitutions [NR01]. The idea is that generating inferences can be restricted to maximal (sub-)terms, reducing the number of overall generated inferences while retaining completeness. Unfortunately, simple adaptions of such term orderings do not exist for the full language of higher-order logic.[3] This seems to render superposition-based approaches for Henkin-complete higher-order reasoning in the full language impossible.

This work aims at constituting a first-step towards effective equational rea-

[3] A simple counter example for the existence of such an ordering, denoted $\succ$ here, is the following cycle unsuitable for any strict ordering relation: $c \equiv_\beta (\lambda X. c)\ c \succ c$, in particular implying that $c \succ c$. Here, the first relationship denotes meta-logical β-equivalence between the constant c and the application term $(\lambda X. c)\ c$, and the $\succ$ step follows from the subterm property which is a core property of usual term orderings as employed by superposition.

soning in higher-order logic. To that end, a complete paramodulation-based calculus is presented that treats equality as a language primitive rather than a defined notion. Although this calculus is still unordered and thus potentially suffers from the same drawbacks as experienced in first-order paramodulation, including early state space explosions and a prolific proof search, the idea is to anyway use this calculus as a basis for a new automated theorem proving system and to pragmatically tackle the problems on the implementation level. The resulting theorem proving system, called *Leo-III*, uses a higher-order term ordering to heuristically restrict generating inferences and to enable equational simplification procedures such as unconditional rewriting. Since this sacrifices completeness in general, the goal is to find a suitable set of search heuristics to nevertheless produce an effective system for practical applications. As initially motivated, various non-classical logics may be automated using an encoding to classical higher-order logic. In order to offer out-of-the-box automation for such logics, the Leo-III system provides native pre-processing routines that automatically apply the appropriate technical translations and hence eliminate the need for external, error-prone, encoding work.

In the remainder of this chapter, first, a semi-formal exposition to higher-order logic is presented alongside a brief motivating survey of its history. Subsequently, the origin and principles of automated deduction are summarized, focusing on the beginnings of higher-order automated reasoning and its recent developments. Finally, an overview over the Leo theorem prover family is presented, specifying the goals of this thesis within the Leo-III research project.

1.2. Higher-Order Logic

The term higher-order logic refers to expressive logical formalisms that allow for quantification over predicate and function variables. While the term is inherently ambiguous and there are many formal systems of such kind, today higher-order logic typically denotes – at least in the context of the automated deduction community – systems based on a typed λ-calculus [BDS13]. The use of quantification over arbitrary types and its mechanism of explicit binding allow a concise representation of complex mathematical concepts and structures [And02a, Far08].

Logical systems admitting quantification beyond first-order, that is (restricted) quantification about sets of objects, were first studied by Frege in the 1870s as attempt to formalize the foundations of mathematics on the basis of formal logic [Zal17]. In his *Begriffsschrift* [Fre79] Frege presented a second-order predicate logic calculus and a notion of proofs that can be used to formally deduce mathematical properties within that calculus. The most remarkable result in this context is probably the formal proof of the Dedekind/Peano axioms for number theory as a theorem of the system [Zal17]. Furthermore, first concepts of function substitutions were included, comparable to rudimentary substitutions of λ-abstractions. Consequently, the work by Frege can be seen as the first genuine higher-order logic as it is understood today. However, in subsequent work that aimed at establishing a theory for arithmetics within his system, Frege included axioms related to *comprehension principles*[4] that ultimately rendered the system inconsistent as it was susceptible to a paradox as Russell pointed out in 1902 [VH67, Rus96]. This paradox involves the construction of a set of all sets that do not contain themselves, today widely known as Russell's Paradox. To overcome the inconsistencies, Russell proposed a formal type theory, which was first depicted in an appendix called *doctrine of types* [Rus03] and further elaborated in his *ramified type theory* [Rus08] that differentiates between objects and sets of that objects. The strict distinction between different natures of objects was, admittedly, already known to Frege, who distinguished predicates from individuals in his calculus.

An alternative formulation of type theory was presented 1940 by Church who employed a formal system based on functions rather than sets as primitives and simplified the formulation of Russell in various aspects [Chu40]. Additionally, the system includes an syntactical representation for functions in terms of λ-abstractions and explicit conversion rules for them [Chu32, Chu41]. The resulting system is referred to as *Simple Type Theory* (STT), Church's Type Theory and sometimes classical higher-order logic (HOL).[5]

HOL is a typed logic. This means that every syntactical object of HOL carries a unique and fixed type that indicates what sort of object the term represents: Starting from a set of base types (or atomic types), the set of simple types is freely generated by these base types and function types. A function type is generated by simple juxtaposition, i.e. if τ and ν are types, then $\nu\tau$ represents the type of functions from objects of type τ to objects of type ν. Usually, there is one dedicated base type ι representing the type of objects from the universe of discourse. In HOL, there is no strict separation between formulas and terms as it is explicitly constructed in first-order logic [Fit96]. This is realized by des-

[4] Comprehension axioms guarantee the existence of certain sets or functions, e.g. an example of an *unrestricted* comprehension axiom is the assertion that there exists a set that consists of exactly those objects that satisfy a given predicate.

[5] Although the term HOL is *not* identified with STT but rather with a subsystem of STT throughout this thesis, the differences are of no importance for most principles and ideas of HOL introduced here. The term HOL is specified to coincide with the ExTT subsystem of STT further below.

ignating a dedicated type o as the type of Boolean terms. Types are written as subscript to terms as in t_τ which can be read as: "t_τ is a term of type τ". The syntax of HOL is minimalistic: Given a logical signature Σ (that is a collection of typed constant symbols) and a set of variable symbols $\mathcal{V}$, terms of HOL are constructed inductively using only the following four rules (τ and ν being two type meta-variables):

Constant:	If $c_\tau \in \Sigma$, then c_τ is a term of type τ.
Variable:	If $X_\tau \in \mathcal{V}$, then X_τ is a term of type τ.
Abstraction:	If $X_\tau \in \mathcal{V}$ and s_ν is a term of type ν, then $(\lambda X_\tau . s_\nu)_{\nu\tau}$ is a term of type $\nu\tau$.
Application:	If $s_{\nu\tau}$ is a term of type $\nu\tau$ and t_τ is a term of type τ, then $(s_{\nu\tau}\, t_\tau)_\nu$ is a term of type ν.

Terms of HOL are based on work by Schönfinkel [Sch24] that studied the fact that functions of more than one argument can be represented by functions of one argument that themselves produce functions as values. The reduction to monadic functions is today also referred to as *currying*[6], named after Haskell Curry who used this technique in his studies of combinatory logic [Cur30].

As noted above, formulas of HOL are simply terms of type o and may consequently also occur as proper subterm. In particular (and in contrast to first-order logic), formulas (or predicates) may be used as arguments to predicates such as $\big(p_{oo}\ (q_{o\iota}\ x_\iota)\big)$ and as arguments to (non-Boolean) terms, as in $(f_{\iota o}\ p_o)$ or even $(f_{\iota(ooo)}\ \vee_{ooo})$. Furthermore, λ-abstractions allow the ad-hoc construction of functions in the language of HOL: If s_ν is a term of type ν, then $(\lambda X_\tau . s_\nu)_{\nu\tau}$ denotes the function that is constructed from mapping each object x_τ of type τ to $s[X/x]$, where $s[X/x]$ is identical to s except that every occurrence of X in s is replaced by x. Hence, λ-abstractions intuitively represent functions with the difference that they do not carry any explicit name. Application of a given term t_τ to such a function is then given by $(\lambda X_\tau . s_\nu)_{\nu\tau}\ t_\tau$ which can be transformed by so-called β-conversion to $s[X/t]$ as desired. The exact rules how to manipulate λ-abstractions within HOL are given by a set of conversion rules including α-, β- and η-conversion (cf. formal introduction of HOL in §2) that have been investigated by Church and others in the context of HOL's underlying λ-calculus [Chu32, Chu41, Bar81, BDS13]. Another advantage of HOL is that the presence of λ-abstractions and their conversion rules as sketched

[6] Although *schönfinkeling* of even *fregeing* would be more accurate as Frege was probably the first who introduced this technique in his works [Fre93, §§35 – 37] and Schönfinkel further developed this technique.

above already make type-restricted comprehension principles[7] derivable in the system [And02a].

HOL comes with built-in notions of functional and Boolean extensionality, denoted EXT$^{\nu\tau}$ and EXT^{o}, respectively [BBK04a, Def. 4.5]. These principles can be formulated as

$$\begin{aligned}\text{EXT}^{\nu\tau} &:= \forall F_{\nu\tau}.\, \forall G_{\nu\tau}.\, (\forall X_\tau.\, F\,X =^{\nu} G\,X) \Rightarrow F =^{\nu\tau} G \\ \text{EXT}^{o} &:= \forall P_o.\, \forall Q_o.\, (P \Leftrightarrow Q) \Rightarrow P =^{o} Q\end{aligned}$$

and state that two functions are equal (where $=^{\tau}_{o\tau\tau}$ is the equality predicate on type τ, written in infix notation for reasonsof readability) if they correspond on every argument and that two formulas are equal if they are equivalent (where $\Leftrightarrow_{ooo}$ denotes equivalence), respectively. Note that the opposite directions are always implied by substitutivity of equality. Using these principles, one can infer that two functions such as $\lambda P_o.\, \top$ and $\lambda P_o.\, P \vee \neg P$ are in fact equal, which is of great importance in many applications of equational reasoning in HOL (where $\top_o$ denotes syntactical truth and $\vee_{ooo}$ represents logical disjunction). Using the same principles one can infer that $\lambda P_o.\, \lambda Q_o.\, P \vee Q$ and $\lambda P_o.\, \lambda Q_o.\, Q \vee P$ are equal.

A prominent example for the expressiveness of HOL is the formulation of Cantor's Theorem that states that the powerset of a set is strictly larger than the set itself. In HOL, sets of type τ are associated with predicates of type $o\tau$ denoting the characteristic function of that set. Then, a variant of Cantor's statement can concisely be expressed as follows:

$$\neg\exists f_{o\tau\tau}.\, \forall Y_{o\tau}.\, \exists X_\tau.\, (f_{o\tau\tau}\, X_\tau) =^{o\tau}_{o(o\tau)(o\tau)} Y_{o\tau}$$

This formula captures the essence of Cantor's Theorem by formally expressing that there does not exist a surjection f from a set of objects of type τ to its powerset. No further axioms or auxiliary constructions are required.

Simple Type Theory as proposed by Church contains various axioms that are not considered valid in HOL as assumed throughout this thesis. Axioms that have been assumed by Church but are usually neglected in modern formulations of HOL are the axiom of infinity (stating that the successor function on Church numerals is injective [BM14, §3.6]), the axiom of description (asserting the existence of an operator that chooses a unique element satisfying a certain predicate) and the existence of at least two individuals. When omitting these

[7] An example of a type-restricted comprehension axiom scheme is the formula $\exists F_{\nu\overline{\tau^n}}.\, \forall \overline{X^n}.\, F\; \overline{X^n} = g_\nu$ stating that there exists a function $F_{\nu\overline{\tau^n}}$ (or set if $\nu \equiv o$) that corresponds to a given term (predicate) g_ν, where F is not a free variable of g_ν. Such a function F is easily constructed within HOL by $\lambda\overline{X^n}.\, g_\nu$ using λ-abstraction.

three principles, the resulting logical system is often denoted *Extensional Type Theory* (ExTT) [BM14] and constitutes the logical basis of most automated theorem proving systems for higher-order logic. This thesis hence follows the work by Henkin [Hen50] and many others and identifies the term *classical higher-order logic* (HOL) with ExTT. Note that ExTT still contains extensionality principles[8] and might, depending on the system at hand, also validate the axiom of choice. Yet another subsystem, denoted *elementary type theory* (ETT) is constructed from removing the extensionality principles (and possibly choice) from ExTT [And74]. ETT roughly corresponds to an extension of first-order logic with quantification over arbitrary types and a term language based on λ-terms. Note that, however, basic extensionality principles are not valid in ETT.[9] While the version of HOL considered throughout this thesis corresponds to ExTT without the axiom of choice, the implementation of the automated theorem proving system presented as a contribution of this thesis does in fact support reasoning with the axiom of choice for pragmatic reasons (it can nevertheless be disabled manually if necessary).

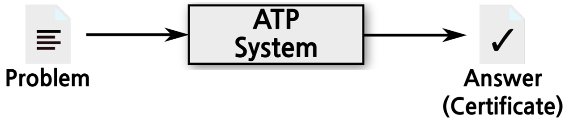
Unfortunately, as a consequence of Gödel's Incompleteness Theorem, all proof calculi for STT (ExTT, ETT) with standard semantics are necessarily incomplete [Göd31], thus eliminating the hope of effective automation. Henkin introduced a generalized notion of semantics for HOL (also referred to as *Henkin semantics*) in which completeness can be achieved, i.e. that provability in ExTT corresponds to validity in all *general models* [Hen50]. Note that standard models for HOL are subsumed by general models (or *Henkin models*) such that every valid formula with respect to general semantics is also valid in the standard sense. The key idea of general semantics is that the domain of functional types need not necessarily be the complete set of total functions but rather a subset of all functions such that every term can be assigned a denotation in a reasonable way. Andrews' work corrected a technical flaw in Henkin's construction that allowed models that did not validate functional extensionality principles [And72a] and clarified the notion of general semantics using a more direct definition that uses elements from combinatory logic [And72b]. For the remainder of this thesis, HOL with general semantics is assumed, unless stated otherwise.

Various theoretical and practical advantages of using HOL are highlighted by

[8] Strictly speaking, Church did not add an axiom for Boolean extensionality into STT but merely discussed the possibility of doing so. The formulation of type theory considered here follows the work by Henkin who did include the axiom in his formulation of type theory [Hen50].

[9] Technically, since many systems for ETT are based on a λ-calculus that includes η-conversion, a weak form of functional extensionality can be derived within that systems. This notion of extensionality is nevertheless strictly weaker than the one included in ExTT [BBK04a].

Figure 1: Bird's-eye perspective on automated theorem proving as a black box.



Farmer, including a clear and uniform syntax, a simple semantics and pragmatic flexibility [Far08]. The expressivity of higher-order logic is however not only exploited by mathematicians or logicians, but also in the field of formal methods in computer science and engineering (e.g. for software and hardware verification [Gup92]) and in computer linguistics [Sch15]. A more recent and innovative field of study is the application of automation of expressive logics in theoretical philosophy, also referred to as *Computational Metaphysics* [FZ07, BWP16], for formalizing and assessing abstract concepts and their consequences. To that end, classical higher-order logic can be used to emulate a large number of expressive logics using a semantic embedding approach [Ben11]. Hence, efficient automation of higher-order logic also permits automation results for practical relevant logics in e.g philosophy, such as quantified modal logics or conditional logics [BP13a, Ben17a]. Note that there are only few systems available even for propositional fragments of such logics. When considering quantified modal logics for example, the situation is even worse and generally restricted to a few special semantical settings. For the case of higher-order quantified systems, there are currently no deduction systems available. Hence, automated theorem provers for HOL can effectively close the gap for such expressive quantified logics.

1.3. Automated Theorem Proving

Automated theorem proving (ATP) denotes the automation of deduction procedures that, given a set of assumptions (usually given as axioms) and a conjecture as input, use a computer program to validate or reject the statement that the input conjecture is a *logical consequence* of the given set of assumptions. Intuitively speaking, a formula is a logical consequence if it is not possible that all assumptions are validated while the conjecture is not (cf. Def. 2.6 for a formal definition). In the context of ATP systems, this reasoning process is done fully automatically, in particular without any further user-interaction. ATP systems are used in academic and industrial applications, such as in planning [Rin09], software and hardware verification [Kro09, CRSS95], or knowledge-based systems [BCM$^+$03]. Particularly successful applications are restricted variants of

SAT and SMT reasoning within decidable logic fragments [BHvMW09].

Figure 1 displays a schematic top-level view on an automated reasoning process. Here, the ATP system is given a problem that consists of a set of formal logical formulas assumed as axioms and, possibly, a dedicated formula serving as the conjecture. In a most simplistic setting, the output of an ATP system is some form of a yes-or-no answer, depending on whether the conjecture could be proven or refuted, respectively. Increasingly many ATP systems additionally output a *proof object* that certifies the afore stated answer. Such a proof certificate can then be used to assess the system's proof for correctness or further additional information.

Early developments. The historical overview of early developments in propositional and first-order automated reasoning presented in the following loosely follows MacKenzie [Mac95], Bachmair and Ganzinger [BG01] as well as Davis [Dav01] for which the author refers to for a more detailed discussion.

Early automated deduction approaches include the development of a decidable procedure for the addition of integers by Presburger [Pre29] in 1929 and its implementation on a vacuum tube computer by Davis in 1954. As expected, the deductive power of that implementation was still very limited, also because the complexity of Presburger's procedure is at least doubly exponential [FR98]. Nevertheless, simple fundamental theorems about integers could be proven automatically [Dav83]. Also during that time, Newell, Shaw and Simon presented their "Logic Theory Machine" [NS56] for reasoning in propositional logic. The Logic Theory Machine found proofs for 38 of 52 selected theorems from the *Principia*, in particular a simpler proof to a specific theorem, compared to the one given by Whitehead and Russell [Dru09]. While the system of Newell et al. was not complete, Wang and Gao presented a Gentzen-style proof system in 1987 which was complete for propositional logic and outperformed the Logic Theory Machine on examples from the Principia [Wan83].

The field of first-order automated theorem proving as it is established today can be traced back to the 1920s and early 1930s where Skolem and Herbrand provided important basic results that later formed the foundations of modern theorem proving software [Dav83]. While the first theorem proving systems for first-order logic based on that results suffered from poor deductive effectivity (e.g. that by Gilmore [Gil60]), subsequent work by Davis and Putnam [DP60] as well as Prawitz [Pra60] prepared the path for more powerful reasoning techniques.

One of the most popular machine-oriented proof procedures, the resolution calculus, was presented by Robinson in 1965 [Rob65] and revolutionized the field of automated deduction. In contrast to previous automation approaches, the resolution calculus was comparably simple to implement and could be executed by a computer program quite effectively. In resolution-based theorem proving and, more generally, modern saturation-based theorem proving, the initial conjecture is negated, transformed into a set of clauses, and successively saturated. If after some number of inferences the empty clause is inferred, a contradiction is derived from the initial set of clauses hence establishing the validity of the initial conjecture. The resolution calculus is refutationally complete for first-order logic, i.e. the empty clause can be derived from every inconsistent clause set by resolution. Nevertheless, it was observed that a naive resolution search process would produce a large number of clauses that do not necessarily contribute to the desired proof. This stimulated a, to this day ongoing [Sch17], search for effective search heuristics that aim at reducing the search space within resolution-based theorem proving systems, including the unit strategy [WCR64] and the set-of-support search strategy [WRC65]. Popular powerful ATP systems for first-order logic based on (extensions of) Robinson's ideas include Otter [MW97], EQP [McC97a] SPASS [WDF$^+$09], E [Sch02] and Vampire [KV13] which all shaped the field of theorem proving as it is today.

Today, there exist a number of different calculi well-suited for automation of first-order logic, including (variants of) resolution-based calculi [BG01], paramodulation based calculi including superposition [NR01], tableaux methods [Häh01] and connection calculi [Bib87, OB17]. The large number of different systems already indicates that each system comes with advantages and disadvantages regarding different aspects, depending on their application domain. However, in the last decades, superposition-based ATP systems have shown to be particularly effective tools for general first-order reasoning.

Automation of higher-order logic. Foundations for theorem proving systems for higher-order logic (also generally referred to as *classical type theory* to that time) were not an extensive topic of research until the 1960s and 1970s, where Robinson presented a tableaux calculus for HOL [Rob69]. Further notable calculi of that time include Andrews's higher-order resolution principle [And71], Jensen's and Pietrowski's approach [PJ72] and the constrained resolution method by Huet [Hue72, Hue73a]. Common to all of these approaches is the intrinsic problem of the undecidability of higher-order unification [Gol81, Hue73b] which complicates all mechanization attempts. Andrew's resolution principle hence avoids unification completely and rather uses an enumeration of the universe, whereas Huet's procedure postpones unification until the end of a refutation attempt and rather uses pre-unification [Hue75] in order to avoid blind guessing as included in full higher-order unification.

The automation approaches and calculi developed until the 1990s [Wol09, Koh94, Koh95] still lacked a calculus-level handling of extensionality principles and thus required the input problem to contain explicit extensionality axioms in order to guarantee completeness with respect to Henkin semantics [Hen50]. This led to the development of extensional higher-order resolution $\mathcal{ER}$ [BK98a] as well as higher-order extensionality paramodulation $\mathcal{EP}$ and higher-order extensional RUE resolution $\mathcal{ERUE}$ [Ben99b]. While $\mathcal{ER}$ already was complete for HOL with Henkin semantics without any additional extensionality axioms, the calculi $\mathcal{EP}$ and $\mathcal{ERUE}$ focused on the handling of primitive equality in extensional HOL. However, in all three calculi, primitive equality is handled by expansion into its definition as proposed by Leibniz[10]. As investigated later by Benzmüller et al., Leibniz equality axioms effectively (just as extensionality axioms) enable cut-simulation [BBK09] and hence should be avoided at all costs for automation attempts.

Higher-order automated theorem proving has recently made major progress and several sophisticated ATP systems for higher-order logic have been developed, including Satallax [Bro12], Isabelle/HOL [NWP02] and LEO-II [BPST15].

Higher-order ATP systems. In this paragraph, a number of popular interactive and automated theorem proving systems for higher-order logic are surveyed:

TPS. One of the earliest system for classical type theory was developed by Andrews and Cohen, originating from experiments for combining Andrews' resolution approach with Huet's unification procedure. Subsequently, the system was completely revised and improved, yielding the system today known as TPS [ABI+96, AB06]. TPS makes heavy use of mode scheduling for its automatic mode, but can also be used semi-automatically and interactively.

[10] The *Identity of Indiscernibles* (also known as *Leibniz's law*) refers to a principle first formulated by Gottfried Leibniz in the context of theoretical philosophy [Lei89]. The principle states that if two objects X and Y coincide on every property P, then they are equal, i.e. $\forall X_\tau . \forall Y_\tau . \; (\forall P_{o\tau} . P\, X \Leftrightarrow P\, Y) \Rightarrow X = Y$. where "=" denotes the desired (primitive) equality predicate. Sometimes the principle is identified with the conjunction of the above statement and its backward implication, yielding $\forall X_\tau . \forall Y_\tau . \; (\forall P_{o\tau} . P\, X \Leftrightarrow P\, Y) \Leftrightarrow X = Y$. Since this principle can easily be formulated in HOL, it is possible to encode equality in higher-order logic without using the primitive equality predicate. An extensive analysis of the intricate differences between primitive equality and defined notions of equality is presented by Benzmüller et al. [BBK04a] to which the author refers to for further details.

HOL. The HOL prover family, starting with HOL88 and HOL98, is based on LCF-style theorem proving [GMW79] in which a rich proof assistant is built on top of a small, trusted logical kernel. Current members of the HOL family are HOL4 [GM93], ProofPower, HOL Zero and the minimalistic HOL Light [Har09].

Isabelle/HOL. The Isabelle [Pau88] system is a theorem prover for an intuitionistic variant of type theory. On top of that, Isabelle/HOL [NWP02] is designed as an interactive proof assistant for HOL that includes sophisticated proof tactics, provides an expressive specification language and comes with a continuously growing library of formalized theories, the *archive of formal proofs* [KNP03].

ΩMEGA. The proof planner and interactive proof assistant ΩMEGA [SBA06] for HOL includes, similar to Isabelle/HOL, various proof tactics and subsystems including LEO [BK98b] (cf. further below) and first-order provers for discharging proof obligations. ΩMEGA aims at providing a deduction system for mathematics and mathematical education.

Satallax. A particular powerful ATP system for extensional type theory is Satallax [Bro12]. It implements a complete ground tableaux calculus for HOL with choice in which successively generated higher-order inferences are associated with propositional clauses. These clauses are checked for unsatisfiability using an external SAT solver.

Nitpick. The higher-order counter-model finder Nitpick [BN10a] systematically explores resp. enumerates finite model structures for establishing the (counter-)satisfiability of a set of formulas. Nitpick is integrated into Isabelle/HOL and is usually used for checking the consistency of user-provided axiomatizations or to quickly generate counter-examples to given conjecture.

AgsyHOL. The higher-order ATP system AgsyHOL is based on generic lazy narrowing [Lin08] and implemented in Haskell. A main focus of the system is to provide checkable proof certificates. The sequent-style proofs of AgsyHOL can e.g. be verified using the Agda system [BDN09].

Zipperposition. The first-order Zipperposition system is based on superposition and supports data types, arithmetic and rewriting [Cru15]. It has quite recently been extended to higher-order logic. However, the support for reasoning in HOL is still experimental.

This list is by no means exhaustive; a more extensive description of higher-order reasoning systems is presented by Benzmüller and Miller [BM14].

Recent developments. In more recent history, automated theorem provers have been successfully applied to relevant mathematical problems. The first computer-assisted proof of an open mathematical problem was conducted in 1976 with the proof of the four-color problem (colorings of maps with four colors) that used an exhaustive case distinction analysis [AH76]. Another prominent example is the proof of Robbins conjecture (identities in a special algebra denoted Robbins algebra) with the help of the theorem prover EQP in the late 1990s [McC97b]. Quite recently, Kepler's conjecture (optimal sphere packing in three dimensions) was proved using computer-assisted reasoning systems [Lag11].

Recently, the expressivity of higher-order logic has been exploited for encoding various expressive non-classical logics within HOL. Semantical embeddings of, among others, higher-order modal logics [BP13a, GSB17], conditional logics [Ben17a], many-valued logics [SB16], deontic logic [BFP18], free logics [BS16], access control logics [Ben09] and combinations of such logics [Ben11] can be used to automate reasoning within the respective logic using ATP systems for classical HOL. A prominent result from the applications of automated reasoning in non-classical logics, here in quantified modal logics, was the detection of a major flaw in Gödel's Ontological Argument [FB17, BWP17] as well as the verification of Scott's variant of that argument [BWP15] using the LEO-II theorem prover [BPST15] and the interactive proof assistant Isabelle/HOL [NWP02]. Similar techniques were used to assess discussions between philosophers regarding non-trivial arguments from the field of metaphysics [BWWP17].

Additionally, Isabelle/HOL and the Nitpick system were used to assess the correctness of concurrent C++ programs against a previously formalized memory model [BWB$^+$11]. The proof assistant HOL Light played a key role in the verification of Kepler's conjecture within the Flyspeck project [HAB$^+$17].

1.4. The Leo Systems

This dissertation project's research and development is conducted within the project "Leo-III", funded by the German National Research Foundation (DFG) under grant BE 2501/11-1, that is conducted by Benzmüller (PI). Benzmüller already developed successful reasoning systems for higher-order logic in the past, such as the automated theorem provers LEO [BK98b] and LEO-II [BPST15]

Figure 2: The logo of the Leo-III theorem prover. The three heads underline the fact that it is the third incarnation of a Leo system and also symbolize its initial layout as massively parallel system.

which pioneered the area of resolution-based theorem proving for Henkin semantics. In particular, the latter system found international acclaim as successful reasoners and won the international CASC competition in 2010 [Sut10].

LEO (or LEO-I) was designed as a resolution-based automated theorem prover component of the proof assistant and proof planner Ωmega [SBA06]. Similar to Isabelle's Sledgehammer system [BBP13], LEO's task was to automatically solve given subgoals originating from the Ωmega system in order to minimize the need for user interaction. The LEO system already supported calculus-level treatment of extensionality principles and used first-order ATP systems for cooperation [BSJK08]. However, LEO lacked proper handling of native equality as it expanded occurrences of equality by Leibniz' definition. LEO was hard-wired into the Ωmega system and could not be used as a stand-alone ATP system.

The LEO-II ATP system is based on Resolution by Unification and Extensionality [DH86], adapted its predecessors treatment of extensionality principles and supports native treatment of equality [BPST15]. Furthermore, LEO-II has been integrated into the proof assistant Isabelle.

The goal of the Leo-III project is to turn the LEO-II prover into a prover based on paramodulation and term orderings which benefits from massive parallelism in the form of a multi-agent blackboard architecture. From the beginning of the project, careful attention is paid to a robust foundation of efficient data structures [Ste14] and compatibility with other proof systems by complying to relevant standards of the TPTP infrastructure [Sut17b].

This dissertation project plays an integral part in the realization of the following goals of the Leo-III project:

- Implementation of a stand-alone ATP system for HOL with Henkin semantics based on a paramodulation calculus that additionally uses a

higher-order term ordering for restricting the inferences and hence reducing the size of the prover's search space.
- Theoretical justification of the work above, in particular for HOL with a primitive notion of equality.
- Development of flexible means for collaboration with external reasoning systems, including higher-order model finders and first-order theorem provers.
- Provision of detailed proof objects that can be used for assessing the correctness of the system's results as well as for communicating with other proof systems.
- Integration of the ATP system into an interactive proof assistant for discharging user-defined sub-goals.
- Development and employment of effective term representation techniques for higher-order automated reasoning, including representation of free variables.
- Evaluation of the system on heterogeneous sets examples from higher-order reasoning and modal logic reasoning.
- Dissemination of Leo-III, in particular making the system freely available within the SystemOnTPTP infrastructure as well as foster developments concerning standards for reasoning in non-classical logics such as modal logic.

Note that the development of means for massive parallel proof search using independent agents is not in the scope of this thesis and not yet part of the Leo-III system. Nevertheless, Leo-III already is one of the most effective higher-order ATP systems available. Despite its relative young age, Leo-III came in second place at the CADE ATP System Competition (CASC) in 2017 [Sut17a], convincingly beating its predecessor LEO-II. With Leo-III's native support for reasoning in polymorphic HOL and higher-order quantified modal logics, it is additionally, up to the author's knowledge, the most widely applicable ATP available to date.

The code of the project, including the code that was developed within the dissertation project, can be found at the Leo-III project repository at GitHub under `https://github.com/leoprover/Leo-III`.

1.5. Structure of the Thesis

The dissertation can roughly be separated into two parts: In the first, theoretical, part of the thesis, §2 formally introduces the syntax and semantics of higher-order logic, followed by the presentation of an extensional paramodulation calculus for that logic in §3, including respective proofs of soundness and completeness. In the second, practically oriented, part of this thesis, the Leo-III system and its associated components are presented. This includes the design and architecture of the Leo-III prover, including details about its proof search, data structures and implementation details, is discussed in §4. Subsequently, the versatile application domains of the Leo-III prover are presented in §5, containing examples of higher-order, polymorphic and modal logic reasoning. An extensive evaluation of the Leo-III system is then presented in §6. Finally, §7 concludes the dissertation and sketches areas of further work.

The interested reader is referred to Appendix A, where detailed instructions for obtaining, installing and using the Leo-III system are presented. Appendix B summarizes the contributions of this thesis. Detailed proofs of Leo-III originating from the application examples of §5 are presented in-full in Appendix C. A brief German summary of the thesis is provided in Appendix D. Finally, a short CV of the author is given in Appendix E.

2. Higher-Order Logic

In this chapter, the syntax and semantics of higher-order logic is introduced as well as notions and definitions that are used throughout the thesis. The term higher-order logic (HOL) is used interchangeably with extensional type theory (ExTT) as described by Henkin. A brief survey over further higher-order formalisms is presented at the end of the chapter.

Different notions of equality will be used in the following: If a concept is defined, the symbol $:=$ is used, e.g. as in $\mathcal{D}_o := \{\mathrm{T}, \mathrm{F}\}$ (cf. further below). The equality relation of HOL, written $=^\tau$ denotes a logical constant from the signature such that $s_\tau =^\tau t_\tau$ is a term of the logic (assuming s_τ and t_τ are). Meta equality $\equiv$ denotes identity between objects.

2.1. Syntax of HOL

HOL is a typed logic. This means that all terms of HOL are associated a fixed and unique *type*. Intuitively, a type can on the level of semantics be regarded as a representation for a specific collection of objects that inhabit that particular type and denote the possible values for each term's interpretation of that type.

Types. The types of HOL are given by so-called *simple types* of the underlying typed λ-calculus. Let $\mathcal{S}$ be a non-empty set of sort symbols which serve as syntactic identifier for the base types of the logic. The set $\mathcal{T}$ of simple types is then freely generated by[1]

$$\tau, \nu ::= s \in \mathcal{S} \mid (\nu\tau) \tag{1}$$

where types of the first form are called *base types* and types of the latter form are *function types*.[2] A function type $(\nu\tau)$ is the type of total functions from objects of type τ to objects of type ν. Parentheses may be omitted whenever possible. By convention, function types are left-associative, i.e. the type $\mu\nu\tau$ is identical to $((\mu\nu)\tau)$. If $\tau \equiv \nu_n \cdots \nu_1 \nu$ is a function type and ν_n is a base type, $\mathrm{goal}(\tau) \equiv \nu_n$ is called the *goal type* of τ. For base types τ, $\mathrm{goal}(\tau) \equiv \tau$.

[1] The symbol $::=$ denotes an abstract syntax production rule.

[2] Another widespread notation of function types uses the $\rightarrow$ sign. Here, $\nu\tau$ corresponds to $\tau \rightarrow \nu$ (note the inverted order of both operands). Especially for programmers, the notation with arrows might be more convenient to read. Nevertheless, already for comparably small examples, this representation quickly becomes too large. Of course, both versions can be thought of interchangeable.

In the context of HOL, the set of base type symbols $\mathcal{S}$ is chosen to be $\mathcal{S} := \{\iota, o\}$ where ι stands for the type of individuals, i.e. the type of objects from the universe of discourse, and o is the type of Boolean truth values.

Terms. Let Σ_τ be a set of constant symbols of type $\tau \in \mathcal{T}$ and let $\Sigma := \bigcup_{\tau \in \mathcal{T}} \Sigma_\tau$ be the union of all typed symbols, called a *signature*. Let further $\mathcal{V}$ denote a set of (typed) variable symbols. From these the terms of HOL are constructed by the following abstract syntax ($\tau, \nu \in \mathcal{T}$):

$$s,t ::= c_\tau \in \Sigma \mid X_\tau \in \mathcal{V} \mid (\lambda X_\tau . s_\nu)_{\nu\tau} \mid (s_{\nu\tau}\, t_\tau)_\nu \tag{2}$$

The terms are called *constants*, *variables*, *abstractions* and *applications*, respectively. The set of all terms of type τ over a signature Σ is denoted $\Lambda_\tau(\Sigma)$ and $\Lambda_\tau^c(\Sigma)$ if only ranging over all closed terms. Likewise, the set of all terms is denoted $\Lambda(\Sigma)$. It is assumed that the set $\mathcal{V}$ contains countably infinitely many variable symbols for each type $\tau \in \mathcal{T}$ and that there exists some cardinal $\aleph_s$ such that all sets Σ_τ are of cardinality $\aleph_s$.

The type of a term is written as subscript and considered a part of its name but may be dropped if clear from the context.[3] By convention, parentheses are omitted whenever possible and application is assumed to be left-associative. Furthermore the scope of an λ-abstraction's body reaches as far to the right as is consistent with the remaining brackets. Nested applications $s\, t^1 \, \dots \, t^n$ may also be written in vector notation $s\, \overline{t^n}$, s is called its *head symbol*. Variables are denoted by capital letters such as X_τ, Y_τ, Z_τ and, more specifically, the variable symbols P_o, Q_o and $F_{\nu\tau}, G_{\nu\tau}$ are used for predicate or Boolean variables and variables of functional type, respectively. Analogously, lower case letters s_τ, t_τ, u_τ denote general terms and $f_{\nu\tau}, g_{\nu\tau}$ are used for terms of functional type.

The notion of free and bound variables are defined as usual. The set of free variables of a term t is denoted by $\mathrm{fv}(t)$. A term t is called *ground* if $\mathrm{fv}(t) \equiv \emptyset$.

[3] The here presented typing semantics approach is sometimes called *Church-style* [Pie02, p. 111] (or intrinsic interpretation) and commonly used in the context of higher-order logic and throughout this thesis. Another approach is the so-called *Curry-style* [Pie02, p. 111] (or extrinsic interpretation) that is rather used in the context of programming languages and logical frameworks. In the first, the type of a term is considered a part of its name and is thus fixed and cannot change between different contexts. This also means that all terms are automatically well-typed terms since we cannot (syntactically) construct terms that are not well-typed. The latter variant views types as an additional information assigned to terms from some external context. Consequently, terms itself do not carry any type information: The term $\lambda x.\, x$ can syntactically be constructed and the typing $\lambda x.\, x : \tau \to \tau$ and $\lambda x.\, x : \nu \to \nu$ (where $\tau \neq \nu$ are types) can be inferred depending on the context the term occurs in. Both styles are equally expressive since they can simulate each other [Rey98b, p. 327ff]. Curry-style can be mimicked by Church-style using type-erasure, and the other way around using type reconstruction.

Syntactical equivalence of terms, written $s_\tau \equiv t_\tau$ for two terms $s_\tau, t_\tau \in \Lambda_\tau(\Sigma)$, is assumed modulo α-conversion, i.e. consistent renaming of bound variables within a term. A substitution σ is a mapping from variables to terms of the same type. It is extended to terms in the usual way and its application to terms is written in post fix notation, e.g. as in $s\sigma$. Application of substitutions is assumed to be capture-free.[4] A finite substitutions σ is written $\sigma \equiv \{s^1/X^1, \dots, s^n/X^n\}$ and denotes the substitution that replaces the variable X^i by s^i, $1 \leq i \leq n$.

The equality predicate, denoted $=^\tau$, for each type τ is assumed to be the only logical connective present in the signature Σ. All (potentially) remaining constant symbols from Σ are called parameters. A formulation of HOL based on equality as sole logical connective originates from Andrew's system $\mathcal{Q}_0$ [And02b, And63]. Figure 1 shows the definitions of the remaining logical connectives that are defined in terms of equality. For simplicity, the binary logical connectives may be written in infix notation, e.g. the term $p_o \lor q_o$ formally represents the application $(\lor_{ooo}\, p_o\, q_o)$. Also, binder notation is used for universal and existential quantification: The term $\forall X_\tau. s_o$ is used as a short-hand for $\Pi^\tau\, (\lambda X_\tau. s_o)$ and analogously for existential quantification. A Σ-*formula* is a term from $\Lambda_o(\Sigma)$ of type o and a Σ-*sentence* if it is a closed Σ-formula. The reference to Σ may be omitted if clear from the context.

An alternative possibility is to regard the logical connectives $\neg, \lor$ and Π^τ as primitives of the logic. The remaining logical connectives are then defined, in particular the equality predicate can be defined using the definition from Leibniz' principle, i.e. to have $\doteq := \lambda X_\tau. \lambda Y_\tau. \forall P_{o\tau}. P\, X \Rightarrow P\, Y$, where $\doteq$ denotes the thereby defined equality predicate (to distinguish it from a possibly existing primitive equality connective). However, Leibniz equality $\doteq$ only denotes a proper equality predicate if the underlying model structure contains the identity relation for every type [And72a, BBK04a]. The use of equality as sole logical connective is motivated by the fact the subsequent proofs generally have to cover fewer cases and are hence more compact. In a practical implementation, the usual connectives might nevertheless be regarded as primitive in order to avoid costly expansion and comparison procedures.

2.2. Semantics of HOL

HOL is built on top of the simply typed λ-calculus. The notions of conversion including α-, β- and η-conversion are defined as usual [BDS13] and denoted $\longrightarrow_\alpha, \longrightarrow_\beta$ and $\longrightarrow_\eta$, respectively. The symmetric closure of $\longrightarrow_\star$ is written

[4] This can always be achieved using appropriate renaming of bound variables using α-conversion [BDS13].

Figure 1: Definitions of logical connectives based on equality. The remaining connectives such as disjunction, implication, existential quantification etc. can be defined as usual.

$$\begin{array}{lll}
\top_o & := & =^o_{ooo} =^{ooo}_{o(ooo)(ooo)} =^o_{ooo} \\
\bot_o & := & (\lambda P_o . P) =^{oo} (\lambda P_o . \top) \\
\neg_{oo} & := & \lambda P_o . P =^o \bot \\
\wedge_{ooo} & := & \lambda P_o . \lambda Q_o . (\lambda F_{ooo} . F \top \top) =^{o(ooo)} (\lambda F_{ooo} . F\ P\ Q) \\
\Leftrightarrow_{ooo} & := & \lambda P_o . \lambda Q_o . P =^o Q \\
\Pi^\tau_{o(o\tau)} & := & \lambda P_{o\tau} . P =^{o\tau} \lambda X_\tau . \top
\end{array}$$

$\longleftrightarrow_\star$, the transitive and reflexive closure of these relations is denoted $\longrightarrow^*_\star$, for $\star \in \{\alpha, \beta, \eta\}$. α-conversion is treated implicitly when necessary in the following.

For $\star \in \{\beta, \eta, \beta\eta\}$, two terms $s_\tau, t_\tau \in \Lambda(\Sigma)$ are *$\star$-equivalent*, written $s_\tau \equiv_\star t_\tau$, if and only if $s_\tau \longleftrightarrow^*_\star t_\tau$, i.e. both terms can be rewritten into the same term using successive applications of $\longleftrightarrow_\star$. Since the simply typed λ-calculus is confluent and terminating every term s_τ has a unique β-normal form ($\beta\eta$-normal form) which is denoted $s{\downarrow_\beta}$ ($s{\downarrow_{\beta\eta}}$). Then, the condition $s_\tau \longleftrightarrow^*_\beta t_\tau$ ($s_\tau \longleftrightarrow^*_{\beta\eta} t_\tau$) is equivalent to $s_\tau{\downarrow_\beta} \equiv t_\tau{\downarrow_\beta}$ ($s_\tau{\downarrow_{\beta\eta}} \equiv t_\tau{\downarrow_{\beta\eta}}$). A detailed review about HOL's underlying λ-calculus and its properties can be found in the literature [BDS13].

For the semantics of HOL a universe $\mathcal{U}_0$ of non-empty sets with the usual properties is assumed [GP94]. Each type is represented by some set within $\mathcal{U}_0$, i.e. the set of all denotations associated to that particular type. Given a type $\tau \in \mathcal{T}$, the set of denotations for τ is referred to by $\mathcal{D}_\tau \in \mathcal{U}_0$. The exact structure of those $\mathcal{D}_\tau$ is given by the notion of frames:

Definition 2.1 (HOL Frame).

Let $\mathcal{D} \equiv (\mathcal{D}_\tau)_{\tau \in \mathcal{T}}$ be a family of sets. $\mathcal{D}$ is called a *HOL frame* (or simply frame) if

(i) $\mathcal{D}_\iota \not\equiv \emptyset$,
(ii) $\mathcal{D}_o := \{\mathrm{T}, \mathrm{F}\}$, and
(iii) $\mathcal{D}_{\nu\tau} \subseteq \mathcal{D}_\nu^{\mathcal{D}_\tau}$,

where $\mathrm{T} \not\equiv \mathrm{F}$ are distinct objects representing truth and falsehood, respectively. $\mathcal{D}_\nu^{\mathcal{D}_\tau}$ is the set of all total (set-theoretic) functions from $\mathcal{D}_\tau$ to $\mathcal{D}_\nu$. ⌟

A frame $\mathcal{D}$ is called *standard* if $\mathcal{D}_{\nu\tau}$ is chosen to be the full set $\mathcal{D}_\nu^{\mathcal{D}_\tau}$. Given a signature Σ and a standard frame $\mathcal{D}$, an *interpretation* $\mathcal{I}$ on $\mathcal{D}$ is a set-theoretic function $\mathcal{I}$ that maps each constant symbol $c_\tau \in \Sigma$ of type $\tau \in \mathcal{T}$ to some (arbitrary) element $d \in \mathcal{D}_\tau$. $\mathcal{I}(c_\tau)$ is called the *denotation of* $c_\tau \in \Sigma$. In an appropriate model for HOL, it is additionally assumed that each equality constant symbol is associated the respective (set-theoretic) identity relation by $\mathcal{I}$.

Definition 2.2 (Standard HOL Model)**.**
Let Σ be a signature with $\{=^\tau_{o\tau\tau} \mid \tau \in \mathcal{T}\} \subseteq \Sigma$. A *standard HOL model* is a pair $\mathcal{M} \equiv (\mathcal{D}, \mathcal{I})$ where $\mathcal{D}$ a standard frame and $\mathcal{I}$ is an interpretation on $\mathcal{D}$ such that for each $\tau \in \mathcal{T}$ it holds that

$$\mathcal{I}(=^\tau_{o\tau\tau})(x,y) \equiv \mathrm{T} \text{ if and only if } x \equiv y$$

for all denotations $x, y \in \mathcal{D}_\tau$. ⌟

A *variable assignment* g is a function mapping variables $X_\tau \in \mathcal{V}$ of type $\tau \in \mathcal{T}$ to some element of $\mathcal{D}_\tau$. For a term s_τ and a variable X_τ, let $g[s/X]$ denote the variable assignment that is identical to g, except that X is mapped to s.

Finally, the *value* of a HOL term under a given model $\mathcal{M}$ is determined by a valuation function $\|.\|$. The valuation $\|.\|$ maps terms s_τ of type $\tau \in \mathcal{T}$ to their denotation within $\mathcal{D}_\tau$, relative to a model $\mathcal{M}$ and a variable assignment g.

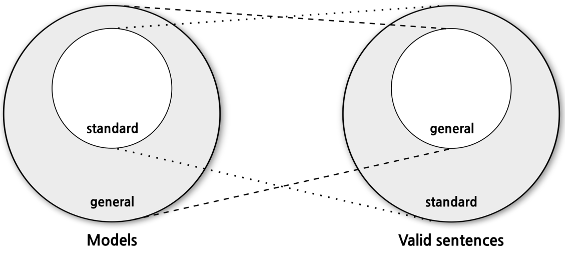
Definition 2.3 (Valuation function)**.**
Let Σ be a signature, $\mathcal{M} \equiv (\mathcal{D}, \mathcal{I})$ a standard HOL model and let g be a variable assignment. A *valuation function* $\|.\|^{\mathcal{M},g}$ is given by ($c_\tau \in \Sigma, X_\tau \in \mathcal{V}, s_{\tau\nu}, t_\nu \in \Lambda(\Sigma)$):

$$\begin{aligned}
\|c_\tau\|^{\mathcal{M},g} &:= \mathcal{I}(c) \\
\|X_\tau\|^{\mathcal{M},g} &:= g(X) \\
\|\lambda X_\tau. t_\nu\|^{\mathcal{M},g} &:= z \mapsto_{set} \|t\|^{\mathcal{M},g[z/X]} \\
\|s_{\tau\nu}\ t_\nu\|^{\mathcal{M},g} &:= \|s\|^{\mathcal{M},g}\left(\|t\|^{\mathcal{M},g}\right)
\end{aligned}$$

where $\mapsto_{set}$ denotes a set-theoretic function from $\mathcal{U}_0$. For $s_\tau \in \Lambda_\tau(\Sigma)$, $\|s\|^{\mathcal{M},g}$ is called the *value of s with respect to model $\mathcal{M}$ and variable assignment g* (or simply the *value of s*). ⌟

As a consequence of Gödel's Incompleteness Theorem [Göd31], HOL with standard semantics is necessarily incomplete [BM14, §4]. However, Leon Henkin developed a generalized notion of HOL models, so called *general models* or *Henkin models*, that allow a meaningful notion of semantics in which completeness can be achieved [Hen50]. Intuitively, the idea of general models is that the domain

Figure 2: Relationship between standard and general models and their valid sentences: The larger the class of considered models, the lower is the number of generally valid sentences in all models.



$\mathcal{D}_{\nu\tau}$ of functional types $\nu\tau \in \mathcal{T}$ need not necessarily be the complete set of functions from τ to ν but merely a subset of those functions with the restriction that there are still enough denotations for every syntactical object.

Definition 2.4 (General HOL Model)**.**
Let Σ be as in Def. 2.2. A *general HOL model* is a pair $\mathcal{M} \equiv \big(\mathcal{D},\mathcal{I}\big)$ where $\mathcal{D}$ is a (possibly non-standard) frame and $\mathcal{I}$ is an interpretation on $\mathcal{D}$ as in Def. 2.2 such that, for each variable assignment g, the valuation $\|.\|^{\mathcal{M},g}$ is a total function. ⌟

The above restriction that $\|.\|^{\mathcal{M},g}$ is a total function ensures that every term denotes with respect to every variable assignment. A less indirect definition of general semantics is due to Andrews and asserts the existence of certain combinators in the respective semantical domains [And72b]. For general HOL models $\mathcal{M}$, the valuation function $\|.\|^{\mathcal{M},g}$ is well-defined and uniquely determined. For the remainder of this thesis, HOL with general semantics is assumed.

For the above defined general semantics, also referred to as *Henkin semantics*, sound and complete machine-oriented calculi exist [And71, Hue72, PJ72, Ben99a]. Note that every standard HOL model is also a general HOL model and hence every valid sentence in the general sense is also valid in the standard sense. This situation is visualized in Fig. 2. The remaining usual connectives such as $\neg$, $\vee$ etc. are given their proper denotation by $\|.\|$ in general and standard semantics:

Lemma 2.5 (Properties of usual connectives [BBK04a, pp. 1046 – 1047])**.**
Let Σ and $\mathcal{M} \equiv (\mathcal{D},\mathcal{I})$ be as in Def. 2.4 and let g be a variable assignment. It holds that

(1) $\|\top\|^{\mathcal{M},g} \equiv \mathrm{T}$,

(2) $||\bot||^{\mathcal{M},g} \equiv \mathrm{F}$,
(3) $||\neg s||^{\mathcal{M},g} \equiv \mathrm{T}$ if and only if $||s||^{\mathcal{M},g} \equiv \mathrm{F}$,
(4) $||s \lor t||^{\mathcal{M},g} \equiv \mathrm{T}$ if and only if $||s||^{\mathcal{M},g} \equiv \mathrm{T}$ or $||t||^{\mathcal{M},g} \equiv \mathrm{T}$,
(5) $||\Pi^\tau\, P_{o\tau}||^{\mathcal{M},g} \equiv \mathrm{T}$ if and only if $||P||^{\mathcal{M},g}(u) \equiv \mathrm{T}$ for every $u \in \mathcal{D}_\tau$. □

Validity and consequence are defined as usual:

Definition 2.6 (Validity, Consequence)**.**
Let $\mathcal{M}$ be a general HOL model, g is variable assignment and $s_o \in \Lambda_o(\Sigma)$ a formula.

(i) s_o is *valid with respect to model $\mathcal{M}$ and variable assignment g*, denoted $\mathcal{M}, g \models s_o$ if and only if $||s||^{\mathcal{M},g} \equiv \mathrm{T}$.
(ii) s_o is *valid with respect to model $\mathcal{M}$*, denoted $\mathcal{M} \models s_o$, if and only if $\mathcal{M}, g \models s_o$ for every variable assignment g.
(iii) s_o is *valid*, denoted $\models s_o$, if and only if $\mathcal{M} \models s_o$ for every general HOL model $\mathcal{M}$.
(iv) Let $t_o \in \Lambda_o(\Sigma)$ be a formula. s_o is called a *consequence of t_o*, denoted $t_o \models s_o$, if and only if for every general HOL model $\mathcal{M}$ such that $\mathcal{M} \models t_o$ it holds that $\mathcal{M} \models s_o$.
(v) Let $\Phi \subseteq \Lambda_o(\Sigma)$ be a set of formulas. s_o is called a *consequence of Φ*, denoted $\Phi \models s_o$, if and only if for every general HOL model $\mathcal{M}$ such that $\mathcal{M} \models t_o^i$, for every $t_o^i \in \Phi$, it holds that $\mathcal{M} \models s_o$. ⌟

Further studies of more general classes of HOL models can be found in the literature, including intensional models [Mus07], generalizations to model structures without functional and/or Boolean extensionality [BBK04a] and the even more general v-complexes by Andrews [And71]. Since this thesis aims at effective automation of HOL with Henkin semantics, a discussion of these more general models is omitted here.

2.3. Related Systems

There exist various alternatives intended to serve as a foundation of mathematics, most notably axiomatic set theory [Zer08] due to Zermelo and Fränkel, von Neumann's set theory [VN25] and, more recently, so-called univalent foundations related to homotopy type theory [PW12].

In the context of higher-order logic, there too exist systems that modify or extend Church's Simple Theory of Types as introduced above. As an example, Andrews' presented the system Q_0 [And63] that only employs a single rule of inferences (in contrast to the Hilbert-style calculus of STT) and bases its logic on equality as sole logical connective, similar to the term language of HOL presented in §2.1. Note that, however, the system Q_0 is essentially equivalent to Church's STT except for the choice of the primitive connectives resp. relations.

A more expressive type system underlying a higher-order logic was studied by Gordon in the context of the HOL family [GP94, GM93]. Here, types are allowed to contain type variables, yielding a polymorphic variant of Church's STT. Furthermore, Andrews' system Q [And65] and Melham's so-called "Extended HOL" [Mel93] permit quantification over types, and even more general systems account for type abstraction as e.g. the one underlying HOL2P [Völ07].

Dependent type theories form another rich field of expressive systems that include functions from terms to types as part of the language. Popular implementations of dependent type systems are PVS [ORR+96], Automath [DB70], Coq [Pau11] and Agda [BDN09].

3. Higher-Order Paramodulation

Paramodulation extends resolution by a native treatment of equality on the calculus level. It was developed in the late 1960s by G. Robinson and L. Wos [RW69] and introduced as an attempt to overcome the shortcomings of resolution-based approaches to handling equality. A paramodulation inference incorporates the principle of *replacing equals by equals* and can be regarded as a conditional rewriting step. However, unrestricted paramodulation was quickly found to be unsuited for practical applications as a large number of redundant inferences are generated that do not contribute to the overall proof goal [NR01]. In the context of first-order reasoning, superposition-based calculi [BG90, NR92, BG94] overcome this weakness by imposing ordering restrictions on the premises of the paramodulation rule such that redundant clauses (with respect to the ordering) are not generated in the first place. The orderings used by these approaches are *term orderings* $\succ$ that satisfy certain properties, including stability under substitutions (if $s \succ t$ then $s\sigma \succ t\sigma$ for every substitution σ) and totality on ground terms (or, a weaker condition, $\succ$ must be completeable). The use of term orderings originates from the field of rewriting, in particular from work by Knuth and Bendix who presented a procedure that transforms a set of rewriting rules into an equivalent set that is confluent and terminating [KB70, Hue81]. However, due to the more complex structure of the term language of higher-order logic, there do not exist suitable term orderings that allow a straight-forward adaption of first-order superposition to HOL.

A further complication of proof search in higher-order logic is intrinsic to its more complicated meta-theory: In first-order refutation-based calculi, proof search is generally layed out as a combination of proof search on the calculus-level (i.e. by means of generating inferences) and a dedicated set of meta operations such as clausification, further pre-processing and unification. In this context, unification is used as a filtering mechanism that restricts inference attempts between incompatible, that is non-unifiable, configurations of clauses. The unification procedure is hereby an isolated subroutine that does not interact with the remaining proof search routines. In particular, first-order unification has linear time complexity [MM76, WM78] and thus provides an efficient auxiliary mechanism to the overall proof search. In the context of higher-order logic, unification is neither decidable nor unitary [Hue73b, Gol81], however semi-decidable. As a consequence, unification cannot be used as a decision routine by machine-oriented calculi for HOL in general. Huet proposed to ignore unification completely and postpone unification attempts until the end of proof search [Hue72, Hue73a] where all produced inferences are tested for unifiability. It turns out that such an approach is unfeasible as a very large number of inferences are produced and even a bounded unification search for all of them takes too long. However, under the assumption that undecidable problems do not occur often in practical applications[1], a hybrid approach can be employed: Unification constraints are encoded into the inferences of calculus rules and can eagerly be tackled by (bounded) unification subroutines. If successful, the resulting unified clauses are inserted into the proof search – if not, the unification constraints are kept until possibly solvable after further inferences. Such an approach was taken by Kohlhase [Koh94] and further developed by Benzmüller [Ben99b]. Since calculus rules may also generate non-CNF inferences, clausification is too lifted to the calculus level and, consequently, interleaved with the overall proof search.

Higher-order paramodulation for extensional type theory was first presented by Benzmüller [Ben99a, Ben99b]. However, this calculus was mainly theoretically motivated and extended a resolution calculus with an additional paramodulation rule instead of being based on a paramodulation rule alone. Additionally, that calculus contained a rule that additionally expanded equality literals by their Leibniz definition. As Leibniz equality axioms effectively enable cut-simulation [BBK09], the proposed calculus seems unsuitable for automation. The approach presented here, in contrast, avoids the expansion of equality predicates but adapts the use of dedicated calculus rules for extensionality principles.

In this chapter, an extensional paramodulation for extensional type theory is presented. It does not include a resolution rule and treats equality as a primitive, rather than defined, notion. The resulting calculus EP is shown to be sound and complete for HOL with Henkin semantics. For the completeness argument, a model existence theorem is presented which rests on variants of abstract consistency conditions that simplify the ones presented by Benzmüller et al. in the context of equality as the only logical connective. Finally, the problems of imposing orderings constraints into the calculus are discussed.

3.1. Preliminaries

Since the calculi discussed throughout the thesis work with formulas in conjunctive normal form, the input formulas are reformulated to a clausal representation

[1] The experiments of the author do in fact support this assumption.

and subsequently transformed into equisatisfiable sets of clauses in clause normal form (cf. further below).

An *equation* is a tuple of terms $s, t \in \Lambda$, written $s \simeq t$. A *literal* ℓ is a signed equation, i.e. an equation together with a polarity $\alpha \in \{\mathtt{tt}, \mathtt{ff}\}$, written $\ell \equiv [s \simeq t]^{\alpha}$. , Polarities denote the intended truth-value of a literal. The opposite polarity $\overline{\alpha}$ to a polarity α is given by $\overline{\mathtt{tt}} \equiv \mathtt{ff}$ and $\overline{\mathtt{ff}} \equiv \mathtt{tt}$. Non-equality predicates such as $(p_{o\iota}\ c_{\iota})$ are represented as an explicit equation with the Boolean truth-value $\top$ as in $(p_{o\iota}\ c_{\iota}) \simeq \top$. A literal $\ell \equiv [s \simeq t]^{\alpha}$ is said to be *equational* if and only if neither s nor t are identical to $\top$ or $\bot$. By convention, non-equational literals ℓ are represented as literals $\ell \equiv [s \simeq \top]^{\alpha}$, i.e. where the right-hand side of the underlying equation is identical to $\top$. Every non-equational literal can be transformed to this representation using one of the three (validity preserving) transformation rules below.

$$[\top \simeq s]^{\alpha} \longrightarrow [s \simeq \top]^{\alpha} \qquad [s \simeq \bot]^{\alpha} \longrightarrow [s \simeq \top]^{\overline{\alpha}} \qquad [\bot \simeq s]^{\alpha} \longrightarrow [s \simeq \top]^{\overline{\alpha}}$$

Non-equational literals $[s \simeq \top]^{\alpha}$ may simply be written $[s]^{\alpha}$. A negative equational literal ℓ is also referred to as *unification constraint*. It is called a *flex-flex* constraint if ℓ is of the form $\ell \equiv [X\ \overline{s^i} \simeq Y\ \overline{t^i}]^{\mathtt{ff}}$, where X, Y are variables. Given a model $\mathcal{M}$, a variable assignment g and a literal $\ell \equiv [s_{\tau} \simeq t_{\tau}]^{\alpha}$, ℓ is valid with respect to $\mathcal{M}$ and g, written $\mathcal{M}, g \models \ell$, if and only if $\mathcal{M}, g \models s = t$ and $\alpha \equiv \mathtt{tt}$ or $\mathcal{M}, g \models s \neq t$ and $\alpha \equiv \mathtt{ff}$. It is valid with respect to a model $\mathcal{M}$, written $\mathcal{M} \models \ell$, if and only if $\mathcal{M}, g \models \ell$ for every g. (Un-)Satisfiability is defined as usual.

Clauses are multisets of literals, denoting their disjunction. A clause $\mathcal{C}$ of literals $\ell_1, \ldots, \ell_n$ is written $\mathcal{C} \equiv \ell_1 \vee \ldots \vee \ell_n$. For uniformity, for two clauses $\mathcal{C}, \mathcal{D}$ and a literal ℓ, let $\mathcal{C} \vee \mathcal{D}$ and $\mathcal{C} \vee \ell$ denote the multiset union $\mathcal{C} \uplus \mathcal{D}$ and $\mathcal{D} \uplus \{\ell\}$, respectively. A clause $\mathcal{C} \equiv \ell_1 \vee \ldots \ell_n$ is valid with respect to a model $\mathcal{M}$ and a variable assignment g, denoted $\mathcal{M}, g \models \mathcal{C}$, if $\mathcal{M}, g \models \ell_i$ for some $1 \leq i \leq n$. It is valid with respect to a model $\mathcal{M}$ whenever $\mathcal{M}, g \models \mathcal{C}$ for every variable assignment g. Again, (un-)satisfiability of clauses is defined as usual. A clause is the *empty clause*, denoted $\square$, if it only consists of flex-flex constraints. This is motivated by the fact that flex-flex unification problems can always be solved, and hence any clause only consisting of flex-flex constraints is unsatisfiable [Hen50]. In particular, $\mathcal{M} \not\models \square$ for any HOL model $\mathcal{M}$.

A literal $\ell \equiv [s_{\tau} \simeq t_{\tau}]^{\alpha}$ is called *atomic* if and only if it is (1) equational or (2) non-equational and the head symbol of s_{τ} is not $\beta\eta$-equivalent to a (primitive or defined) logical connective. A clause $\mathcal{C}$ is in *clause normal form* (CNF) if and only if all literals in $\mathcal{C}$ are atomic. Clauses that are not in CNF are also referred to as *pre-clauses*. It is well-known that every formula s_o can be transformed into an equisatisfiable set of clauses in CNF (cf. further below).

A substitution σ is a function mapping variables X_τ for terms of the same type. Application of a substitution σ to a term s_τ is defined as usual. Let $\ell \equiv [s \simeq t]^\alpha$ be a literal and σ a substitution, then the application of σ onto ℓ, written $\ell\sigma$, is given by the component-wise application $\ell\sigma \equiv [s\sigma \simeq t\sigma]^\alpha$. Analogously, for a clause $\mathcal{C} \equiv \ell_1 \vee \ldots \vee \ell_n$ and a substitution σ, $\mathcal{C}\sigma$ is given by $\mathcal{C}\sigma \equiv \ell_1\sigma \vee \ldots \vee \ell_n\sigma$. Positions π are defined as usual, the set of all positions of a term s is denoted $\text{pos}(s)$, and the subterm of a term s at position π is denoted $s|_\pi$. $s[t]_\pi$ denotes the term that is equivalent to s except that its subterm at position π is replaced by t.

A *calculus* is a set $R \equiv \{r_i\}$ of inference rules of the form

$$r_i \equiv \quad \frac{\mathcal{C}_1 \qquad \cdots \qquad \mathcal{C}_n}{\mathcal{C}}\ (R_i)$$

where $\mathcal{C}_i$ are clauses and (R_i) is the name of the rule. The $\mathcal{C}_i$ are called premises and $\mathcal{C}$ is the conclusion of (R_i). If an inference rule has multiple conclusions, they are simply written below each other. In the following, it is always assumed that all premises are variable disjoint. Given a calculus R, a set of clauses C and a clause $\mathcal{C}$, $\mathcal{C}$ is said to be *derivable from C by R* (or simply *derivable*), denoted $C \vdash_R \mathcal{C}$, if there exists a sequence $C_0 C_1 \ldots C_n$ of clause sets such that $C_0 \equiv C$, $\mathcal{C} \in C_n$ and for each $0 < i < n$ it holds that there exists a rule $r \in R$ such that $C_{i+1} \equiv C_i \cup D$ where D is a set of clauses that are the conclusion of r on some premises from C_i. The subscript R in $\vdash_R$ may be dropped if R is clear from the context. Concrete rule names may also be used as subscript to $\vdash$ to indicate (ambiguously) that a derivation exists that uses the given inference rule(s).

3.2. Extensional Paramodulation Calculus EP

The calculus EP lifts (unordered) first-order paramodulation to extensional higher-order logic. As mentioned above, unification cannot be used as a filtering mechanism. Following earlier work of Benzmüller et al., the inference rules of EP will encode unification constraints into the resulting clause that are then eligible to appropriate unification inferences. Similar to the $\mathcal{ERUE}$ system of Benzmüller et al. [Ben99b], EP allows eager solving of unification constraints such that they are not necessarily postponed until end of proof search and then solved in a dedicated routine (as proposed by Henkin).

The inference rules of EP can be grouped into four classes of inferences which are introduced individually in the following:

$\mathcal{CNF}$: Clausification rules for transforming pre-clauses into clausal normal form,

Figure 1: Clause normalization rules $\mathcal{CNF}$.

$\boxed{\text{CLAUSIFICATION RULES } \mathcal{CNF}}$

$$\frac{\mathcal{C} \vee [(l_\tau = r_\tau) \simeq \top]^{\alpha}}{\mathcal{C} \vee [l_\tau \simeq r_\tau]^{\alpha}}\ \text{(LiftEq)} \qquad \frac{\mathcal{C} \vee [\neg s_o]^{\alpha}}{\mathcal{C} \vee [s_o]^{\overline{\alpha}}}\ \text{(CNFNeg)}$$

$$\frac{\mathcal{C} \vee [s_o \vee t_o]^{\mathtt{tt}}}{\mathcal{C} \vee [s_o]^{\mathtt{tt}} \vee [t_o]^{\mathtt{tt}}}\ \text{(CNFDisj)} \qquad \frac{\mathcal{C} \vee [s_o \vee t_o]^{\mathtt{ff}}}{\begin{array}{c}\mathcal{C} \vee [s_o]^{\mathtt{ff}} \\ \mathcal{C} \vee [t_o]^{\mathtt{ff}}\end{array}}\ \text{(CNFConj)}$$

$$\frac{\mathcal{C} \vee [\forall X_\tau . s_o]^{\mathtt{tt}}}{\mathcal{C} \vee [s_o[Z/X]]^{\mathtt{tt}}}\ \text{(CNFAll)}^{\dagger} \qquad \frac{\mathcal{C} \vee [\forall X_\tau . s_o]^{\mathtt{ff}}}{\mathcal{C} \vee [s_o[\mathrm{sk}\ \overline{\mathrm{fv}(\mathcal{C})}/X]]^{\mathtt{ff}}}\ \text{(CNFExists)}^{\ddagger}$$

†: where Z_τ is a fresh variable for $\mathcal{C}$ ‡: where sk is a new Skolem constant of appropriate type

$\mathcal{PI}$:	Primary inference rules including paramodulation and primitive substitution,
$\mathcal{EXT}$:	Extensionality rules for Boolean and functional extensionality principles, and
$\mathcal{UNI}$:	Unification rules for eagerly solving unification constraints.

Given a set of formulas Φ, every formula $s_o \in \Phi$ is initially transformed into corresponding pre-clauses that are then admissible for clausification. Pre-clausification is used several times in the remainder of this chapter and formally given by the following definition.

Definition 3.1 (Pre-clausification)**.**
Let Φ be a set of sentences. Then, the *pre-clausification of* Φ, denoted Φ_{cl}, is given by $\Phi_{\mathrm{cl}} := \{[s_o{\downarrow_{\beta\eta}} \simeq \top]^{\mathtt{tt}} \mid s_o \in \Phi\}$. Pre-clausification essentially converts formulas to their corresponding pre-clauses without any further normalization or clausification steps. ⌟

Pre-clausification is evidently validity preserving as it does not change the semantics of the initial formulas. In general, the pre-clausified formulas will not be in clause normal form (CNF) and hence need to be converted to proper clauses as analogously done in resolution-based calculi.

Transformation to equisatisfiable clauses in clause normal form is realized using the clausification rules $\mathcal{CNF}$ displayed in Fig. 1. By convention, free variables of clauses are implicitly universally quantified. Existential quantifiers are eliminated from clauses via Skolemization by rule (CNFExists). Note that Skolemization in HOL requires special attention as a naive adaption of first-order Skolemization, as Andrews first pointed out in 1973, is unsound for HOL without choice and incomplete for HOL with choice. However, Skolemization can be adapted such that it is sound with respect to HOL semantics (Henkin semantics and standard semantics) without choice: Every introduced Skolem constant is not allowed to be used as proper term symbol without being fully applied to all possible arguments [Mil83, Mil91b]. For the remainder of the chapter, this sound Skolemization variant due to Miller is assumed. In particular, the clausification rules are sound with respect to Henkin semantics as assured by Lemma 3.2.

Lemma 3.2 (Soundness of $\mathcal{CNF}$)**.**
The clausification inference rules

$$\mathcal{CNF} := \{\text{(LiftEq)}, \text{(CNFNeg)}, \text{(CNFDisj)}, \text{(CNFConj)}, \text{(CNFAll)}, \\ \text{(CNFExists)}\}$$

preserve satisfiability with respect to Henkin models. ⌟

Proof. It is well-known that all rules of $\mathcal{CNF}$ except for (CNFExists) preserve validity and hence also satisfiability. (CNFExists) applies Skolemization which is satisfiability preserving in the version presented by Miller [Mil83, Mil91b]. □

The primary inferences $\mathcal{PI}$ of EP are paramodulation, factorization and primitive substitution as given in Fig. 2. The first two inference rules also exist in the context of first-order logic but differ significantly from their first-order counterparts. This is due to the fact that unifiability of the relevant terms is not used as a pre-condition to the rule but rather asserted by encoding the respective unification constraint as negative equality literal into the conclusion of the inference rule. These unification constraints are then subject to the unification inference rules (cf. further below) and can be eliminated if a unifier is found. The latter rule, primitive substitution, is required for completeness in the context of HOL as observed by Andrews [And89] and Huet [Hue72].

The paramodulation rule (Para) replaces subterms of literals within clauses by (potentially) equal terms given from a positive equality literal. Since the latter clause might not be a unit clause, the rewriting step can be considered *conditional* where the remaining literals represent the respective additional conditions. In contrast to previous work by Benzmüller et al, the calculus EP does not include expansion of equality by its definition and rather interprets equality as a primitive logical constant. Of course, (Para) will generate a large number of inference as

Figure 2: Primary inference rules of EP.

$\boxed{\text{PRIMARY INFERENCE RULES } \mathcal{PI}}$

$$\frac{\mathcal{C} \lor [s_\tau \simeq t_\tau]^\alpha \qquad \mathcal{D} \lor [l_\nu \simeq r_\nu]^{\mathtt{tt}}}{[s[r]_\pi \simeq t]^\alpha \lor \mathcal{C} \lor \mathcal{D} \lor [s|_\pi \simeq l]^{\mathtt{ff}}}\ (\text{Para})^\dagger$$

$$\frac{\mathcal{C} \lor [s_\tau \simeq t_\tau]^\alpha \lor [u_\tau \simeq v_\tau]^\alpha}{\mathcal{C} \lor [s_\tau \simeq t_\tau]^\alpha \lor [s_\tau \simeq u_\tau]^{\mathtt{ff}} \lor [t_\tau \simeq v_\tau]^{\mathtt{ff}}}\ (\text{Fac})$$

$$\frac{\mathcal{C} \lor [H_\tau\ \overline{s^i_{\tau^i}}]^\alpha \qquad G \in \mathcal{GB}_\tau^{\{\neg,\lor\}\cup\{\Pi^\nu,=^\nu \mid \nu\in\mathcal{T}\}}}{\mathcal{C} \lor [H_\tau\ \overline{s^i_{\tau^i}}]^\alpha \lor [H \simeq G]^{\mathtt{ff}}}\ (\text{Prim})$$

†: if $s|_\pi$ is of type ν and $\text{fv}(s|_\pi) \subseteq \text{fv}(s)$

potentially every (type-correct) combination of equality literals with subterms of the other clause yield a dedicated inference result. In practice, Leo-III tackles this problem by heuristic inference restrictions that try to imitate the ideal of first-order superposition (cf. §4.3.3).

Factorization (Fac) contracts two literals that are semantically equivalent but not syntactically equal, reducing the size of the clause if eager unification is successful on the generated unification constraints.

Primitive substitution (Prim) approximates the logical structure of literals with flexible heads. In contrast to early attempts that blindly guessed a concrete instance of head variables, (Prim) uses so-called *general bindings* [And89]:

Definition 3.3 (General Bindings [SG89])**.**
A *general binding* of type $\tau\overline{\tau^n}$ for head $h_{\tau\overline{\nu^m}}$, also referred to as *partial binding* or *approximating binding* in the literature, is a term g_τ of the form

$$g_\tau \equiv \lambda \overline{X^n_{\tau^i}}.h\ (H^1\ \overline{X^n})\ \ldots\ (H^m\ \overline{X^n})$$

where $\tau \in \mathcal{T}$, $\tau^i \in \mathcal{T}$ for $1 \leq i \leq n$, $\nu^j \in \mathcal{T}$ for $1 \leq j \leq m$, and the H^i are fresh variables of appropriate types. A general bindings is called *imitation binding* if h is a constant from the signature Σ, and a *projection binding* if h is a bound variable $X^i_{\tau^i}$.

For a set of constants H, the set of all imitation bindings of type τ and head $h \in H$ and projection bindings of type τ is denoted $\mathcal{GB}^H_\tau$. ⌟

General bindings step-wise approximate the instantiated term structure and hence limit the explosive growth of primitive substitution. Nevertheless, (Prim) still in a way enumerates the whole universe of terms but, in practical applications, often few applications of the rule are sufficient to find a refutation. An example where primitive substitution is necessary is the following: Consider a clause $\mathcal{C}$ given by $\mathcal{C} \equiv [P_o]^{\mathtt{tt}}$ where P is a Boolean variable. This clause corresponds to the formula $\forall P_o.P$ which is clearly not a theorem. Neither (Para) nor (Fac) or any other calculus rules presented so far or further below (except for (Prim)) allow to construct a derivation to the empty clause. However, using (Prim), there exists a derivation $[P]^{\mathtt{tt}} \vdash_{\text{(Prim),(Bind)}} [\neg P']^{\mathtt{tt}} \vdash_{\text{CNFNeg}} [P']^{\mathtt{ff}}$. Note that $\{\neg P'/P\}$ is a substitution applying a general binding from $\mathcal{GB}_o^{\{\neg,\vee\}\cup\{\Pi^\tau,=^\tau \mid \tau\in\mathcal{T}\}}$ that approximates logical negation. Now, a simple derivation involving $[P]^{\mathtt{tt}}$ and $[P']^{\mathtt{ff}}$ using (Para) and (Triv) yields the empty clause.

Soundness of the primary inference rules is asserted by Lemma 3.4:

Lemma 3.4 (Soundness of $\mathcal{PI}$)**.**
The primary inference rules $\mathcal{PI} := \{\text{(Para)}, \text{(Fac)}, \text{(Prim)}\}$ preserve satisfiability with respect to Henkin models. ⌟

Proof. The assertion follows from a case distinction:

(Para): If the freshly created unification constraint is not satisfiable, then the conclusion is trivially valid in every model. If, on the other hand, the unification constraint is valid in the current model, then the replacement of the subterm is valid by substitutivity of equality (if the respective literals from the premises were valid). All other cases are trivial.

(Fac): Analogous to (Para).

(Prim): Trivial, as only a disjunct is added to the premise. □

In the extensional setting of HOL, the calculus needs to guarantee completeness with respect to the extensionality principles. Earlier attempts simply added the corresponding extensionality axioms to the problem formulation in order to guarantee completeness even though the underlying calculus does not provide special extensionality rules. As adding extensionality principles to the search space enables cut-simulation [BBK09] and tremendously increases the number of inferences that can be generated, this seems infeasible in practice. The here presented calculus instead follows the work of Benzmüller et al. and includes inference rules for both positive and negative instances of Boolean and functional extensionality, respectively (cf. Fig. 3).[2]

[2] Note that ”extensionality principles” usually only refer to the validity of $(f_{\nu\tau}\, X_\tau = g_{\nu\tau}\, X_\tau) \Rightarrow$

Figure 3: Extensionality rules of EP.

EXTENSIONALITY RULES $\mathcal{EXT}$

$$\frac{\mathcal{C} \vee [s_o \simeq t_o]^{\mathtt{tt}}}{\begin{array}{c}\mathcal{C} \vee [s_o]^{\mathtt{tt}} \vee [t_o]^{\mathtt{ff}} \\ \mathcal{C} \vee [s_o]^{\mathtt{ff}} \vee [t_o]^{\mathtt{tt}}\end{array}}\ \text{(PBE)} \qquad \frac{\mathcal{C} \vee [s_o \simeq t_o]^{\mathtt{ff}}}{\begin{array}{c}\mathcal{C} \vee [s_o]^{\mathtt{tt}} \vee [t_o]^{\mathtt{tt}} \\ \mathcal{C} \vee [s_o]^{\mathtt{ff}} \vee [t_o]^{\mathtt{ff}}\end{array}}\ \text{(NBE)}$$

$$\frac{\mathcal{C} \vee [s_{\nu\tau} \simeq t_{\nu\tau}]^{\mathtt{tt}}}{\mathcal{C} \vee [s\, X_\tau \simeq t\, X_\tau]^{\mathtt{tt}}}\ \text{(PFE)}^{\dagger} \qquad \frac{\mathcal{C} \vee [s_{\nu\tau} \simeq t_{\nu\tau}]^{\mathtt{ff}}}{\mathcal{C} \vee [s\, \mathrm{sk}_\tau \simeq t\, \mathrm{sk}_\tau]^{\mathtt{ff}}}\ \text{(NFE)}^{\ddagger}$$

†: where X_τ is fresh for $\mathcal{C}$ ‡: where sk_τ is a Skolem term

Soundness of the extensionality rules is asserted by Lemma 3.5.

Lemma 3.5 (Soundness of $\mathcal{EXT}$).
The extensionality inference rules $\mathcal{EXT} := \{\text{(PBE)}, \text{(NBE)}, \text{(PFE)}, \text{(NFE)}\}$ preserve satisfiability with respect to Henkin models. ⌟

Proof. Let $\mathcal{M}$ be a Henkin model and g a variable assignment. The proof makes use of the fact that the Boolean and functional extensionality principles are valid in all Henkin models.

(PBE): Suppose $\mathcal{M} \models s_o = t_o$ for some formulas s_o and t_o. By definition it follows that $\|s = t\|^{\mathcal{M},g} \equiv$ T. This is equivalent to requiring that $\|s\|^{\mathcal{M},g} \equiv \|t\|^{\mathcal{M},g}$. Since $\mathcal{D}_o$ only contains two elements, this is equivalent to either (1) $\|s\|^{\mathcal{M},g} \equiv$ T and $\|t\|^{\mathcal{M},g} \equiv$ T or (2) $\|s\|^{\mathcal{M},g} \equiv$ F and $\|t\|^{\mathcal{M},g} \equiv$ F. In case (1) $\mathcal{M} \models s \vee \neg t$ since $\|s\|^{\mathcal{M},g} \equiv$ T and $\mathcal{M} \models \neg s \vee t$ since $\|t\|^{\mathcal{M},g} \equiv$ T, implying the assertion. Case (2) is analogous and follows from the fact that $\|\neg s\|^{\mathcal{M},g} \equiv$ T whenever $\|s\|^{\mathcal{M},g} \equiv$ F.

(NBE): Analogous to (PBE).

(PFE): Suppose $\mathcal{M} \models f_{\nu\tau} = g_{\nu\tau}$ for some functions f and g. It follows that $\|f\|^{\mathcal{M},g} \equiv \|g\|^{\mathcal{M},g}$ and by congruence that $\|f\|^{\mathcal{M},g}(x) \equiv \|g\|^{\mathcal{M},g}(x)$ for every $x \in \mathcal{D}_\tau$. Hence it holds that $\mathcal{M} \models \forall X_\tau.\, f\, X = g\, X$ which implies the assertion.

$f_{\nu\tau} = g_{\nu\tau}$ (functional extensionality) or of $(p_o \Leftrightarrow q_o) \Rightarrow p_o = q_o$ (Boolean extensionality) or their contrapositives as given by rule (NFE) and (NBE), respectively. Technically, the positive variants are more correctly referred to as congruence rules or simply applications of substitutivity of equality. However, for an uniform presentation of the rules, both variants are referred to by the term ”extensionality”.

Figure 4: Unification rules of EP

UNIFICATION RULES $\mathcal{UNI}$

$$\frac{\mathcal{C} \vee [s_\tau \simeq s_\tau]^{\mathit{ff}}}{\mathcal{C}}\ (\text{Triv}) \qquad \frac{\mathcal{C} \vee [X_\tau \simeq s_\tau]^{\mathit{ff}}}{\mathcal{C}\{s/X\}}\ (\text{Bind})^{\dagger}$$

$$\frac{\mathcal{C} \vee [c\,\overline{s^i} \simeq c\,\overline{t^i}]^{\mathit{ff}}}{\mathcal{C} \vee [s^1 \simeq t^1]^{\mathit{ff}} \vee \ldots \vee [s^n \simeq t^n]^{\mathit{ff}}}\ (\text{Decomp})$$

$$\frac{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq c_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathit{ff}} \qquad g_{\nu\overline{\mu}} \in \mathcal{GB}^{\{c\}}_{\nu\overline{\mu}}}{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq c_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathit{ff}} \vee [X \simeq g]^{\mathit{ff}}}\ (\text{FlexRigid})$$

$$\frac{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq Y_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathit{ff}} \qquad g_{\nu\overline{\mu}} \in \mathcal{GB}^{\{h\}}_{\nu\overline{\mu}}}{\mathcal{C} \vee [X_{\nu\overline{\mu}}\ \overline{s^i} \simeq Y_{\nu\overline{\tau}}\ \overline{t^j}]^{\mathit{ff}} \vee [X \simeq g]^{\mathit{ff}}}\ (\text{FlexFlex})^{\ddagger}$$

†: where $X_\tau \notin \text{fv}(s)$ ‡: where $h \in \Sigma$ is an appropriate constant

(NFE): Suppose $\mathcal{M} \models f_{\nu\tau} \neq g_{\nu\tau}$ for some functions f and g. It follows that $\|f\|^{\mathcal{M},g} \not\equiv \|g\|^{\mathcal{M},g}$ and hence there exists some object $x \in \mathcal{D}_\tau$ such that $\|f\|^{\mathcal{M},g}(x) \not\equiv \|g\|^{\mathcal{M},g}(x)$. Consequently it holds that $\mathcal{M} \models \exists X_\tau .\, f\,X \neq g\,X$. Application of (satisfiability preserving) Skolemization yields the desired result. $\square$

As mentioned earlier, higher-order unification is neither decidable nor unitary. To circumvent this problem unification rules are integrated into the calculus EP and unification tasks from generating inferences are encoded as negative equality literals (also referred to as *unification constraints*). A clause $\mathcal{C} \equiv \mathcal{D} \vee [s^1 \simeq t^1]^{\mathit{ff}} \vee \cdots \vee [s^n \simeq t^n]^{\mathit{ff}}$ can be regarded as a conditional clause $\mathcal{C} \equiv \big([s^1 \simeq t^1]^{\mathit{tt}} \wedge \cdots \wedge [s^n \simeq t^n]^{\mathit{tt}}\big) \to \mathcal{D}$. As a consequence, the negative equality literals denote unification constraints that represent the conditions under which the remaining clause $\mathcal{D}$ is valid. Unification constraints can be solved by iterative transformation using the unification rules from Fig. 4. A unification constraint $\ell \equiv [X_\tau \simeq s_\tau]^{\mathit{ff}}$ is *solved* if and only if $X \notin \text{fv}(s)$. Solved unification constraints can be propagated to the remaining clause using the rule (Bind). Note that equality resolution as known from first-order theorem proving is subsumed by (Bind). The (FlexRigid) approximates the opposite site using either projection or imitation bindings. The here presented unification rules are a variant of Huet's unification procedure [Hue75] which is known to be sound and complete with respect to higher-order unification [Sny91, SG89].

The unification rules can be used to eagerly solve unification constraints of freshly generated clauses. As unification constraints are in practice often solvable using a depth-bounded unifier search, eager unification can avoid or at least delay a search space explosion. Note that the here discussed calculus includes the infinitely branching (FlexFlex) rule. This is primarily motivated by the fact that the completeness proof of EP (in particular, the lifting lemma) is simpler when (FlexFlex) is available. However, this rule effectively enables blind guessing within a proof search and should be avoided whenever possible. There is strong evidence that (FlexFlex) is in fact admissible in EP as no proof so far faced by the author ever made use of that rule. This claim is further substantiated by similar results for tableaux-based calculi [Bro13]. This is also why the implementation of EP within the Leo-III system omits this rule. A formal proof for the admissibility of (FlexFlex) remains further work.

Soundness of the pre-unification rules wrt. Henkin semantics is asserted by Lemma 3.6.

Lemma 3.6 (Soundness of $\mathcal{UNI}$)**.**
The unification inference rules

$$\mathcal{UNI} := \big\{(\text{Triv}), (\text{Bind}), (\text{Decomp}), (\text{FlexRigid}), (\text{FlexFlex})\big\}$$

preserve satisfiability with respect to Henkin models. ⌟

Proof. Let $\mathcal{M}$ be a Henkin model and g a variable assignment. The assertion follows from a case distinction:

(Triv): Suppose $\mathcal{M} \models s_\tau \neq s_\tau$ for some term s. Evidently by reflexivity of equality $\|s \neq s\|^{\mathcal{M},g} \equiv \text{F}$ for any variable assignment g.

(Bind): Suppose $\mathcal{M} \models \forall X_\tau.\, p_o \vee X_\tau \neq s_\tau$ for some formula p_o and a term s such that $X \notin \text{fv}(s)$. By definition it holds that $\|\forall X_\tau.\, p_o \vee X_\tau \neq s_\tau\|^{\mathcal{M},g} \equiv \text{T}$ if and only if $\|p_o\|^{\mathcal{M},g[a/X]} \equiv \text{T}$ or $\|X_\tau \neq s_\tau\|^{\mathcal{M},g[a/X]} \equiv \text{T}$ for all $a \in \mathcal{D}_\tau$. Choosing $a_0 := \|s\|^{\mathcal{M},g}$ can always be done since $X \notin \text{fv}(s)$. Consequently it holds that $\|p_o\|^{\mathcal{M},g[a_0/X]} \equiv \text{T}$ or $\|X_\tau \neq s_\tau\|^{\mathcal{M},g[a_0/X]} \equiv \text{T}$ but by the above choice of a_0 it holds that
$\|X_\tau \neq s_\tau\|^{\mathcal{M},g[a_0/X]} \equiv \left(\|s\|^{\mathcal{M},g[a_0/X]} \not\equiv \|s\|^{\mathcal{M},g[a_0/X]}\right) \equiv \text{F}$. Hence, it holds that $\|p_o\|^{\mathcal{M},g[a_0/X]}$ which implies the assertion.

(Decomp): Suppose $\mathcal{M} \models f\ \overline{s^n} \neq f\ \overline{t^n}$ for some function $f_{\nu\tau_1\cdots\tau_n}$ and terms $s^i_{\tau_i}, t^i_{\tau_i}$. By definition $\|f\ \overline{s^n} = f\ \overline{t^n}\|^{\mathcal{M},g} \equiv \mathrm{F}$ and hence $\|f\ \overline{s^n}\|^{\mathcal{M},g} \not\equiv \|f\ \overline{t^n}\|^{\mathcal{M},g}$. It follows that there exists an i, $1 \leq i \leq n$, such that $\|s^i\|^{\mathcal{M},g} \not\equiv \|t^i\|^{\mathcal{M},g}$ and thus $\mathcal{M} \models s^i \neq t^i$ for some $1 \leq i \leq n$. In particular it then follows that $\mathcal{M} \models (s^1 \neq t^1) \vee \cdots \vee (s^n \neq t^n)$.

(FlexRigid), (FlexFlex): Trivial, since only a disjunct is added. $\square$

Finally, the calculus EP is presented as a combination of all above presented calculus rules:

Definition 3.7 (Higher-order Paramodulation)**.**
The *higher-order extensional paramodulation calculus EP* is given by

$$EP := \mathcal{CNF} \cup \mathcal{PI} \cup \mathcal{EXT} \cup \mathcal{UNI}$$

i.e. by all calculus rules given in Fig. 1 – 4. A set of formulas Φ is *refutable in EP* if and only if there exists a derivation $\Phi_{\mathrm{cl}} \vdash_{\mathrm{EP}} \square$ where Φ_{cl} is the pre-clausification of Φ. ⌟

Using the considerations from above, the argument for the soundness of EP is straight-forward and asserted by Theorem 3.8:

Theorem 3.8 (Soundness of EP)**.**
The calculus EP is sound wrt. Henkin semantics. More formally, for any set of formulas Φ it holds that if $\Phi \vdash \square$ then Φ is Henkin-unsatisfiable. ⌟

Proof. All inference rules of EP are satisfiability preserving (some even validity preserving) by Lemmas 3.2, 3.4, 3.5 and 3.6. Hence, if a set of sentences Φ is Henkin-satisfiable, it holds that $\Phi \nvdash \square$ which implies the assertion. $\square$

3.3. Completeness

In this section, completeness of EP is verified. To that end, a *model existence theorem* is presented that adapts and simplifies earlier work of Benzmüller et al. in the context of equational higher-order logic with Henkin semantics. Model existence results itself are developed independently from EP using the usual multi-step approach: First, syntactical properties are presented that constitute so-called abstract consistency classes Γ_Σ. Given a set of formulas Φ contained in such an abstract consistency class Γ_Σ, Hintikka sets $\mathcal{H}$ extending Φ are then constructed as maximal elements of Γ_Σ. As a last step, a concrete Henkin model $\mathcal{M}$ is constructed from $\mathcal{H}$ such that in particular $\mathcal{M} \models \Phi$.

The completeness proof of EP then employs the model existence theorem by verifying that the set of formulas that cannot be refuted by EP forms an abstract consistency class.

3.3.1. Abstract Consistency Classes

The key idea of abstract consistency properties is that the verification of completeness properties of a given calculus R can be reduced to checking whether certain proof-theoretic properties are satisfied (i.e. that the set of irrefutable sentences in R constitutes an abstract consistency class), instead of analyzing model-theoretic properties which are usually more difficult and tedious to assess. Abstract consistency conditions for such an abstraction were first studied by Smullyan in the context of first-order logic [Smu63, Smu95, Hin55] (referred to as *unifying principles* in that context), adapted to elementary type theory by Andrews [And71, And02a] and further developed for extensional type theory by Kohlhase, Benzmüller and Brown [Koh94, Ben99a, BBK04a].

Definition 3.9 (Sufficient Σ-purity)**.**
Let Σ be a signature and $\Phi \subseteq \Lambda_o(\Sigma)$ and set of Σ-sentences. Φ is called *sufficiently* Σ*-pure* if and only if for each type $\tau \in \mathcal{T}$ there exists a set $P \subseteq \Sigma_\tau$ of parameters with equal cardinality to $\Lambda_\tau(\Sigma)$ such that no element of P occurs in any sentence from Φ. ⌟

For technical reasons, in the remainder of this section every set of sentences is assumed to be sufficiently Σ-pure as defined by Def. 3.9. This assumption makes sure that there are always enough witnessing parameters available in a given signature Σ. Sufficient Σ-purity can be obtained by enriching Σ with spurious constants [Ben99a].

Definition 3.10 (Closed under subsets, Compactness)**.**
Let $\mathcal{S}$ be a class of sets. $\mathcal{S}$ is called

(1) *closed under subsets* if and only if for all sets S, T it holds that if $S \subseteq T$ and $T \in \mathcal{S}$ then $S \in \mathcal{S}$, and
(2) *compact* if and only if for all sets S it holds that $S \in \mathcal{S}$ if and only if every finite subset of S is a member of $\mathcal{S}$. ⌟

The technical notions of subset closure and compactness as defined in Def. 3.10 are key properties of abstract consistency classes in the following. It can simply be verified that every compact class of sets is also closed under subsets. By convention, let $\Phi * s$ denote the set $\Phi \cup \{s\}$ where Φ is some set of formulas and s is a formula. Also, multiple applications of $*$ are assumed to associate to the left, e.g. $\Phi * s * t$ represents the expression $(\Phi * s) * t$ and, consequently, the set $\Phi \cup \{s, t\}$.

Then, abstract consistency classes are given by the following definition:

Definition 3.11 (Abstract consistency classes)**.**
Let Σ be a signature and Γ_Σ a class of sets of Σ-sentences. Γ_Σ is called an *abstract consistency class* if it is closed under subsets and the following properties hold:

∇_c:	If s is atomic, then $s \notin \Phi$ or $\neg s \notin \Phi$.
$\nabla_{\beta\eta}$:	If $s \equiv_{\beta\eta} t$ and $s \in \Phi$, then $\Phi * t \in \Gamma_\Sigma$.
$\nabla_=^r$:	$(s \neq s) \notin \Phi$.
$\nabla_=^s$:	If $u[s]_p \in \Phi$ and $s = t \in \Phi$ then $\Phi * u[t]_p \in \Gamma_\Sigma$.
$\nabla_\mathfrak{b}^+$:	If $s = t \in \Phi$, then $\Phi * s * t \in \Gamma_\Sigma$ or $\Phi * \neg s * \neg t \in \Gamma_\Sigma$.
$\nabla_\mathfrak{b}^-$:	If $s \neq t \in \Phi$, then $\Phi * s * \neg t \in \Gamma_\Sigma$ or $\Phi * \neg s * t \in \Gamma_\Sigma$.
$\nabla_\mathfrak{f}^+$:	If $f_{\nu\tau} = g_{\nu\tau} \in \Phi$, then $\Phi * f\ s = g\ s \in \Gamma_\Sigma$ for any closed term $s \in \Lambda^c(\Sigma)$.
$\nabla_\mathfrak{f}^-$:	If $f_{\nu\tau} \neq g_{\nu\tau} \in \Phi$, then $\Phi * f\ w \neq g\ w \in \Gamma_\Sigma$ for some parameter $w \in \Sigma_\tau$ that does not occur in any sentence of Φ.
∇_m:	If s, t are atomic and $s, \neg t \in \Phi$, then $\Phi * s \neq t \in \Gamma_\Sigma$.
∇_d:	If $h\ \overline{s^n} \neq h\ \overline{t^n} \in \Phi$, then there is an i with $1 \leq i \leq n$ such that $\Phi * s^i \neq t^i \in \Gamma_\Sigma$.

The collection of all abstract consistency classes is denoted $\mathfrak{Acc}$. $\lrcorner$

Note that the notion of abstract consistency classes presented here differs from the one presented by Benzmüller et al. [Ben99a, BBK04b, BBK04a, BBK09] in the following ways:

- Since the completeness proof is only required for Henkin models, the class $\mathfrak{Acc}$ implicitly corresponds to the functionally and Boolean extensional abstract consistency classes $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$ as defined by Benzmüller et al.
- In the work of Benzmüller et al. the existence of identity relations in each domain $\mathcal{D}_{o\tau\tau}$ (called property $\mathfrak{q}$) is assumed for every class of model structures [BBK04a]. Due to the choice of primitive connectives of the here presented HOL logic, this requirement is automatically met.
- As equality is a primitive (versus defined) logical connective, the definition of abstract consistency classes presented here does not include con-

sistency conditions for extensionality that make use of Leibniz equality $\doteq$. This implies that negative as well as positive conditions are required for functional and Boolean extensionality properties as the positive part cannot be inferred by expansion of Leibniz equality.

- Equality is assumed to be the only logical connective in this work. This is why there are no consistency conditions for the usual connectives such as disjunction or universal quantification. These properties can however be inferred from the definition given here (cf. Lemma 3.13).
- The class of abstract consistency classes $\mathfrak{Acc}$ natively includes the conditions ∇_m and ∇_d. These properties were studied by Benzmüller et al. in the context of *acceptable* classes for $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ [BBK04b, BBK09]. As a consequence, the completeness proof in the remainder of this section does not assume *saturated* abstract consistency classes. Saturation is related to cut-elimination and thus difficult to prove, while ∇_m and ∇_d are usually easier to verify in practice [BBK09].

Some technically convenient properties are established by Lemma 3.12 that are used to shorten later proofs:

Lemma 3.12.
Let Γ_Σ be an abstract consistency class and let $\Phi \in \Gamma_\Sigma$. It holds that

(1) $\bot \notin \Phi$
(2) $\neg\top \notin \Phi$
(3) $\bot = \top \notin \Phi$
(4) If $s = \top \in \Phi$ (or $s \neq \bot \in \Phi$) then $\Phi * s \in \Gamma_\Sigma$
(5) If $s = \bot \in \Phi$ (or $s \neq \top \in \Phi$) then $\Phi * \neg s \in \Gamma_\Sigma$. ⌟

Proof. Let Γ_Σ be an abstract consistency class and let $\Phi \in \Gamma_\Sigma$.

(1) Assume $\bot \in \Phi$. By definition of $\bot$ it then holds that $\lambda P_o.\top = \lambda P_o.P \in \Phi$. By $\nabla_{\mathfrak{f}}^{+}$ and $\nabla_{\beta\eta}$ it then holds that $\Phi * \top = u \in \Gamma_\Sigma$ for any closed $u_o \in \Lambda_o^c(\Sigma)$. In particular taking $u \equiv \neg\top$ produces $\Phi * \top = \neg\top \in \Gamma_\Sigma$. Applying $\nabla_{\mathfrak{b}}^{+}$ then contradicts to ∇_c, hence $\bot \notin \Phi$.

(2) Suppose $\neg\top \in \Phi$. By definition it holds that $=_{ooo} \neq =_{ooo} \in \Phi$ which contradicts $\nabla_{=}^{r}$. Hence, $\neg\top \notin \Phi$.

(3) Suppose $\bot = \top \in \Phi$. By $\nabla_{\mathfrak{b}}^{+}$ it then holds that either $\Phi * \bot * \top \in \Gamma_\Sigma$ or $\Phi * \neg\bot * \neg\top \in \Gamma_\Sigma$. Since both cases contradict (1) and (2), respectively, it follows that $\bot = \top \notin \Phi$.

(4) Suppose $s = \top \in \Phi$, then by $\nabla_{\mathfrak{b}}^{+}$ it holds that either $\Phi * s * \top \in \Gamma_\Sigma$ or $\Phi * \neg s * \neg\top \in \Gamma_\Sigma$. By (2) the latter case is impossible hence $\Phi * s * \top \in$

Γ_Σ and, by subset closure, $\Phi * s \in \Gamma_\Sigma$ as desired. The argument for $s \neq \bot$ is analogous.

(5) Suppose $s = \bot \in \Phi$. Then $\Phi * \neg s \in \Gamma_\Sigma$ by definition of $\neg$ and $\nabla_{\beta\eta}$. If $s \neq \top \in \Phi$, then by $\nabla_{\mathfrak{b}}^{-}$ either $\Phi * s * \neg\top \in \Gamma_\Sigma$ or $\Phi * \neg s * \top \in \Gamma_\Sigma$. By (2) the first case is contradictory, so $\Phi * \neg s * \top \in \Gamma_\Sigma$ and by subset closure $\Phi * \neg s \in \Gamma_\Sigma$.

□

The properties of the usual remaining connectives are respected by abstract consistency classes:

Lemma 3.13 (Abstract consistency properties of usual connectives)**.**
Let Γ_Σ be an abstract consistency class and let $\Phi \in \Gamma_\Sigma$. It holds that

$\nabla_\neg$: If $\neg\neg s \in \Phi$, then $\Phi * s \in \Gamma_\Sigma$
$\nabla_\vee$: If $s \vee t \in \Phi$, then $\Phi * s \in \Gamma_\Sigma$ or $\Phi * t \in \Gamma_\Sigma$
$\nabla_\wedge$: If $(s \wedge t) \in \Phi$, then $\Phi * s * t \in \Gamma_\Sigma$
$\nabla_\forall$: If $\Pi F \in \Phi$, then $\Phi * F\, w \in \Gamma_\Sigma$ for every $w \in \Lambda^c(\Sigma)$.
$\nabla_\exists$: If $\neg(\Pi F) \in \Phi$, then $\Phi * \neg(F\, c) \in \Gamma_\Sigma$ for any parameter $c \in \Sigma$.

⌟

Proof. Let Γ_Σ be an abstract consistency class and let $\Phi \in \Gamma_\Sigma$.

$\nabla_\neg$: Suppose $\neg\neg s \in \Phi$, then by definition of $\neg$ and $\nabla_{\beta\eta}$ it holds that $\Phi * (s \neq \bot) \in \Gamma_\Sigma$. Then, by $\nabla_{\mathfrak{b}}^{-}$ and subset closure either (a) $\Phi * s * \neg\bot \in \Gamma_\Sigma$ or (b) $\Phi * \neg s * \bot \in \Gamma_\Sigma$. Since the latter case contradicts ∇_c (as $\neg\neg s \in \Phi$), it holds that $\Phi * s * \neg\bot \in \Gamma_\Sigma$ and thus by subset closure $\Phi * s \in \Gamma_\Sigma$.

$\nabla_\vee$: Suppose $s \vee t \in \Phi$, then by definition of $\vee$ and $\nabla_{\beta\eta}$ it holds that $\Phi * \lambda g.g \top \top \neq \lambda g.g\, s\, t \in \Gamma_\Sigma$. Then, by $\nabla_{\mathfrak{f}}^{-}$, $\nabla_{\beta\eta}$ and subset closure it holds that $\Phi * w \top \top \neq w\, s\, t \in \Gamma_\Sigma$ for some parameter $w \in \Sigma$ not occurring in any sentence of Φ and by ∇_d, Lemma 3.12 and subset closure either $\Phi * s \in \Gamma_\Sigma$ or $\Phi * t \in \Gamma_\Sigma$ as desired.

$\nabla_\wedge$: Suppose $(s \wedge t) \in \Phi$, then by definition of $\wedge$ and $\nabla_{\beta\eta}$ it holds that $\Phi * \lambda g.g \top \top = \lambda g.g\, s\, t \in \Gamma_\Sigma$ and thus by $\nabla_{\mathfrak{f}}^{+}$ and $\nabla_{\beta\eta}$ it holds that $\Phi * (\lambda g.g \top \top = \lambda g.g\, s\, t) * (u \top \top = u\, s\, t) \in \Gamma_\Sigma$ for any closed term $u \in \Lambda_{ooo}^c(\Sigma)$. Taking $u \equiv \lambda X,Y.X = Y$ gives $\Phi * (\lambda g.g \top \top = \lambda g.g\, s\, t) * s = t \in \Gamma_\Sigma$ by $\nabla_{\mathfrak{b}}^{+}$, $\nabla_=^r$ and subset closure. Again, by $\nabla_{\mathfrak{b}}^{+}$ we either have $\Phi * (\lambda g.g \top \top = \lambda g.g\, s\, t) * s * t \in \Gamma_\Sigma$ or $\Phi * (\lambda g.g \top \top = \lambda g.g\, s\, t) * \neg s * \neg t \in \Gamma_\Sigma$. The latter case implies $\Phi * (\lambda g.g \top \top = \lambda g.g\, s\, t) * s = \bot * t = \bot \in \Gamma_\Sigma$ by definition of $\neg$, $\nabla_{\beta\eta}$ and subset closure. Now by $\nabla_=^s$

and subset closure it follows that $\Phi * \lambda g.\, g \top \top = \lambda g.\, g \perp \perp \in \Gamma_\Sigma$. But this contradicts Lemma 3.12 (3) by $\nabla_{\mathfrak{f}}^{+}$ and $\nabla_{\beta\eta}$ (e.g. take $g \equiv \lambda X, Y. X$). Hence, it follows that $\Phi * s * t \in \Gamma_\Sigma$.

$\nabla_\forall$: Suppose $\Pi F \in \Phi$. then by definition of Π and $\nabla_{\beta\eta}$ it holds that $\Phi * (F = \lambda X. \top) \in \Gamma_\Sigma$ and by $\nabla_{\mathfrak{f}}^{+}$, $\nabla_{\beta\eta}$ and subset closure also $\Phi * F\ w = \top \in \Gamma_\Sigma$ for every closed term $w \in \Lambda^c(\Sigma)$. From Lemma 3.12 (4) and subset closure it then follows that $\Phi * F\ w \in \Gamma_\Sigma$.

$\nabla_\exists$: Suppose $\neg(\Pi F) \in \Phi$. then by definition of Π, $\neg$ and $\nabla_{\beta\eta}$ it holds that $\Phi * (F \neq \lambda X. \top) \in \Gamma_\Sigma$ and by $\nabla_{\mathfrak{f}}^{-}$, $\nabla_{\beta\eta}$ and subset closure also $\Phi * F\ w \neq \top \in \Gamma_\Sigma$ for any parameter $w \in \Sigma$ not occurring in any sentence of Φ. From Lemma 3.12 (5) and subset closure it then follows that $\Phi * \neg(F\ w) \in \Gamma_\Sigma$. □

In the construction of models further below, it will be necessary to assume that abstract consistency classes are compact. As Lemma 3.14 shows, abstract consistency classes Γ_Σ can be assumed to be compact without loss of generality.

Lemma 3.14 (Compactness of abstract consistency classes)**.**
Let Γ_Σ be an abstract consistency class. Then there exists another abstract consistency class Γ'_Σ such that Γ'_Σ is compact and $\Gamma_\Sigma \subseteq \Gamma'_\Sigma$. ⌟

Proof. Let Γ_Σ be an abstract consistency class and let Γ'_Σ be defined by

$$\Gamma'_\Sigma := \left\{\Phi \subseteq \Lambda^c_o(\Sigma) \mid \text{every finite subset of } \Phi \text{ is an element of } \Gamma_\Sigma\right\}$$

Γ'_Σ is compact (hence closed under subsets) and it holds that $\Gamma_\Sigma \subseteq \Gamma'_\Sigma$ (Γ_Σ is closed under subsets and hence for any $\Phi \in \Gamma_\Sigma$, every finite subset of Φ is in Γ_Σ). It remains to be shown that all $\nabla_\star$ hold for Γ'_Σ. Let for that purpose $\Phi \in \Gamma'_\Sigma$ in the following. The proofs follow earlier work by Benzmüller et al. [BBK04b, Lemma 4.3] [BBK04a, Lemma 6.18].

∇_c: Suppose there is a $s \in \Lambda^c_o(\Sigma)$ such that $\{s, \neg s\} \subseteq \Phi$. Since $\{s, \neg s\}$ is a finite subset of Φ it holds that $\{s, \neg s\} \in \Gamma_\Sigma$, contradicting ∇_c for Γ_Σ.

$\nabla_{\beta\eta}$: Let $s \equiv_{\beta\eta} t$ and $s \in \Phi$ for some $s, t \in \Lambda^c(\Sigma)$. Let further be Ψ a finite subset of $\Phi * t$ and $\Theta := (\Psi \setminus \{t\}) * s$. Evidently, Θ is a finite subset of Φ and thus $\Theta \in \Gamma_\Sigma$. By $\nabla_{\beta\eta}$ for Γ_Σ it also holds that $\Theta * t \in \Gamma_\Sigma$. Since Γ_Σ is closed under subsets and $\Psi \subseteq \Theta * s$ it holds that $\Psi \in \Gamma_\Sigma$. Then, by definition of Γ'_Σ, $\Phi * t \in \Gamma'_\Sigma$.

$\nabla^r_=$: Analogous to ∇_c.

$\nabla^s_=$: Analogous to $\nabla_{\beta\eta}$.

$\nabla_{\mathfrak{b}}^+$: Let $s_o = t_o \in \Phi$ and suppose that $\Phi * s * t \notin \Gamma'_\Sigma$ and $\Phi * \neg s * \neg t \notin \Gamma'_\Sigma$. Then there exist finite subsets Ψ_1, Ψ_2 of Φ such that $\Psi_1 * s * t \notin \Gamma_\Sigma$ and $\Psi_2 * \neg s * \neg t \notin \Gamma_\Sigma$. $\Psi := \Psi_1 \cup \Psi_2 * (s = t)$ is evidently a finite subset of Φ and thus $\Psi \in \Gamma_\Sigma$. By $\nabla_{\mathfrak{b}}^+$ for Γ_Σ it holds that either $\Psi * s * t \in \Gamma_\Sigma$ or $\Psi * \neg s * \neg t \in \Gamma_\Sigma$ and, by subset closure, that $\Psi_1 * s * t \in \Gamma_\Sigma$ or $\Psi_2 * \neg s * \neg t \in \Gamma_\Sigma$. Since this contradicts the choice of Ψ_1 and Ψ_2, it follows that $\Phi * s * t \in \Gamma'_\Sigma$ or $\Phi * \neg s * \neg t \in \Gamma'_\Sigma$.

$\nabla_{\mathfrak{b}}^-$: Analogous to $\nabla_{\mathfrak{b}}^+$.

$\nabla_{\mathfrak{f}}^+, \nabla_{\mathfrak{f}}^-$, ∇_m: Analogous to $\nabla_{\beta\eta}$.

∇_d: For $1 \leq i \leq n$, let each $s^i_{\tau^i}, t^i_{\tau^i} \in \Lambda^c(\Sigma)$, $\tau_i \in \mathcal{T}$, be closed terms and $h_{\nu\tau_n\cdots\tau_1} \in \Sigma$ a parameter. Let further $(h\,\overline{s^i} \neq h\,\overline{t^i}) \in \Phi$. Suppose $\Phi * (s^i \neq t^i) \notin \Gamma'_\Sigma$ for all $1 \leq i \leq n$. Then, there exist finite subsets Ψ_i of Φ such that $\Psi_i * (s^i \neq t^i) \notin \Gamma_\Sigma$ for all $1 \leq i \leq n$. Now let $\Psi := \left(\bigcup_{1 \leq i \leq n} \Psi_i\right) * (h\,\overline{s^i} \neq h\,\overline{t^i})$. Ψ is a finite subset of Φ, hence $\Psi \in \Gamma_\Sigma$. By ∇_d for Γ_Σ it holds that there is an i such that $\Psi * (s^i \neq t^i) \in \Gamma_\Sigma$. As $\Psi_i \subseteq \Psi$ and Γ_Σ is closed under subsets it follows that $\Psi_i * (s^i \neq t^i) \in \Gamma_\Sigma$. Since this contradicts the fact that $\Psi_i * (s^i \neq t^i) \notin \Gamma_\Sigma$ for all $1 \leq i \leq n$, it follows that $\Phi * (s^i \neq t^i) \in \Gamma'_\Sigma$ for some $1 \leq i \leq n$.

□

3.3.2. Hintikka Sets

Hintikka sets represent the most important tool for the subsequent model existence theorem as they allow computations similar to those of HOL models. Hintikka sets will be constructed from maximal elements of abstract consistency classes and allow the construction of concrete valuation structures that are then turned into a corresponding Henkin model.

Definition 3.15 (Hintikka set)**.**

Let $\mathcal{H}$ be a set of Σ-sentences. $\mathcal{H}$ is called an *Hintikka* set if the following properties hold:

$\vec{\nabla}_c$:	$s \notin \mathcal{H}$ or $\neg s \notin \mathcal{H}$.
$\vec{\nabla}_{\beta\eta}$:	If $s \equiv_{\beta\eta} t$ and $s \in \mathcal{H}$, then $t \in \mathcal{H}$.
$\vec{\nabla}^r_=$:	$(s \neq s) \notin \mathcal{H}$.
$\vec{\nabla}^s_=$:	If $u[s]_p \in \mathcal{H}$ and $s = t \in \mathcal{H}$ then $u[t]_p \in \mathcal{H}$.
$\vec{\nabla}^+_\mathfrak{b}$:	If $s = t \in \mathcal{H}$, then $\{s,t\} \subseteq \mathcal{H}$ or $\{\neg s, \neg t\} \subseteq \mathcal{H}$.
$\vec{\nabla}^-_\mathfrak{b}$:	If $s \neq t \in \mathcal{H}$, then $\{s, \neg t\} \subseteq \mathcal{H}$ or $\{\neg s, t\} \subseteq \mathcal{H}$.
$\vec{\nabla}^+_\mathfrak{f}$:	If $f_{\nu\tau} = g_{\nu\tau} \in \mathcal{H}$, then $f\ s = g\ s \in \mathcal{H}$ for any closed term $s \in \Lambda^c(\Sigma)$.
$\vec{\nabla}^-_\mathfrak{f}$:	If $f_{\nu\tau} \neq g_{\nu\tau} \in \mathcal{H}$, then $f\ w \neq g\ w \in \mathcal{H}$ for some parameter $w \in \Sigma_\tau$.
$\vec{\nabla}_m$:	If s,t are atomic and $s, \neg t \in \mathcal{H}$, then $s \neq t \in \mathcal{H}$.
$\vec{\nabla}_d$:	If $h\ \overline{s^n} \neq h\ \overline{t^n} \in \mathcal{H}$, then there is an i with $1 \leq i \leq n$ such that $s^i \neq t^i \in \mathcal{H}$.

The collection of all Hintikka sets is denoted $\mathfrak{H}$. ⌟

As the following lemma states, subset-maximal elements of abstract consistency classes that satisfy a certain condition are in fact Hintikka sets. A set $\Phi \in \Gamma_\Sigma$ is called *subset-maximal* if for every $s_o \in \Lambda^c_o(\Sigma)$ such that $\Phi * s \in \Gamma_\Sigma$ it already holds that $s \in \Phi$.

Lemma 3.16 (Hintikka lemma)**.**
Let Γ_Σ be an abstract consistency class and let $\mathcal{H} \in \Gamma_\Sigma$. If $\mathcal{H}$ is subset-maximal in Γ_Σ and satisfies $\vec{\nabla}^-_\mathfrak{f}$ then $\mathcal{H}$ is a Hintikka set. ⌟

Proof. Let Γ_Σ be an abstract consistency class and let $\mathcal{H} \in \Gamma_\Sigma$ as above. It needs to be shown that all $\vec{\nabla}_*$ properties hold for $\mathcal{H}$:

$\vec{\nabla}_c$: Follows from the fact that $\mathcal{H} \in \Gamma_\Sigma$ satisfies ∇_c.
$\vec{\nabla}_{\beta\eta}$: Suppose $s \equiv_{\beta\eta} t$ and $s \in \mathcal{H}$. Since $\mathcal{H} \in \Gamma_\Sigma$ and $\nabla_{\beta\eta}$ it holds that $\mathcal{H} * t \in \Gamma_\Sigma$. Since $\mathcal{H}$ is subset-maximal by assumption, this implies that $t \in \mathcal{H}$.
$\vec{\nabla}^r_=$: Follows from the fact that $\mathcal{H} \in \Gamma_\Sigma$ satisfies $\nabla^r_=$.
$\vec{\nabla}^s_=$: Analogous to $\vec{\nabla}_{\beta\eta}$.
$\vec{\nabla}^+_\mathfrak{b}$: Suppose $s = t \in \mathcal{H}$. Since $\mathcal{H} \in \Gamma_\Sigma$ and $\nabla^+_\mathfrak{b}$ it holds that $\mathcal{H} * s * t \in \Gamma_\Sigma$ or $\mathcal{H} * \neg s * \neg t \in \Gamma_\Sigma$. Since $\mathcal{H}$ is subset-maximal by assumption, this implies that either $\{s,t\} \subseteq \mathcal{H}$ or $\{\neg s, \neg t\} \subseteq \mathcal{H}$.
$\vec{\nabla}^-_\mathfrak{b}$: Analogous to $\vec{\nabla}^+_\mathfrak{b}$.
$\vec{\nabla}^+_\mathfrak{f}$: Analogous to $\vec{\nabla}_{\beta\eta}$.
$\vec{\nabla}^-_\mathfrak{f}$: By assumption.
$\vec{\nabla}_m, \vec{\nabla}_d$: Analogous to $\vec{\nabla}_{\beta\eta}$.

□

Lemma 3.17 now confirms that any compact abstract consistency class can be extended to a Hintikka set. In the proof of this lemma, a transfinite induction over a sequence of all closed Σ-sentences in conducted. Recall that it is assumed that $\mathcal{V}$ is countably infinite and Σ_τ is infinite and of cardinality $\aleph_s$ for each type $\tau \in \mathcal{T}$. The proof idea is adopted from earlier work of Benzmüller et al. [BBK04a] which in turn follows Henkin [Hen50].

Lemma 3.17 (Abstract extension lemma)**.**
Let Γ_Σ be a compact abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be sufficiently Σ-pure. Then there exists a Hintikka set $\mathcal{H}$ such that $\Phi \subseteq \mathcal{H}$. ⌟

Proof. The proof of this lemma follows the construction by Benzmüller et al. [BBK04a, Lemma 6.32], where a transfinite sequence $\left(\mathcal{H}^\delta\right)_{\delta<\varepsilon}$ is constructed such that $\mathcal{H}^\delta \in \Gamma_\Sigma$ for every $\delta < \varepsilon$ and the resulting Hintikka set is given by $\mathcal{H} := \bigcup_{\delta<\varepsilon} \mathcal{H}^\delta$. To that end, the set $\Lambda_o^c(\Sigma)$ is enumerated as $\left(A^\delta\right)_{\delta<\varepsilon}$ and successively inserted into $\mathcal{H}^\delta$ if consistent with Γ_Σ, together with witnesses if A^δ denotes a negative equality between functions. Note that the sequence $\left(A^\delta\right)_{\delta<\varepsilon}$ exists by the well-ordering principle and the fact that $\Lambda_o^c(\Sigma)$ has cardinality of at most $\aleph_s$, where ε is the first ordinal of this cardinality.

The construction and the proof is identical to the one by Benzmüller et al. with the following exceptions:

(1) Only one witnessing sequence X^δ is defined by $X^\delta := (s\, w_\tau^\delta) \neq (t\, w_\tau^\delta)$ if A^δ is of the form $A^\delta \equiv s_{\nu\tau} \neq t_{\nu\tau}$ for some types $\tau, \nu \in \mathcal{T}$. Otherwise, $X^\delta := A^\delta$. Note that this subsumes witnesses for existential quantification by definition of Π in terms of equality.

(2) Consequently, the transfinite sequence $\left(\mathcal{H}^\delta\right)_{\delta<\varepsilon}$ is modified to let $\mathcal{H}^{\delta+1} := \mathcal{H}^\delta * A^\delta * X^\delta$ if $\mathcal{H}^\delta * A^\delta \in \Gamma_\Sigma$ and $\mathcal{H}^{\delta+1} := \mathcal{H}^\delta$ otherwise in the successor case. For limit ordinals δ, $\mathcal{H}^\delta$ is defined as usual by $\mathcal{H}^\delta := \bigcup_{\sigma<\delta} \mathcal{H}^\sigma$ and $\mathcal{H}^0 := \Phi$.

(3) The argument that $\mathcal{H}^{\delta+1} \in \Gamma_\Sigma$ whenever $\mathcal{H}^\delta \in \Gamma_\Sigma$ is simplified as follows: If $\mathcal{H}^\delta * A^\delta \notin \Gamma_\Sigma$ then this is trivially true since because $\mathcal{H}^{\delta+1} \equiv \mathcal{H}^\delta \in \Gamma_\Sigma$. If, on the other hand, $\mathcal{H}^\delta * A^\delta \in \Gamma_\Sigma$, there are two sub-cases:
    (i) If $X^\delta \equiv A^\delta$ then evidently $\mathcal{H}^\delta * A^\delta * X^\delta \in \Gamma_\Sigma$.
    (ii) If $X^\delta \not\equiv A^\delta$, then by $\nabla_{\mathfrak{f}}^-$ it holds that $\mathcal{H}^\delta * A^\delta * X^\delta \in \Gamma_\Sigma$ since w_τ^δ does not occur in any sentence of $\mathcal{H}^\delta * A^\delta$.

The remaining arguments remain unchanged, in particular $\mathcal{H}$ is subset-maximal and satisfies $\vec{\nabla}_{\mathfrak{f}}^-$ by (3) and hence $\mathcal{H} \in \mathfrak{H}$ by Lemma 3.16. □

3.3.3. Model Existence

Using the abstract consistency classes and Hintikka sets from the previous section appropriate HOL models can be constructed, yielding a so-called *model existence theorem* that states that for every set of formulas that is abstractly consistent there exists a HOL model that satisfies this set. While the main result asserts the existence of appropriate Henkin models (cf. Theorem 3.20), the result is constructed using an indirection over more general notions of HOL model structures. The model structures used by Theorem 3.18 and Theorem 3.19 are given by the class $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ of extensional Σ-models with primitive equation as introduced by Benzmüller et al. [BBK04a, §3.3]. These model structures allow an abstraction from HOL models over frames as introduced in §2.2 to a more general notion based on *applicative structures*. For brevity, a thorough and verbose introduction of these model structures is omitted here, but can be found in the relevant literature [BBK04a, BBK04b].

Theorem 3.18 now states that a generalized notion of HOL model structure exists for a given Hintikka set $\mathcal{H}$ whereas Theorem 3.19 connects this result to abstract consistency classes.

Theorem 3.18 (Model existence theorem for Hintikka sets)**.**
Let $\mathcal{H}$ be a Hintikka set. There exists a model structure $\mathcal{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ such that $\mathcal{M} \models \mathcal{H}$. ⌟

Proof. Analogous to the construction of Benzmüller et al [BBK04b, Theorem 8.12]. The only exception is that (weak-)compatibility is defined in terms of primitive equality instead of Leibniz equality and the choice of logical connectives. The definition of Σ-per evaluations and consequently the model constructed can be simplified to only show the appropriate denotation of primitive equality. However, to avoid exhaustive repetition of analogous constructions (e.g. all occurrences of Leibniz equality are replaced by primitive equality) and the fact that the usual logical connectives are given their standard denotation (cf. Lemma 2.5), the proof construction of Benzmüller et al. can be used. □

Since abstract consistency classes can be extended to Hintikka sets, every abstractly consistent set of formulas also has a general HOL model structure.

Theorem 3.19 (Model existence theorem)**.**
Let Γ_Σ be an abstract consistency class and let $\Phi \subseteq \Gamma_\Sigma$ be a sufficiently Σ-pure set of sentences. Then there exists a model structure $\mathcal{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ such that $\mathcal{M} \models \Phi$. ⌟

Proof. Suppose Γ_Σ is an abstract consistency class and let $\Phi \subseteq \Gamma_\Sigma$ be sufficiently Σ-pure. By Lemma 3.14 there exists a compact $\Gamma'_\Sigma \in \mathfrak{Acc}$ such that $\Gamma_\Sigma \subseteq \Gamma'_\Sigma$. By Lemma 3.17 there exists a Hintikka set $\mathcal{H} \in \Gamma'_\Sigma$ such that $\Phi \subseteq \mathcal{H}$. Finally, Theorem 3.18 guarantees the existence of a model structure $\mathcal{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ such that $\mathcal{M} \models \mathcal{H}$ and hence $\mathcal{M} \models \Phi$. □

The main result of this section is finally given by Theorem 3.20. This result effective utilizes the fact that for any generalized model structure $\mathcal{M} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ there exists an isomorphic Henkin model that validates the same sentences.

Theorem 3.20 (Henkin model existence theorem)**.**
Let Γ_Σ be a abstract consistency class and let $\Phi \subseteq \Gamma_\Sigma$ be a sufficiently Σ-pure set of sentences. Then there exists a Henkin model $\mathcal{M}$ such that $\mathcal{M} \models \Phi$. ⌟

Proof. By Theorem 3.19 there exists a model structure $\mathcal{M}^{\text{pre}} \in \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}$ such that $\mathcal{M}^{\text{pre}} \models \Phi$. It is known that for every model from $\mathcal{M}_{\beta\mathfrak{f}\mathfrak{b}}$ there exists an isomorphic Henkin model $\mathcal{M}$ [BBK04a, Theorem 3.68], in particular $\mathcal{M} \models \Phi$. □

3.3.4. Completeness of EP

In this section, completeness of EP with respect to Henkin semantics is verified. To that end, it will be shown that the set of irrefutable sentences in EP constitutes an abstract consistency class and hence always has a model. This validates completeness of EP in its refutational setting.

The most complex technical result for the completeness of EP is the *lifting lemma*. This lemma states that for any refutation of a set of instantiated clauses $C\sigma$ there exists an analogous abstract refutation of the more general set of clauses C. An even more general result is given by the following Lemma. The desired result then follows directly.

Lemma 3.21 (General Lifting lemma)**.**
Let C be a set of clauses, $\mathcal{D}_1$ a clause and σ a substitution. If $C\sigma \vdash \mathcal{D}_1$ then there exists a derivation $C \vdash \mathcal{D}_2\xi$ for a clause $\mathcal{D}_2$ and a substitution ξ such that $\mathcal{D}_2\xi \equiv \mathcal{D}_1$.

Proof. The proof is by induction about the length of the derivation $\Delta : C\sigma \vdash \mathcal{D}_1$. The base case, i.e. a derivation of length zero, is trivial (take $\mathcal{D}_2 := \mathcal{D}_1$ and $\xi := \sigma$). For the induction step, assume that Δ applies some inference rule $r \in$ EP for the first derivation step. The assertion then follows from a case distinction over the rules of EP and the induction hypotheses. Let $\mathcal{C}\sigma \in C\sigma$ be a clause in the following.

(LiftEq): Suppose $\mathcal{C}\sigma \equiv \ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee [s\sigma = t\sigma \simeq \top]^\alpha$ and $\mathcal{D}_1 \equiv \ell_1\sigma \vee \ldots \vee \ell_n\sigma \vee [s\sigma \simeq t\sigma]^\alpha$ for some literals ℓ_i, $1 \leq i \leq n$, and terms s,t of equal type. The uninstantiated clause $\mathcal{C} \equiv \ell_1 \vee \ldots \ell_n \vee [X \simeq Y]^\alpha$ is now examined for its possible structure with X,Y being two meta variables (note that $\simeq$ is not a logical symbol from Σ and can thus not be instantiated by any substitution σ). The following cases are distinguished:
(1) If $X \equiv s = t$ and $Y \equiv \top$, then the assertion follows trivially (take $\xi := \sigma$).
(2) If $Y \equiv \top$ there exists a variable H and terms u^i, $1 \leq i \leq m$ of appropriate types with $X \equiv H\,\overline{u^i}$ such that $(H\,\overline{u^i})\sigma \equiv (s\sigma = t\sigma)$. For $H\,\overline{u^i}$ it holds that either (i) $H\sigma \equiv \lambda\overline{X^i}.l = r$ for some terms l,r such that $l\{\overline{u^i}/\overline{X^i}\}\sigma \equiv s\sigma$ and $r\{\overline{u^i}/\overline{X^i}\}\sigma \equiv t\sigma$, or (ii) $H\sigma \equiv \lambda\overline{X^i}.X^j\,u$, for some term u, such that $(X^j\,u)\{\overline{u^i}/\overline{X^i}\}\sigma \equiv s\sigma = t\sigma$.

Case (i): Rule (Prim) can be applied to $\mathcal{C}_1$ with general binding $\lambda\overline{X^m}.(H^1\,\overline{X^m}) = (H^2\,\overline{X^m})$ for predicate variable H, where H^1 and H^2 are fresh predicate variables of appropriate type. This yields a new clause $\mathcal{E}_1$ with

$$\mathcal{E}_1 \equiv \ell_1 \vee \ldots \vee \ell_n \vee [H\,\overline{u^i} \simeq \top]^\alpha \vee [H \simeq \lambda\overline{X^m}.(H^1\,\overline{X^m}) = (H^2\,\overline{X^m})]^{\mathrm{ff}}$$

and, after application of (Bind) to $\mathcal{E}_1$ another clause

$$\mathcal{E}_2 \equiv \ell_1\xi \vee \ldots \vee \ell_n\xi \vee [(H^1\,\overline{u^i}) = (H^2\,\overline{u^i}) \simeq \top]^\alpha$$

where $\xi \equiv \{\overline{X^m}.(H^1\,\overline{X^m}) = (H^2\,\overline{X^m})/H\}$. Now, (Triv) can be applied, yielding

$$\mathcal{E}_2 \equiv \ell_1\xi \vee \ldots \vee \ell_n\xi \vee [(H^1\,\overline{u^i}) \simeq (H^2\,\overline{u^i})]^\alpha$$

Note that ξ is more general than σ since it is an approximating binding using two flexible variables H^1 and H^2 and hence there exists another substitution γ that instantiates the variables H^1 and H^2 as desired and coincides with σ on all other variables, i.e. $\ell_i(\gamma \circ \xi) \equiv \ell_i\sigma$, $(H^1\,\overline{u^i})(\gamma \circ \xi) \equiv s\sigma$ and $(H^2\,\overline{u^i})(\gamma \circ \xi) \equiv t\sigma$. Consequently, $\mathcal{E}_2(\gamma \circ \xi) \equiv \mathcal{D}_1$.
Case (ii): This case is analogous by applying (Prim) to $\mathcal{C}_1$ with a projection binding $\lambda\overline{X^i}.X^j\,(H\,\overline{X^i})$.

(3) If $X \equiv s = t$ there exists a variable H and terms u^i, $1 \leq i \leq m$, of appropriate types with $Y \equiv H\,\overline{u^i}$ such that $(H\,\overline{u^i})\sigma \equiv \top$. For $H\,\overline{u^i}$ it holds that either (i) $H\sigma \equiv \lambda\overline{X^i}.\top$ or (ii) $H\sigma \equiv \lambda\overline{X^i}.X^j\,u$ for some term u such that $(X^j\,u)\{\overline{u^i/X^i}\}\sigma \equiv \top$. Now, (PBE) (if $\alpha \equiv \mathtt{tt}$) resp. (NBE) (if $\alpha \equiv \mathtt{ff}$) followed by (LiftEq) can be used to infer the clause

$$\mathcal{E}_1 \equiv \ell_1 \vee \ldots \vee \ell_n \vee [s \simeq t]^\alpha \vee [H\,\overline{u^i}]^{\mathtt{ff}}$$

Note that the flexible literal $[H\,\overline{u^i}]^{\mathtt{ff}}$ formally represents the literal $[H\,\overline{u^i} \simeq \top]^{\mathtt{ff}}$. Hence, unification rule (FlexRigid) can be applied to approximate the rigid constant $\top$.

Case (i): Using (FlexRigid) and (Bind), an imitation binding $\lambda\overline{X^i}.\top$ can be applied to clause $\mathcal{E}_1$, yielding a new clause

$$\mathcal{E}_2 \equiv \ell_1\xi \vee \ldots \vee \ell_n\xi \vee [s \simeq t]^\alpha\xi \vee [\top \simeq \top]^{\mathtt{ff}}$$

for substitution $\xi := \{\lambda\overline{X^i}.\top/H\}$ and, consequently

$$\mathcal{E}_3 \equiv \ell_1\xi \vee \ldots \vee \ell_n\xi \vee [s \simeq t]^\alpha\xi$$

by rule (Triv). Note that ξ is again at least as general as σ, hence there exists another substitution γ that coincides with σ on all other variables, i.e. $\ell_i(\gamma \circ \xi) \equiv \ell_i\sigma$. Consequently, $\mathcal{E}_3(\gamma \circ \xi) \equiv \mathcal{D}_1$.
Case (ii): This case is analogous by choosing a suitable projection binding to be instantiated by (FlexRigid) and (Bind).

(4) In the remaining case where $X \not\equiv s = t$ and $Y \not\equiv \top$ the previous two cases can be applied successively, yielding the desired result.

(CNFNeg), (CNFDisj), (CNFConj), (CNFAll), (CNFExists): The argument is analogous to the previous case.

(Para): Lifting is straight-forward if the paramodulation step occurs in a subterm that is also contained in the uninstantiated literal. In all other cases, the lifting argument of Benzmüller [Ben99a, Corollary 5.26] can be applied analogously.

(Prim): Suppose $\mathcal{C}\sigma \equiv \mathcal{C}'\sigma \vee [H\,\overline{s^i\sigma}]^\alpha$ and $\mathcal{D}_1 \equiv \mathcal{C}'\sigma \vee [H\,\overline{s^i\sigma}]^\alpha \vee [H \simeq P]^{\mathtt{ff}}$ for a variable H, terms s^i of appropriate type and a suitable general binding $P \in \mathcal{GB}^{\{\neg,\vee\}\cup\{\Pi^\nu,=^\nu \mid \nu\in\mathcal{T}\}}$ such that $\mathcal{C}\sigma \vdash_{\text{(Prim)}} \mathcal{D}_1$. An analogous inference of (Prim) can be applied to the uninstantiated clause $\mathcal{C} \equiv \mathcal{C}' \vee [H\,\overline{s^i}]^\alpha$ producing

$$\mathcal{E}_1 \equiv \mathcal{C}' \vee [H\,\overline{s^i}]^\alpha \vee [H \simeq P]^{\mathtt{ff}}$$

where $\mathcal{E}_1\sigma \equiv \mathcal{D}_1$.

(Fac), (PBE), (NBE), (PFE), (NFE): Analogous to the previous case.

(Bind): Suppose $\mathcal{C}\sigma \equiv \mathcal{C}'\sigma \vee [X \simeq s\sigma]^\alpha$ for a variable X and a term s of appropriate type. The uninstantiated clause is then of form $\mathcal{C} \equiv \mathcal{C}' \vee [Z \simeq s]^\alpha$ where Z is a meta variable for terms of appropriate type. If X has base type then Z is also a variable and the inference can analogously be simulated on the abstract level. If, on the other hand, X has functional type then Z might also be of the form $Z \equiv X\,t$ where X is a variable and t is some term. Then, a substitution that is identical or more general than σ can subsequently be approximated by (FlexRigid) and (FlexFlex) on the calculus level, yielding a clause $\mathcal{E}$ such that there exists a substitution τ with $\mathcal{E}\tau \equiv \mathcal{D}_1$ [Ben99a].

(Triv): Analogous to the previous case.

(Decomp): Suppose $\mathcal{C}\sigma \equiv \mathcal{C}'\sigma \vee [f\ \overline{s^i\sigma} \simeq f\ \overline{t^i\sigma}]^{\mathrm{ff}}$ for a term f and terms s^i, t^i, $1 \leq i \leq n$. If the uninstantiated clause already has term f as head symbol on both sides of the literal, the assertion trivially follows. Otherwise, there are two cases:

Case (i): If only one side of the abstract literal has f has head symbol, then the uninstantiated clause $\mathcal{C}$ is of the form $\mathcal{C} \equiv \mathcal{C}' \vee [H; \overline{a^i} \simeq f\ \overline{b^i}]^{\mathrm{ff}}$ such that $(H\ \overline{a^i})\sigma \equiv f\ \overline{s^i\sigma}$ and $b^i\sigma \equiv t^i\sigma$. In this case, rule (FlexRigid) followed by (Bind) generate the clause

$$\mathcal{E}_1 \equiv \mathcal{C}'\tau \vee [f\ (H^1\ \overline{a^i}) \ldots (H^k\ \overline{a^i}) \simeq f\ \overline{b^i}]\tau^{\mathrm{ff}}$$

where $\tau \equiv \{\lambda \overline{X^i}. f\ (H^1\ \overline{a^i}) \ldots (H^k\ \overline{a^i})/H\}$ and k is the arity of f. As this inference uses approximating bindings, the substitution τ is at least as general as σ (note the flexible heads H^j, $1 \leq j \leq k$). Hence, (Decomp) can now be applied yielding a clause $\mathcal{E}_2$ and there exists a substitution ξ such that $\mathcal{E}_2\xi \equiv \mathcal{C}_1$.

Case (ii): If no side of the abstract literal has f as head symbol, an application of (FlexFlex) can instantiate one occurrence as desired, followed by the same argument as in case (i).

(FlexRigid): Suppose $\mathcal{C}\sigma \equiv \mathcal{C}'\sigma \vee [H\ \overline{s^i\sigma} \simeq f\ \overline{t^j\sigma}]^{\mathrm{ff}}$ for a variable H, a term f and terms s^i, t^j, $1 \leq i \leq n$ and $1 \leq j \leq m$. If (FlexRigid) is already applicable on the abstract literal, then the assertional trivially follows. In all other cases, the uninstantiated clause $\mathcal{C}$ is of form $\mathcal{C} \equiv \mathcal{C}' \vee [H\ \overline{s^i} \simeq G\ \overline{a^k}]^{\mathrm{ff}}$ such that $G\ \overline{a^k}\sigma \equiv f\ \overline{t^j\sigma}$. Then, an application of (FlexFlex) approximates f for variable H, hence yielding a clause on which (FlexRigid) can again be applied. Since the substitution from the flex-flex step is at least as general as σ the assertion follows.

(FlexFlex): Analogous to the case of (Prim).

$\square$

Corollary 3.22.
Let C be a set of clauses and σ a substitution. If $C\sigma \vdash \square$ then $C \vdash \square$. $\square$

For the completeness proof, the notion of consistent sets of clauses will play an important role.

Definition 3.23 (EP-consistent)**.**
Let $\Phi \subseteq \Lambda_o^c(\Sigma)$ be a set of sentences. Φ is called *EP-inconsistent* if $\Phi \vdash \square$ and *EP-consistent* otherwise.

Finally, the proof that the set of irrefutable sentences of EP constitutes an abstract consistency property is given. The class of EP-consistent sets of formulas are those which cannot be refuted by EP.

Lemma 3.24.
Let $\Gamma_\Sigma := \{\Phi \subseteq \Lambda_o^c(\Sigma) \mid \Phi_{cl} \text{ is EP-consistent } \}$ be the class of all sets of Σ-sentences that cannot be refuted by EP. Then, Γ_Σ is an abstract consistency class.

Proof. Let Γ_Σ be as above. It suffices to show each ∇_* property (cf. Def 3.11). Let $\Phi \in \Gamma_\Sigma$ be a set of sentences in the following. In all cases except for $\nabla_{\beta\eta}$ each sentence $s_o \in \Phi$ is without loss of generality identified with its pre-clausified variant $[s]^{\mathtt{tt}} \in \Phi_{\text{cl}}$. However, technically for each sentence $s_o \in \Phi$ there exists a clause $[s']^{\mathtt{tt}} \in \Phi_{\text{cl}}$ by pre-clausification with $s' \equiv s{\downarrow_{\beta\eta}}$. Since $\longrightarrow_{\beta\eta}$ is known to be convergent (i.e. confluent and terminating), this does not affect the validity of the arguments.

∇_c: Let $s \in \Lambda_o^c(\Sigma)$ be atomic and suppose that $s, \neg s \in \Phi$. Then $\Phi_{\text{cl}} \equiv \Phi'_{\text{cl}} * [s]^{\mathtt{tt}} * [\neg s]^{\mathtt{tt}}$ for some Φ'_{cl}. Furthermore, since s is atomic, there is a derivation $\Phi'_{\text{cl}} * [s]^{\mathtt{tt}} * [\neg s]^{\mathtt{tt}} \vdash_{\text{(Neg)}} \Phi'_{\text{cl}} * [s]^{\mathtt{tt}} * [s]^{\mathtt{ff}} \vdash_{\text{(Para),(Triv)}} \square$. This contradicts the assumption that Φ is EP-consistent.

$\nabla_=^r$: Let $s \in \Lambda^c(\Sigma)$ and suppose that $(s \neq s) \in \Phi$. By initial pre-clausification $\Phi_{\text{cl}} \equiv \Psi_{\text{cl}} * [\neg(s = s)]^{\mathtt{tt}}$ for some set Ψ_{cl}. Then, there is a derivation $\Psi_{\text{cl}} * [\neg(s = s)]^{\mathtt{tt}} \vdash_{\text{(Neg),(LiftEq)}} \Psi_{\text{cl}} * [s \simeq s]^{\mathtt{ff}} \vdash_{\text{(Triv)}} \square$, again contradicting the assumptions.

The remaining cases are proved by showing their contrapositive:

$\nabla_{\beta\eta}$: This is trivially true since if $s_o, t_o \in \Phi$ with $s \equiv_{\beta\eta} t$ and $\Phi_{\text{cl}} \vdash \Box$ then by definition of pre-clausification there is a pre-clause $[u]^{\mathtt{tt}} \in \Phi_{\text{cl}}$ such that $s{\downarrow}_{\beta\eta} \equiv u \equiv t{\downarrow}_{\beta\eta}$ and $\Phi_{\text{cl}} \equiv \Phi'_{\text{cl}} * [u]^{\mathtt{tt}} \vdash \Box$ which is equivalent to $\Phi' * [s{\downarrow}_{\beta\eta}]^{\mathtt{tt}} \vdash \Box$.

$\nabla^{s}_{=}$: Suppose that $\Phi_{\text{cl}} * [u[s]]^{\mathtt{tt}} * [s = t]^{\mathtt{tt}} * [u[t]]^{\mathtt{tt}} \vdash \Box$. There is a derivation $\Phi_{\text{cl}} * [u[s]]^{\mathtt{tt}} * [s = t]^{\mathtt{tt}} \vdash_{\text{(LiftEq)}} \Phi_{\text{cl}} * [u[s]]^{\mathtt{tt}} * [s \simeq t]^{\mathtt{tt}} \vdash_{\text{(Para),(Triv)}} \Phi_{\text{cl}} * [u[s]]^{\mathtt{tt}} * [s = t]^{\mathtt{tt}} * [u[t]]^{\mathtt{tt}}$ and hence $\Phi_{\text{cl}} * [u[s]]^{\mathtt{tt}} * [s = t]^{\mathtt{tt}} \vdash \Box$.

$\nabla^{+}_{\mathfrak{b}}$: Suppose that $\Phi_{\text{cl}} * [s_o = t_o]^{\mathtt{tt}} * [s]^{\mathtt{tt}} * [t]^{\mathtt{tt}} \vdash \Box$ and $\Phi_{\text{cl}} * [s_o = t_o]^{\mathtt{tt}} * [\neg s]^{\mathtt{tt}} * [\neg t]^{\mathtt{tt}} \vdash \Box$. There exists a derivation $\Phi_{\text{cl}} * [s_o = t_o]^{\mathtt{tt}} \vdash_{\text{(LiftEq),(PBE)}} \Phi_{\text{cl}} * [s_o = t_o]^{\mathtt{tt}} * [s]^{\mathtt{tt}} \vee [t]^{\mathtt{ff}} * [s]^{\mathtt{ff}} \vee [t]^{\mathtt{tt}}$ from which there exists a derivation of $\Box$. The argument is similar to the one presented by Benzmüller [Ben99a, Lemma 4.15].

$\nabla^{-}_{\mathfrak{b}}$: Analogous to $\nabla^{+}_{\mathfrak{b}}$.

$\nabla^{+}_{\mathfrak{f}}$: Suppose that $\Phi_{\text{cl}} * [f_{\mu\tau} = g_{\mu\tau}]^{\mathtt{tt}} * [f\, s \neq g\, s]^{\mathtt{tt}} \vdash \Box$ for each closed formula s_τ. By the lifting lemma (Corollary 3.22), it holds that $\Phi_{\text{cl}} * [f_{\mu\tau} = g_{\mu\tau}]^{\mathtt{tt}} * [f\, X \neq g\, X]^{\mathtt{tt}} \vdash \Box$ for a fresh variable X_τ. Hence, it follows that $\Phi_{\text{cl}} * [f_{\mu\tau} = g_{\mu\tau}]^{\mathtt{tt}} \vdash \Box$ by a simple application of (PFE).

$\nabla^{-}_{\mathfrak{f}}$: Suppose that $\Delta : \Phi_{\text{cl}} * [s_{\mu\tau} \neq t_{\mu\tau}]^{\mathtt{tt}} * [s\, w \neq t\, w]^{\mathtt{tt}} \vdash \Box$ where w is a parameter that does not occur in any sentence in Φ_{cl}. There exists a derivation $\Phi_{\text{cl}} * [s_{\mu\tau} \neq t_{\mu\tau}]^{\mathtt{tt}} \vdash_{\text{(LiftEq),(NFE)}} \Phi_{\text{cl}} * [s_{\mu\tau} \simeq t_{\mu\tau}]^{\mathtt{ff}} * [s\, v \simeq t\, v]^{\mathtt{ff}}$ where v is a fresh parameter. Since both w and v do not occur in any sentence of Φ_{cl} the derivation, the desired derivation $\Phi_{\text{cl}} * [s_{\mu\tau} \simeq t_{\mu\tau}]^{\mathtt{ff}} * [s\, v \simeq t\, v]^{\mathtt{ff}} \vdash \Box$ can be constructed by renaming all occurrences of w in Δ by v.

∇_m: Suppose that $\Phi_{\text{cl}} * [s]^{\mathtt{tt}} * [\neg t]^{\mathtt{tt}} * [s \neq t]^{\mathtt{tt}} \vdash \Box$. There exists a derivation $\Phi_{\text{cl}} * [s]^{\mathtt{tt}} * [\neg t]^{\mathtt{tt}} \vdash_{\text{(Neg)}} \Phi_{\text{cl}} * [s]^{\mathtt{tt}} * [t]^{\mathtt{ff}} \vdash_{\text{(Para),(Triv)}} \Phi_{\text{cl}} * [s]^{\mathtt{tt}} * [t]^{\mathtt{ff}} * [s \simeq t]^{\mathtt{ff}}$ and hence a derivation of $\Box$.

∇_d: Suppose that $\Phi_{\text{cl}} * [h\, \overline{s^i} \neq h\, \overline{t^i}]^{\mathtt{tt}} * [s^i \neq t^i]^{\mathtt{tt}} \vdash \Box$ for some $1 \leq i \leq n$. There exists a derivation $\Phi_{\text{cl}} * [h\, \overline{s^i} \neq h\, \overline{t^i}]^{\mathtt{tt}} \vdash_{\text{(Neg),(LiftEq)}} \Phi_{\text{cl}} * [h\, \overline{s^i} \simeq h\, \overline{t^i}]^{\mathtt{ff}} \vdash_{\text{(Decomp)}} \Phi_{\text{cl}} * [h\, \overline{s^i} \simeq h\, \overline{t^i}]^{\mathtt{ff}} * [s^1 \simeq t^1]^{\mathtt{ff}} * \ldots * [s^n \simeq t^n]^{\mathtt{ff}}$ and hence a derivation of $\Box$.

□

Completeness of EP now follows quite simply:

Theorem 3.25 (Completeness of EP)**.**
EP is complete with respect to Henkin semantics, i.e. for every sufficiently Σ-pure set of sentences Φ and any sentence s_o it holds that $\Phi * \neg s \vdash \Box$ whenever $\Phi \models s$.
⌟

Proof. Let $\Phi \subset \Lambda^c_o(\Sigma)$ and $s_o \in \Lambda^c_o(\Sigma)$ as above such that $\Phi \models s$. Since $\Phi \models s$ it holds that s is valid in every Henkin model that satisfies Φ, hence $\Phi * \neg s$ in unsatisfiable in every Henkin model $\mathcal{M}$. Since only finitely many parameters occur in $\neg s$ it follows that $\Phi * \neg s$ is sufficiently Σ-pure. Furthermore by Lemma 3.24 the class $\Gamma_\Sigma := \{\Phi \subseteq \Lambda^c_o(\Sigma) \mid \Phi_{cl}$ is EP-consistent $\}$ constitutes an abstract consistency class and hence, by Theorem 3.20, every EP-consistent set of sentences has a Henkin model. Consequently, $\Phi * \neg s$ must be EP-inconsistent and hence $\Phi * \neg s \vdash \Box$. $\Box$

3.4. Ordering Constraints

In first-order theorem proving, plain (unordered) paramodulation approaches were quickly neglected in favor of ordered paramodulation and in particular superposition-based calculi [NR01]. This is due to the fact that the unordered paramodulation rule generates many inferences many of which irrelevant for the ultimate proof goal. In contrast, first-order proof calcali such as superposition pose restriction on the premises of the inference rule such that only a subset of all possible inferences are generated [BG94]. These restriction are based on a ordering relations derived from a term ordering $\succ$ and allow to consider only maximal (wrt. $\succ$ or extensions thereof) sides of maximal literals within a clause for generating inferences. More formally, in superposition $\succ$ is assumed to be a *reduction ordering* [BN98], that is a noetherian strict partial order on (first-order) terms such that it is total on groud terms (or, at least completable) and has the following properties:

- Stability under substitutions: If $s \succ t$, then $s\sigma \succ t\sigma$ for every substitution σ.
- Compatibility with term structure: If $s \succ t$, then $u[s]_p \succ u[t]_p$ for every term u.

Note that the above properties of an reduction ordering $\succ$ imply that for every term s and any strict subterm t of s it holds that $s \succ t$ [BN98], this is often referred to as *subterm property*.

Due to the more complex structure of higher-order logic terms there do not exist straight-forward adaptions of first-order reduction orderings to HOL. A simple counter-example is given by the following scenario: Assume there exists some constant $c_\tau \in \Sigma$, then there exists a cycle $c \equiv_\beta (\lambda X_\tau . c)\ c \succ c$ implying $c \succ c$ which contradicts irreflexity of a strict partial order. Note that the second step in the cycle is implied by the subterm property of $\succ$ since c is a proper subterm of $(\lambda X_\tau . c)\ c$.

In the context of program termination, there has been intensive theoretical and practical research on higher-order rewriting and respective term orderings [GMR$^+$15] and a wide field of different notions of higher-order rewriting have been developed [Kop12]. However, the ordering relations developed in this context lack the desired properties for utilization within a superposition-based calculus. It is an open but highly interesting research question whether meaningful ordered calculi can be constructed if usual properties of higher-order ordering relations, such as compatibility with β-reduction, are dropped.

Another approach is currently pursued in the context of the Matryoshka project [BFSW17] in which first-order reduction orderings and superposition-based calculi are generalized to a restricted fragment of higher-order logic devoid of λ-abstraction, referred to as *lambda-free HOL*.[3] Here, adaptions of a Knuth-Bendix-order [BBWW17] and recursive path orderings [BWW17] have been developed. Still, it is ongoing work to generalize such an approach to full HOL with Henkin semantics, in particular including λ-abstractions and Boolean extensionality.

In contrast to above, in this work a more pragmatic approach is chosen to increase the practical effectivity of the (unordered) calculus EP: In the implementation of EP, a higher-order term ordering originating from the termination community is used to (heuristically) restrict the inferences generated during proof search. The ordering of choice is hereby a variant of the *Computability Path Ordering* (CPO) [BJR15] which is a well-founded higher-order term ordering with a restricted notion of substitution stability and subterm compatibility. As this restriction does not guarantee completeness in general, the hope is that the restrictions on EP still perform sufficiently well in practice.

[3] Note that the formulation of "Lambda-free higher-order logic" by Bentkamp et al. does not come with any comprehension axioms and hence cannot be regarded as a higher-order logic formalism in the usual sense, cf. discussions on higher-order formalism in the literature [End15, VBD83]. Additionally, in their formulation there exist no Boolean subterms (and consequently no Boolean extensionality).

4. The Leo-III System

As outlined at the beginning of the thesis, the main goal of the PhD project was to develop an ATP system for classical HOL with Henkin semantics that addresses the shortcomings of resolution-based approaches when handling equality. To that end, the previously discussed paramodulation calculus EP has been implemented and augmented with practically motivated inference mechanisms. The resulting theorem proving system is called Leo-III, named after its predecessors LEO-II [BPST15] and LEO [BK98b].[1]

Leo-III is an automated theorem prover for classical higher-order logic with Henkin semantics and choice. It is implemented in Scala, open-source and available for usage and contribution under a BSD license at GitHub.[2] Instructions for installation and usage can be found in Appendix A.

Leo-III supports all common TPTP dialects [Sut17b], including CNF and FOF [Sut09], TFF [SSCB12] and THF [SB10]. Additionally, as one of the first higher-order ATP systems, Leo-III supports reasoning in rank-1 polymorphic HOL using the TF1 [BP13b] and TH1 [KSR16] language dialects. During the development of Leo-III, careful attention has been paid to providing maximal compatibility with existing systems and conventions of the peer community, especially to those of the TPTP infrastructure [Sut17b]. The prover returns results according to the standardized SZS ontology [Sut08] and additionally produces a TSTP-compatible (refutation) proof certificate [Sut07, Sut10], if a proof is found.

In the tradition of the cooperative nature of the LEO prover family, Leo-III collaborates during proof search with external reasoning systems, in particular, with first-order ATPs such as E [Sch02], iProver [Kor08] and Vampire [RV02] as well as SMT solvers like CVC4 [B$^+$11]. Again, this collaboration strongly benefits from the TPTP language standards as it makes use of the respective dialects for communicating with the external systems. Unlike LEO-II, which

[1] Initially designed as theorem prover component as part of a larger reasoning framework called Ωmega [SBA06, BFM$^+$06], the name LEO is an acronym for *Logical Engine for Omega*. As of the development of LEO-II, the provers were designed as stand-alone ATP systems for HOL without any employment restrictions related to Ωmega. Hence the original acronym's meaning does not apply anymore to the Leo prover family. That is why the official capitalization used for the most recent incarnation of the Leo provers, i.e. "Leo-III", was chosen to explicitly reduce the reference to its original meaning while still gracefully referring to its predecessor versions.

[2] The most current version of Leo-III can be retrieved at `https://github.com/leoprover/Leo-III/releases/latest`.

translated proof obligations into untyped first-order languages, Leo-III, by default, translates its higher-order clauses to (polymorphic or monomorphic) many-sorted first-order formulas. Leo-III accumulates higher-order clauses during proof search and repeatedly invokes all cooperating systems on the generated proof obligations. For that purpose, various translation mechanisms from HOL to different variants of first-order logic are implemented. If any external (first-order) reasoning system finds the submitted proof obligation to be unsatisfiable, the original HOL problem is unsatisfiable as well and a proof for the original conjecture is found.

While LEO-II is based on an augmented version of RUE resolution [DH86, BPST15] and thereby supports handling of primitive equality on the calculus level, its actual reasoning power concerning equality is still limited. Most of LEO-II's reasoning effectivity originates from external cooperation with E. This is also due to the fact that LEO-II does not include effective reasoning and simplification techniques from equational first-order theorem provers such as rewriting and (ordered) paramodulation. The cooperation principles of LEO-II are adapted, improved and combined with a restricted paramodulation calculus in Leo-III. Furthermore, major effort has been invested to adapt simplification and search control techniques from first-order reasoning.

Leo-III aims at general purpose theorem proving in HOL and should hence be based on a complete calculus. However, a primary goal of this work is to develop a theorem proving system that performs well in practice, regardless if its known not to be complete on a theoretical level or not. Such an approach is also reflected by ongoing developments in the ATP community: Since the development of Gandalf [Tam96], mode scheduling has been a major improvement to the overall success of ATPs. In mode scheduling, theorem proving systems run different configurations (i.e., vary parameters that influence the search space traversal) within a single invocation using a limited resource scheduling strategy. Here, some modes may intentionally sacrifice completeness due to empirical evidence that the restrictions perform reasonably well on (some subset of) problems. Current HOL ATP systems such as Satallax [Bro12] and TPS [AB06] use this approach as well. Additionally, there is an ongoing discussion about the use of strategies in reasoning systems [Wei17].

Finally, Leo-III natively supports reasoning within every normal higher-order modal logic [BvBW06]. Modal logics have many relevant applications in computer science, artificial intelligence, mathematics and computational linguistics. Many challenging applications, as recently explored in metaphysics, require first-order or higher-order modal logics (HOMLs). The development of ATPs for these logics, however, is still in its infancy. Leo-III is addressing this gap. In addition to its HOL reasoning capabilities, it is the first ATP that natively supports a very wide range of normal HOMLs. To achieve this, Leo-III internally implements a shallow semantical embeddings approach [BP13a, GSB17]. The key idea in this approach is to provide and exploit faithful mappings for HOML input problems to HOL that encode its Kripke-style semantics. These hybrid logic competencies make Leo-III, up to the author's knowledge, the most widely applicable ATP available to date.

This chapter introduces the Leo-III theorem prover, its underlying principles and implementation details. The general architecture and design of Leo-III is discussed in §4.1. In §4.2, additional, mostly practically motivated extensions to Leo-III's EP calculus are discussed. The proof search process, including preprocessing routines and search heuristics, and the resulting proof certificates of Leo-III are introduced in §4.3 and §4.5, respectively. The extensions to these procedures for admitting reasoning in polymorphic HOL are described in §4.6. Subsequently, the embedding approach that enables reasoning in higher-order modal logics and its utilization in Leo-III is presented in §4.7. Next, §4.8 sketches various implementation details. Finally, §4.9 presents further pragmatic features and functions of Leo-III.

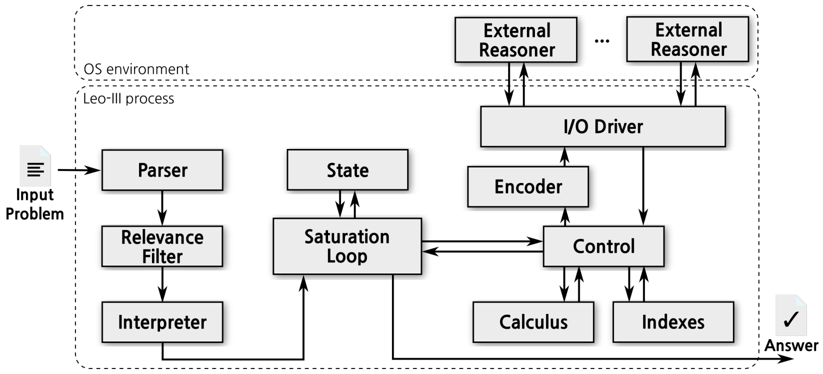
4.1. System Architecture

The fundamental architecture of Leo-III is quite complex due to the agent-based design of the system. Here, the initial idea of the project is to structure Leo-III as a massively parallel theorem proving system that employs various independent agents on top of a blackboard-based internal communication architecture. These agents are then supposed to act as specialists for certain reasoning tasks and exchange information (i.e. newly generated inferences) using the shared blackboard. The goal is thus to collaboratively find a proof using the so independently generated reasoning steps.

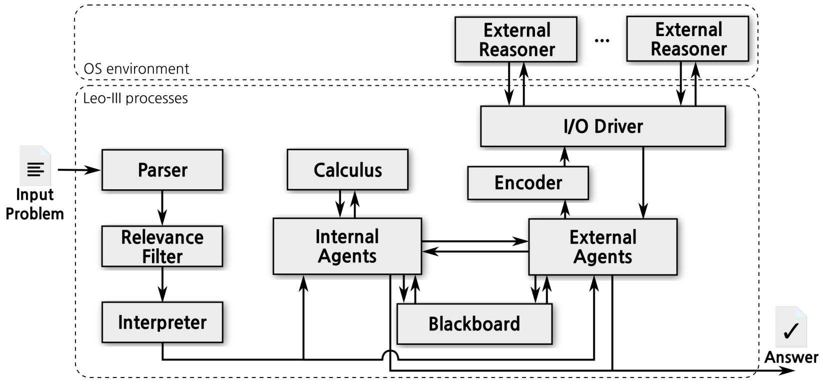
In principle, the employment of independent reasoning agents may speed-up the reasoning process as a whole since they may exploit the availability of multiple CPU cores and, hence, can be executed in parallel. In practice, however, the design of such a complex system with intricate relationships between reasoning tasks makes it necessary to use sophisticated control and scheduling layers and data consistency mechanisms while ensuring correctness of the reasoning process at the same time. As mentioned before, the implementation of Leo-III is roughly divided into two parts; the design and implementation of an agent-based architecture for automated theorem proving being an entire, quite separated, project itself. As a consequence, the system architecture of Leo-III as

Figure 1: Top-level system architecture of Leo-III. Each box roughly corresponds to a component from the implementation. The arrow visualizes the data flow through the system's design.

(a) Architecture for sequential setting.



(b) Architecture for agent-based setting.



designed by the author and presented in this section deviates from this ambitious goal. Nevertheless, it should be general enough to replace the reasoning procedures (as presented in the following sections) by an agent-based approach later on. The resulting functional design is schematically displayed in Fig. 1. Here, Fig. 1a displays the actual architecture currently implemented by Leo-III. The relevant components, described partially ordered by their use during an invocation of Leo-III, are:

Parser The parser transforms the ASCII-based input stream to an internal abstract syntax tree (AST) representation. The problem file can be read from local sources or via a network connection from a specified URL. The parser reports syntax errors if the input is malformed.

Relevance filter The relevance filter removes axioms from the input AST that are considered irrelevant for the subsequent proof search using syntactic heuristics. The relevance filtering techniques used by Leo-III are roughly sketched in §4.3.3.

Interpreter The interpreter translates the pruned input AST into a collection of typed λ-terms I and, in turn, creates a problem-specific signature Σ under which these terms are interpreted. The translated set is checked for shallow semantical flaws such as typing errors or use of unsupported operations (e.g. arithmetic in the case of Leo-III).

Saturation loop The λ-terms are passed to the main refutation procedure of Leo-III. The refutation procedure roughly decomposes into three stages:

1. Firstly, the input formulas are pre-processed. This operation produces, for each input term $s \in I$, an equisatisfiable set of clauses C_s. The overall result of this stage is a set of clauses given by $C := \bigcup_{s \in I} C_s$.
2. Then, the set C is saturated until either the empty clause is found or the system exceeds some provided resource limits (e.g. a time limit).
3. Finally, if the empty clause is found, a proof is constructed and presented to the user.

Pre-processing the individual input λ-terms include multiple techniques that are surveyed in §4.3.1. The saturation procedure is organized as a sequential loop algorithm which is described in detail in §4.3.2. An overview on the generated proof certificates and their generation is included in §4.5.

State The state captures all relevant information about the current proof search, including various sets of clauses, known unit equations, heuristic parameters and many more. The state itself is a passive object that mostly serves as a lookup device for important information required by the saturation loop or its control.

Control The so-called control of Leo-III is an abstraction layer between the concrete proof search algorithm and the implementation of its underlying calculus. Based on current state information and heuristics, the control decides how to use the low-level inference rules and further operations (e.g., external cooperation). Some details of the control layer abstraction are discussed in §4.3.2.

Calculus The calculus represents the low-level implementation of Leo-III's underlying inference rules. These implementations are never used directly by the saturation loop, but only indirectly through the control layer. The calculus implemented in Leo-III slightly modifies and augments the one described in §3. This includes e.g., multiple simplification rules and reasoning rules focused on heuristics. These rules are discussed in §4.2.

Indexes Indexing data structures are used to store terms, literals or clauses in such a way that certain query operations can be executed more efficiently. The indexing data structures used in Leo-III are surveyed in §4.8.2.

Encoder, I/O Driver Whenever the Leo-III control invokes an external cooperation attempt, the encoder translates the higher-order proof obligation into the target language (commonly first-order languages) and passes the transformed problem to the I/O driver. The driver creates new operating system processes for the desired cooperating systems and waits asynchronously for their results. All completed results are stored into the current state via the control layer and wait for further processing. External cooperation is described in detail in §4.4.

The strict separation between saturation, state, control and calculus may seem over-engineered in the sense that it introduces several data flow indirections and possibly implies a more complex code base. However, since the release of Leo-III 1.0 in 2016, this layered architecture has, on the contrary, been of great value to the improvement of the system.

For example, since version 1.1 of Leo-III, the mode of external cooperation has been completely revised. This was achieved thanks to the outlined separation of abstractions, with only local changes to the control layer, and without any further changes in the saturation loop (or elsewhere). Another example is the correction of various errors concerning the handling of polymorphic types in Leo-III 1.2. Here, the low-level inference mechanisms needed to be improved to avoid typing errors. However, no changes needed to be made at the control layer. These examples suggest that the multi-layer architecture of Leo-III indeed helps to isolate certain implementation tasks (improvements as well as corrections) to a minimum of affected routines where (often) the visible interface assertions stay unchanged.

Recall that Leo-III was initially layed-out as agent-based system. This scenario can also be captured by the presented architecture and, moreover, many of the above described components can be re-used. Fig. 1b displays how independent agents make use of these components. Here, the saturation loop as well as its state and control layer are replaced by concrete internal and external agents. These agents carry their own state and are controlled by a scheduler (replacing the control layer) that is based on combinatorical auctions. Agents may serve as specialists for certain tasks and can, for example, represent bundled calculus rules that are often used in combination. Agent-based theorem proving is a complex subject that is not included in the scope of this thesis project and is therefore not discussed further. An overview over the principles and design of agents and their the utilization for theorem proving can be found in the literature [Wei99, WB16, SWB16].

Challenges. The implementation of an effective theorem proving system for higher-order logic poses various challenges. Due to nature of HOL, tedious and computationally heavy procedures are required in order to produce a theoretically complete reasoning system. However it seems plausible that some of the theoretically necessary operations can be avoided, or at least restricted, without sacrificing a significant amount of reasoning effectivity. Nevertheless, it is not obvious how a reasonable balance between potentially inefficient yet complete and efficient but, in general, incomplete inference mechanisms is combined in practice. Some of the most important challenges are:

(1) *Primitive substitution.* In a higher-order setting, for completeness it needs to be ensured that free predicate variables can be instantiated with arbitrary formulas. An unrestricted use of primitive substitution may, however, generate infinitely many consequences even for a single occurrence of a free predicate variable.
Fortunately, primitive substitution can be restricted to only instantiate free variables that occur at the top-level with approximating bindings that imitate the logical connectives from the signature. An unrestricted use of these partial instantiations may still increase the size of the search space by magnitudes. As a consequence, practically effective means for the control of instantiations of free predicate variables are required. The challenge here is to find suitable heuristics that restrict the instantiations while not being too restrictive to hamper automation of relevant reasoning tasks.

(2) *Unification and extensionality.* Higher-order unification is undecidable and, additionally, not unitary. Nevertheless, eager application of unifi-

cation is a key operation for effective automation in most state-of-the-art HOL reasoning systems. The control of unification tasks hence incorporates of at least two main aspects: How should a reasonable search for higher-order unifiers be used to ensure termination of the operation, and, additionally, how many unifiers (if more than one is found) should be applied for subsequent reasoning? Apparently, both problems strongly influence the number of generated clauses and, hence, the growth rate of the overall search space.

Additionally, it is possible that a unification task is semantically solvable modulo extensionality. A further challenge is, consequently, to restrict the application of extensionality inferences in order to minimize the number of mutually recursive invocations of proof search and unification.

(3) *Paramodulation inferences.* In contrast to higher-order resolution calculi, the number of paramodulation inferences between a pair of clauses is greater by magnitudes. This is due to the fact that paramodulation acts on arbitrary subterms of each literal whereas resolution only considers inferences on top-level. As a result, for every pair of clauses, the number of freshly generated clauses by paramodulation tremendously increases the size of the search space. A major challenge is to restrict the paramodulation inferences in such a way that only relevant inferences are drawn. This includes the search for ordering techniques as employed in first-order superposition.

(4) *Equational simplification procedures.* In the context of equational first-order calculi, there are numerous powerful simplification techniques available that involve, in particular, unit equations. These techniques include rewriting but also contraction of clauses based on non-orientable unit equations.

Reasoning systems such as Leo-III would strongly benefit from suitably lifted simplification techniques to the higher-order setting. However, these techniques often employ orderings or matching procedures to identify simplifiable clauses. Full higher-order matching is decidable but, in general, inefficient and not suitable for light-weight simplification methods. Another challenge is thus to develop reasonable adaptions of equational simplification mechanism that lead to practically relevant improvements.

(5) *Indexing techniques.* Indexing data structures are used for quickly calculating relevant subsets of clauses (literals, terms) for regularly employed operations such as unification or matching. In the higher-order

case, few to none of such indexing techniques exist. Another, practically relevant challenge is thus to find indexing techniques that can be used to speed-up (some parts of) commonly used operations in HO reasoning.

4.2. Calculus Extensions

Leo-III features several additional calculus rules that are not captured by the formal EP calculus from §3. The rules presented here are practically motivated and primarily target technical issues that complicate effective automation within implementations of ATP systems. First and foremost, such rules address the inevitable problem of the explosive growth of the search space during proof search. This commonly includes the employment of various simplification techniques but, in the case of Leo-III, also heuristically guided inference mechanisms concerning guessing of specific instances that serve as counter-examples.

In particular, Leo-III uses specific rules for clause contraction (simplification), choice reasoning, defined equalities, function synthesis and higher-order unification. These rules are discussed in the following.

4.2.1. Clause Contraction

Clause contraction are among the most important operations of a competitive theorem proving system. Although the rules of EP are theoretically sufficient for a complete reasoning system, a considerable share of the overall inferences that are drawn during a system's execution are commonly used for contracting (that is removing or, in some sense simplifying) existing clauses. This is due to the fact that generally a lot of clauses are generated, e.g. by paramodulation, that are of no avail for the proof goal.

Additional to straight-forward simplification routines, Leo-III implements are variety of (equational) simplification procedures. Leo-III is, up to the author's knowledge, the first HOL ATP system exploiting sophisticated equational simplification methods. Many of these rules are adaptions of corresponding first-order simplification rules known from powerful first-order reasoning systems such as E. As discussed further below, it is not always possible or feasible to lift rules from the first-order domain to higher-order reasoning.

Following commonly used presentation conventions, modifying (or contracting) inference rules are written using double lines:

$$\begin{array}{c}\texttt{<main premise>}\qquad\texttt{<side premise 1>}\qquad\cdots\qquad\texttt{<side premise n>}\\ \hline\hline \texttt{<conclusion>}\end{array}$$

Figure 2: Simplification rules for simp. s and t denote an arbitrary formula and term of type τ, respectively. $\vee, \wedge, =$ and $\neq$ are identified with their symmetric variants.

$$
\begin{array}{rll rll}
s \vee s & \longrightarrow s & & s \wedge s & \longrightarrow s & \\
\neg s \vee s & \longrightarrow \top & & \neg s \wedge s & \longrightarrow \bot & \\
s \vee \top & \longrightarrow \top & & s \wedge \top & \longrightarrow s & \\
s \vee \bot & \longrightarrow s & & s \wedge \bot & \longrightarrow \bot & \\
t = t & \longrightarrow \top & & t \neq t & \longrightarrow \bot & \\
s = \top & \longrightarrow s & & s = \bot & \longrightarrow \neg s & \\
\forall X_\tau . s & \longrightarrow s & \text{if } X \notin \mathrm{fv}(s) & \exists X_\tau . s & \longrightarrow s & \text{if } X \notin \mathrm{fv}(s) \\
\neg \bot & \longrightarrow \top & & \neg \top & \longrightarrow \bot & \\
 & & & \neg\neg s & \longrightarrow s &
\end{array}
$$

Here, the leftmost premise (also referred to as the *main premise*) is by convention the premise that is being modified by the inference. If no conclusion is given, the inference denotes a *deleting rule* that effectively removes clauses from the search space. All possible side premises are regarded immutable (as only the main premise is being modified) and are not included in the conclusion. In particular, side premises are never deleted from the search space.

Formula simplification A straight-forward simplification procedure simp is specified by the transformation rules displayed at Fig. 2. These rules represent well-known Boolean identities and are thus obviously sound. Non-Boolean subterms remain unchanged. The resulting contraction rule (Simp) is given by:

$$
\frac{[l^1 \simeq r^1]^{\alpha_1} \vee \cdots \vee [l^n \simeq r^n]^{\alpha_n}}{[\mathsf{simp}(l^1) \simeq \mathsf{simp}(r^1)]^{\alpha_1} \vee \cdots \vee [\mathsf{simp}(l^n) \simeq \mathsf{simp}(r^n)]^{\alpha_n}} \text{ (Simp)}
$$

Since the procedure simp has linear time-complexity in the size of the term to be simplified, the calculus rule (Simp) can be implemented very efficiently.

Clause simplification Clause simplification rules allow the elimination of literals from clauses or even the deletion of whole clauses from the search space. A representative of literal elimination inferences is rule (DD):

$$
\frac{\mathcal{C} \vee [s \simeq t]^\alpha \vee [s \simeq t]^\alpha}{\mathcal{C} \vee [s \simeq t]^\alpha} \text{ (DD)}
$$

This inference rule eliminates duplicated literals from clauses. This operation is sound as it only exploits idempotency of disjunction. For first-order ATP systems, deletion of resolved literals is usually another contraction rule. In this setting, this inference is already captured by rule (Triv) which is related to higher-order unification.

Tautological clauses are those that are implied by any theory. Such clauses increase the search space without contributing any helpful information towards to proof goal. The following two inference rules remove tautological clauses that can efficiently be recognized:

$$\frac{\mathcal{C} \vee [s \simeq s]^{\mathtt{tt}}}{}\ (\text{TD1}) \qquad \frac{\mathcal{C} \vee [s \simeq t]^{\mathtt{tt}} \vee [s \simeq t]^{\mathtt{ff}}}{}\ (\text{TD2})$$

Both rules are sound as they only remove clauses from the search space. In particular, (TD1) and (TD2) implement reflexivity of equality and the law of excluded middle. First-order ATP systems often use another rule, called destructive equality resolution. In the context of EP, this is already captured by the unification rule (Bind).

Clause-Clause contraction Clause-clause contractions simplify or remove clauses based on information provided by further clauses from the search space. Unit equations provide the most important means for simplification techniques in state-of-the-art equational first-order theorem provers. This includes, in particular, rewriting using unit equations that can be oriented with respect to suitable term orderings. In superposition-based first-order reasoning systems, corresponding term orderings are generally implied by the underlying calculus. As discussed in §3, there is currently no higher-order ordering available that meets the requirements of their first-order counterparts. Nevertheless, Leo-III implements rewriting using a higher-order term ordering (cf. further below).

Any unit equation can be used for simplifying a clause if one literal of that clause can be eliminated. More formally, positive unit equations are exploited by the so-called *positive simplify-reflect* rule that is captured by rule (PSR):

$$\frac{\mathcal{C} \vee [s \simeq t]^{\mathtt{ff}} \qquad [l \simeq r]^{\mathtt{tt}}}{\mathcal{C}}\ (\text{PSR})$$

if there is a substitution σ that matches the literal $[s \simeq t]$ against the unit equation $l \simeq r$, i.e. $\sigma(l \simeq r) \equiv s \simeq t$. Quite recent results show that higher-order matching is decidable [Sti09]. Nevertheless, it is known to have non-elementary complexity [Sta77, Wie99] and seems inadequate for an effective automation procedure. In Leo-III, a matching algorithm based on higher-order pattern unification is used this scenario. This is, of course, a trade-off solution that tries to mitigate the above situation.

Analogously, negative unit equations are exploited by *negative simplify-reflect* (NSR):

$$\frac{\mathcal{C} \vee [s \simeq t]^{\mathtt{tt}} \qquad [l \simeq r]^{\mathtt{ff}}}{\mathcal{C}} \text{(NSR)}$$

if there is a substitution σ with $\sigma(l \simeq r) \equiv s \simeq t$. Soundness of (PSR) is straightforward: In any interpretation $\mathcal{M}$ that validates $[l \simeq r]^{\mathtt{tt}}$, all instances of $[l \simeq r]^{\mathtt{ff}}$, including $[s \simeq t]^{\mathtt{ff}}$, can be simplified to $\bot$ and hence eliminated from the (disjunctive) clause. Analogously for (NSR).

Another contraction technique that uses unit equations is rewriting which has been developed in the context of Knuth-Bendix completion [KB70]. Leo-III uses the computablility path ordering (CPO) by Blanqui et al. [BJR15] for its rewriting procedures. CPO is a well-founded and monotone higher-order term ordering that has been developed in the context of higher-order termination. The ordering relation induced by CPO is denoted $\succ_{\text{CPO}}$ is the following.

$$\frac{\mathcal{C} \vee [s \simeq t]^{\alpha} \qquad [l \simeq r]^{\mathtt{tt}}}{\mathcal{C} \vee [s[r\sigma]_p \simeq t]^{\alpha}} \text{(RW)}$$

if $s|_p \equiv l\sigma$ for some substitution σ, where $l\sigma \succ_{\text{CPO}} r\sigma$. Furthermore, if $\alpha \equiv \mathtt{tt}$, either $p \not\equiv \varepsilon$ or $s \not\succ t$ or σ is not a renaming substitution. While the first conditions ensure that matching terms are replaced by (wrt. $\succ_{\text{CPO}}$) smaller equivalents, the latter condition prevents a destructive self-rewrite of positive unit equations. Soundness of rule (RW) is a direct consequence of substitutivity of equality.

In contrast to rewriting in first-order, rewriting in HOL can also be applied to formulas, that is, on terms of Boolean type. As a consequence, also non-equational unit clauses such as $\mathcal{C}\colon [p]^{\mathtt{tt}}$ can be utilized for rewriting.[3] Moreover, orientable negative unit equations $p_o \simeq q_o$ between Boolean terms can be employed as a rewrite rule $p \rightarrow \neg q$.

Clause subsumption (CS) is a popular method for reducing the size of the search space by removing clauses for which more general variants have already been inferred. More formally, a clause $\mathcal{C}$ may be deleted from the search space if there exists a smaller clause that generalizes a subset of $\mathcal{C}$'s literals. The rule (CS) is given by:

[3] Note that it is customary to set-up a term ordering $\succ$ such that $s \succ \top \succ \bot$ for any term s. Hence, any unit clause $[p]^{\mathtt{tt}}$ with $p \not\equiv \bot$ and $p \not\equiv \top$, represents an oriented equation $p \simeq \top$. Similarly, a negative unit clause $[p]^{\mathtt{ff}}$, with $p \not\equiv \bot$, represents an oriented equation $p \simeq \bot$.

$$\frac{\mathcal{C} \lor \mathcal{C}' \qquad \mathcal{D}}{}\ (\text{CS})$$

if $\mathcal{D}\sigma \equiv \mathcal{C}'$ for some substitution σ. This rule is sound as, again, clauses are only removed from the search space. Again, only a restricted form of higher-order matching is employed by Leo-III for the implementation of rule (CS).

4.2.2. *Defined Equalities*

Next to the use of the interpreted equality connective ”=” (referred to as *primitive equality*), equality predicates can be defined in HOL. Popular examples of defined equalities are

$$\lambda X_\tau . \lambda Y_\tau . \forall P_{o\tau} . P\, X \Longrightarrow P\, Y \tag{1}$$

and

$$\lambda X_\tau . \lambda Y_\tau . \forall R_{o\tau\tau} . \forall Z_\tau . R\, Z\, Z \Longrightarrow R\, X\, Y \tag{2}$$

which are known as *Leibniz equality* and *Andrews equality* [And02a], respectively.

Both variants support cut-simulation [BBK09] and might, hence, hinder proof search due to free set variables that are subject to primitive substitution. Leo-III scans for the above definitions of equality predicates instantiates them with primitive equality:

$$\frac{\mathcal{C} \lor [P\, s]^{\mathfrak{ff}} \lor [P\, t]^{\mathfrak{tt}}}{\mathcal{C}\{\lambda X . s = X / P\} \lor [s \simeq t]^{\mathfrak{tt}}}\ (\text{LEQ}) \qquad \frac{\mathcal{C} \lor [P\, s\, s]^{\mathfrak{ff}}}{\mathcal{C}\{\lambda X . \lambda Y . X = Y / P\}}\ (\text{AEQ})$$

Note that infinitely many different equality predicates can be defined in HOL. A simple syntactic recognition of these predicates is thus necessarily incomplete. These rules have originally been developed in the context of LEO-II [BS13]. In Leo-III, however, the user can specify whether these rules are applied as replacing inferences or as standard, generating, inferences (the latter is used by default).

(LEQ) and (AEQ) are sound since they are specific instances of primitive substitution rule (Prim).

4.2.3. *Choice*

The axiom scheme of choice is, for every type τ, given by

$$\exists E_{\tau(o\tau)} . \forall P_{o\tau} . (\exists X_\tau . P\, X) \Longrightarrow P\, (E\, P) \tag{3}$$

This axiom scheme postulates the existence of an operator E that chooses an element satisfying a given predicate, if such an element exists. Leo-III is designed as an ATP system for HOL with choice. To that end, $\varepsilon^{\tau}_{(\tau\to o)\to\tau}$ is a primitive choice operator for type τ that satisfies the above property. However, instances of (3) might also be postulated directly within HOL problems. These instances allow cut-simulation [BBK09] and hence might impede proof search. Following LEO-II, Leo-III keeps track of user-defined choice functions using a set CF where initially $\mathsf{CF} \equiv \{\varepsilon^{\tau}_{(\tau\to o)\to\tau} \mid \tau \in \mathcal{T}\}$. Corresponding choice instances are removed from the search space by rule (ACD):

$$\frac{[P\,X]^{\mathrm{ff}} \vee [P(f\,P)]^{\mathrm{tt}}}{\quad} \text{(ACD)}$$

where, as a side-effect, the set of choice functions is augmented by $\mathsf{CF} := \mathsf{CF} \cup f$. This is obviously sound as (ACD) only removes clauses from the search space.

In order to enable reasoning with (primitive or user-defined) choice operators, Leo-III instantiates concrete choice instances for given choice operators and predicates:

$$\frac{\mathcal{C}' \vee [s[E\,t]]^{\alpha}}{[t\,X]^{\mathrm{ff}} \vee [t\;(\varepsilon\,t)]^{\mathrm{tt}}} \text{(ACI)}$$

where $E \in \mathsf{CF}$ or $E \in \mathrm{fv}(\mathcal{C})$ is a free variable to the clause, $\mathrm{fv}(E) \subseteq \mathrm{fv}(\mathcal{C})$ and X is a fresh variable. Rule (ACI) instantiates choice with subterms that represent either concrete choice operator applications (if $E \in \mathsf{CF}$) or potential applications of choice (if E is a free variable to the clause). In contrast to LEO-II, Leo-III supports choice operators as interpreted constant symbols and hence does not need to provide fresh uninterpreted constant symbols for each type, serving as choice operators, on-the-fly (i.e. during application of (ACI)).

Rule (ACI) is sound as it introduces specific instances of the axiom of choice, which is a theorem in the logic addressed by Leo-III, to the search space.

4.2.4. Heuristic Instantiation

Prior to clause normalization, Leo-III might instantiate universally quantified variables. Experiments with TPS suggest that early instantiations of such variables with complex terms might improve proof attempts in some cases [ABI+96]. In Leo-III, instantiation of universally quantified variables in formulas is captured by a general *heuristic instantiation* rule (HeuInst):

$$\frac{\mathcal{C} \vee [s[\forall X_{\tau}.\,u] \simeq t]^{\alpha} \qquad v \in \mathrm{Heu}^{\tau}}{\mathcal{C} \vee [s[u\{v/X\}] \simeq t]^{\alpha}} \text{(HeuInst)}$$

where Heu^{τ} is a (possibly empty) set of closed terms of type τ that is a parameter to the rule.

Currently implemented heuristics include exhaustive instantiation of finite types such as o, oo and ooo as well as partial instantiation for otherwise interesting types. Examples for the latter class include (where τ and the τ^i are some types): $o\tau^n \cdots \tau^1$ (predicate/relation types), $o\tau\tau$ (equality/inequality), $\tau(o\tau)$ (choice terms), and $\tau\tau\tau o$ (instances of if-then-else). Additionally, an experimental syntactical detection of finite types is incorporated into Leo-III. The recognized types can then be exhaustively instantiated.

Also, recent applications of Leo-III for studies in theoretical philosophy suggest that it might be fruitful to include means for externally specifying certain terms that are deemed meaningful or relevant for the reasoning task at hand. That way, the user can give hints to the reasoning system which terms could be considered for instantiation during proof search by (HeuInst). In practice, this usually includes terms that are too complex to be enumerated by primitive substitution in reasonable resource bounds.

4.2.5. Function Synthesis

Function synthesis denotes the generation of syntactical representations of functions that meet a given specification. The general formulation of this problem has many applications in e.g. program synthesis [KMPS10], software and hardware verification and invariant discovery. In the context of automated reasoning, a strong connection between reasoning and function synthesis has already been highlighted in the 1980s [Gre81, MW80]. In particular, powerful function synthesis mechanisms have been developed for utilization by SMT systems such as CVC4 [B⁺11] and Z3 [DMB08].

Function synthesis tasks can be regarded as validity problems of the form

$$\exists F_{\nu\tau_n\cdots\tau_1}. \forall X^1_{\tau_1}. \cdots \forall X^n_{\tau_n}. P[F, \overline{X^i}] \tag{4}$$

where F is the function symbol that is conjectured to have the property represented by the predicate meta-variable P. Such conjectures are also called *synthesis conjectures* [RDK⁺15] and are, in the refutational setting of Leo-III, transformed by initial negation and subsequent Skolemization to a formula

$$\forall F_{\nu\tau_n\cdots\tau_1}. \neg P[F, \overline{\mathrm{sk}^i}] \tag{5}$$

where the sk^i are fresh Skolem terms of appropriate type. The task is thus to show the unsatisfiability of the negation of the desired property for function F.

Let C be a set of clauses produced by clausification of the above formula (5). Leo-III scans for clauses $\mathcal{C} \in C$ of the form

$$\mathcal{C} \equiv \mathcal{C}' \vee [F\ \overline{s^{1,j}}^{1\leq j\leq n} \simeq t^1]^{\mathfrak{ff}} \vee \cdots \vee [F\ \overline{s^{m,j}}^{1\leq j\leq n} \simeq t^m]^{\mathfrak{ff}} \tag{6}$$

that represent a specification of a function F by explicit enumeration of its input-output pairs.

Since these input-output pairs are represented by negative equations, they denote unification problems that are tackled using built-in unification procedures. However, even for simple problems, such an approach fails. Consider the specification of a function $F_{\iota\iota}$ where $F\ a = b$ and $F\ b = a$ for some constants a_ι, b_ι. The negated, clausified conjecture of the existence of such an function reads

$$\mathcal{C} \equiv [F\ a \simeq b]^{\mathfrak{ff}} \vee [F\ b \simeq a]^{\mathfrak{ff}}$$

When solved individually, both unification literals results in solutions that do not establish the unsatisfiability of the clause. When regarding both literals simultaniously, the unification task cannot be solved (note that the formula is only valid in Henkin semantics with choice, but counter-satisfiable if the axiom of choice is not valid):

1. When solving the first literal, pre-unification will result in a solution $\sigma = \{\lambda X_\iota . b/F\}$. Applying σ results in the clause $\mathsf{simp}(\mathcal{C}\sigma) \equiv [b \simeq a]^{\mathfrak{ff}}$.
2. When solving the second literal, pre-unification will result in a solution $\sigma = \{\lambda X_\iota . a/F\}$. Applying σ results in the clause $\mathsf{simp}(\mathcal{C}\sigma) \equiv [a \simeq b]^{\mathfrak{ff}}$.
3. When regarding both literals for unification simultaniously, pre-unification will fail as no syntactical unifier σ exists such that $(F\ a)\sigma \equiv b\sigma$ and $(F\ b)\sigma \equiv a\sigma$.

In order to remedy this situation, Leo-III can try to synthesize a function given an according goal specification. The idea is as follows: In a higher-order setting, a function can simply be constructed by a choice term that specifies the function's properties. These properties are, as outlined above, given by the desired input-output relations of the function symbol. A corresponding choice term that represents these input-output relations can be constructed by an enumeration of all explicitly given function value cases. This term can be regarded a (partially defined) *if-then-else* term. The resulting inference rule that approximates function specifications given by clauses (6) is given by:

$$\frac{\mathcal{C}' \vee [F\ \overline{s^{1,j}}^{1\leq j\leq n} \simeq t^1]^{\mathfrak{ff}} \vee \cdots \vee [F\ \overline{s^{m,j}}^{1\leq j\leq n} \simeq t^m]^{\mathfrak{ff}}}{\mathcal{C}\Big\{\lambda \overline{X^j}. \varepsilon Z_\iota .\ \bigwedge_{k=1}^{m} \Big(\big(\bigwedge_{j=1}^{n} X^j = s^{k,j}\big) \longrightarrow Z = t^k\Big)/F\Big\}}\ \text{(FS)}$$

The term

$$\lambda \overline{X^j}.\, \varepsilon Z_\iota.\, \bigwedge_{k=1}^{m} \Big(\big(\bigwedge_{j=1}^{n} X^j = s^{k,j} \big) \longrightarrow Z = t^k \Big) \tag{7}$$

instantiated by rule (FS) roughly corresponds to a nested if-then-else term of the form

```
if ($X^1 = s^{1,1} \wedge \ldots \wedge X^n = s^{1,n}$) then $t^1$
else if ($X^1 = s^{2,1} \wedge \ldots \wedge X^n = s^{2,n}$) then $t^2$
...
else if ($X^1 = s^{m,1} \wedge \ldots \wedge X^n = s^{m,n}$) then $t^m$
```

where the X^j, $1 \leq j \leq n$, are the parameters of the function, the $s^{k,j}$, $1 \leq k \leq m$, are the concrete argument for each X^j such that t^k is produced as result of the function. In the swapping function example, the so generated term can be expressed as

```
if ($X = a$) then $b$
else if ($X = b$) then $a$
```

where X is the single parameter of the function substituted for F. This conditional is then given by a specific instance of (7) with $m = 2$ and $n = 1$, i.e. the choice term

$$\lambda X_\iota.\, \varepsilon Z_\iota.\, (X = a \longrightarrow Z = b) \wedge (X = b \longrightarrow Z = a)$$

This example is revisted in §5.1 as example *E7* where also a complete refutation is presented.

Leo-III employs rule (FS) if plain unification fails for a set of unification constraints. In general, this rule tremendously increases the search space but can, however, enable Leo-III to solve hard problems. In a way, rule (FS) converts the syntactical problem of finding a suitable function instance into a semantic search problem for discharging the hypothesis that the conjectured function instance (the choice term) actually exists. In conjunction with rule (ACI), this enables Leo-III to solve problems for which otherwise no function instance could be derived (within reasonable resource bounds).

In contrast, in first-order reasoning and SMT solving, concrete closed function representations are synthesized. This requires more involved and specialized techniques [RDK+15, KKKS13, JK11] that usually pose additional restrictions on the synthesis conjectures or the search space. As an example, popular synthesis techniques require the synthesis conjectures to be a *single invocation property* [JK11]. The here presented approach does not pose such restrictions. Nevertheless, the *refutation-based synthesis* method [RDK$^+$15] as employed by CVC4 is strongly related to this approach. The main difference is that Leo-III does not validate the existence of a concrete function instance but rather exploits a choice operator for that purpose.

Rule (FS) is sound as it simply instantiates the universally quantified variable by a concrete term. In particular, if no such function instance exists, rule (ACI) guards against unsound conclusions.

Additionally, Leo-III supports improved reasoning with injective functions. Leo-III scans for occurrences of injectivity axioms and, for each recognized injective function $f_{\nu\tau}$, postulates the existence of a left-inverse function $f^{\text{inv}}_{\tau\nu}$. As an example, if f is an unary function, the formal relation is given by $f^{\text{inv}}\ (f\ X) = X$ where X_τ is a variable of type τ. This is captured for function symbols of any arity by the following rule:

$$\frac{[f\,X^1\ \cdots\ X^{i-1}\,Y\,X^{i+1}\ \cdots\ X^n \simeq f\,X^1\ \cdots\ X^{i-1}\,Z\,X^{i+1}\ \cdots\ X^n]^{\mathtt{tt}} \vee [Y \simeq Z]^{\mathtt{ff}}}{[f^{\text{inv}}\,X^1 \cdots X^{i-1}\,X^{i+1}\ \cdots\ X^n\ (f\,X^1\ \cdots\ X^n) \simeq X^i]^{\mathtt{tt}}}\ \text{(INJ)}$$

where f^{inv} is a fresh Skolem constant and the X^j, $1 \le j \le n$, are variables of appropriate type. The premise expresses that f is injective in its i-th argument.

This rule is sound as every injective function in known to be left-invertible. The practical benefits of including rule (INJ) into Leo-III are discussed in §5.1 where also an example is presented (cf. example *E2*). A similar technique is implemented by Z3 [DMB08].

4.2.6. Pattern Unification

A central operation within higher-order ATP systems is unification of λ-terms. Unification procedures are frequently invoked during proof search and constitute a notable share of the overall computational effort, and hence also influence the effectivity and efficiency of the reasoning system. As depicted in §3, higher-order unification is undecidable and produces, in general, infinitely many unifiers for a given unification task. In practice, unification procedures of HO reasoning systems usually base upon Huet's unification algorithm [Hue75] which performs well enough for most applications. However, it has been observed by Miller [Mil91a] that there exists a subclass of higher-order unification problems for which unification is indeed decidable and unitary, i.e. there is a procedure to find a most general unifier if such a unifier exists. This class of λ-terms is called *higher-order patterns*, or simply *patterns*. They allow for a simple unification algorithm and, as it turns out, occur very often in practice [Mil91a, Nip91].

A higher-order pattern is a λ-term $s \in \Lambda_\tau$ in β-normal form for which the following property holds: Every free variable $F_\tau \in \text{fv}(s)$ of s only occurs in subterms $t \triangleleft s$ of the form $t \equiv F\ \overline{X^i_{\tau_i}}$, where the X^i are η-equivalent to distinct bound variables.

Higher-order pattern unification is known to have linear time complexity [Qia96]. However, the linear time algorithm by Qian is based on a special imperative term representation and seems overly complicated for a practical utilization by Leo-III. Leo-III implements a variant of pattern unification that is based on a formulation by Nipkow [Nip93] with minor modifications concerning the representation of free and bound variables. Although this variant has exponential run time in worst case, it is of great value for many HO reasoning systems, including Isabelle/HOL as it performs quite well in practice for many inputs.

Leo-III prefers HO pattern unification to plain pre-unification whenever possible. If a unification task is, however, not in the pattern fragment, standard higher-order pre-unification is used. The control of unification during proof search is described in more detail in §4.3.2.

4.3. Proof Search

The proof search is organized as a sequential procedure that iteratively saturates the set of input clauses with respect to the inference rules from §3 and §4.2 until the empty clause is found.[4] Since in most cases the number of clauses that can be generated during proof search is not finite, the actual enumeration of inferences is limited artificially using time resource bounds (often called the *timeout*) that can be configured by the user. If the proof search exceeds the specified timeout, the system terminates even if the empty clause was not found and report that it is unknown whether the initial set of clauses is inconsistent or not. See §4.5 for details on the system's result output format.

For the remainder of this section, it is assumed that the input problem has already been parsed and translated to an internal representation of typed λ-terms. This input, denoted I, is assumed to be a finite sequence $I = I_1, I_2, \ldots$ of well-typed λ-terms.[5] Additionally, a problem-specific signature Σ is generated during

[4] Recall that a clause is said to be empty if it is the empty set or only consists of flex-flex unification constraints, see the remark in §4.3.2.

[5] Note that the input can be regarded a finite sequence without loss of generality since it is assumed that the input problem statements are of finite size and, hence, only contain finitely many formulas.

interpretation of the input (cf. §4.1). Note that I may only consist of a subset of those formulas contained in the input problem due to preceding procedures such as relevance filtering. Nevertheless, if a set of clauses constructed from I can be shown unsatisfiable during proof search, then any super set containing I is evidently unsatisfiable and hence so is the input problem.

The overall proof search procedure consists of three consecutive phases (in that order):

1. **Pre-Processing.** Prior to saturation, the input formulas are pre-processed to a fully Skolemized $\beta\eta$-normal clausal normal form. In addition, methods including definition expansion, simplification, miniscoping, replacement of defined equalities, and clause renaming are applied.
2. **Saturation.** Saturation is done using a sequential loop procedure similar to the given clause algorithm as used by the E prover [Sch02].
3. **Proof reconstruction.** If the empty clause was inferred during saturation and the user requested a proof output, a proof object is generated using backwards traversal of the respective search subspace. Proofs in Leo-III are presented as TSTP CNF refutations [Sut10] in THF format.

The first two aspects are discussed in the following. The presentation of the proof reconstruction algorithm as well as the resulting proof's syntactical representation is postponed until §4.5 since it does not strongly relate to the actual proof search described in this section.

4.3.1. Pre-Processing

Pre-processing is a key procedure in modern ATP systems that modifies the input problem in such a way that it is somehow simpler, smaller or better suited for the subsequent saturation. Typically, pre-processing includes a transformation of formulas into a particular normal form and the elimination of redundant information. The goal of the former is to allow a uniform treatment of identical formulas (with respect to the normal form) by choosing a certain syntactical representative. The latter reduces the size of the resulting search space. However, pre-processing can also increase the size of the initial set of formulas by augmenting the problem with additional information that seems relevant to the search process. This may include heuristic insertion of theory axioms or instantiation of specific formulas that are already contained in the problem. Finally, during pre-processing, characteristics about the problem itself may be collected for later utilization. Such information can for example be used for selecting specialized search strategies or relevance filtering techniques.

For hard reasoning tasks, the right choice of pre-processing techniques is often the key for finding the solution. Leo-III uses the following functions that operate on formulas or clauses:

expandDefs(ϕ) returns a formula ϕ' that is generated by replacing all defined constant symbols in ϕ by their respective definition.

propSimp(ϕ) applies the simp simplification procedure from Fig. 2 to ϕ and returns the result $\mathsf{simp}(\phi)$.

heuristicInstances(ϕ) applies rule (HeuInst) to ϕ and returns all instances. For each type $\tau \in \mathcal{T}$, the particular sets Heu^{τ} of instantiations used in Leo-III are:

$$\begin{aligned}
\text{Heu}^{o} &:= \{\top, \bot\}, \\
\text{Heu}^{oo} &:= \{\lambda X_o.\, \top, \lambda X_o.\, \bot, \lambda X_o. X, \lambda X_o.\, \neg X\}, \\
\text{Heu}^{ooo} &:= \{\lambda X_o.\, \lambda Y_o.\, \top, \lambda X_o.\, \lambda Y_o.\, \bot, \lambda X_o.\, \lambda Y_o. X, \lambda X_o.\, \lambda Y_o. Y, \\
&\qquad \lambda X_o.\, \lambda Y_o.\, \neg X, \lambda X_o.\, \lambda Y_o.\, \neg Y, \ldots\}, \\
\text{Heu}^{o\tau} &:= \{\lambda X_\tau.\, \top, \lambda X_\tau.\, \bot\}, \\
\text{Heu}^{o\tau\tau\tau} &:= \{\lambda P_o.\, \lambda X_\tau.\, \lambda Y_\tau.\, \text{ite}\, P\, X\, Y\}, \\
\text{Heu}^{\tau(o\tau)} &:= \{\lambda P_{o\tau}.\, \varepsilon^{\tau}\, P, \lambda P_{o\tau}.\, \varepsilon^{\tau}\, \neg P\}
\end{aligned}$$

Here, only the sets Heu^{o}, Heu^{oo} and Heu^{ooo} are exhaustive enumerations of instances of the respective type. Leo-III hence eliminates universal quantifiers for such types by exhaustive instantiation. The remaining instances are purely heuristic and do not eliminate the original universal quantification. The instances $\lambda X_\tau.\, \top$ and $\lambda X_\tau.\, \bot$ of $\text{Heu}^{o\tau}$ represent canonical instances of the full and empty set of elements of type τ, respectively. Finally, $\text{Heu}^{o\tau\tau\tau}$ and $\text{Heu}^{\tau(o\tau)}$ contain an abstract if-then-else term and two instances of choice terms, respectively.

miniscope(ϕ) applies miniscoping [NW01] to the formula ϕ and returns the result.

cnf(ϕ) exhaustively applies all clausification rules from $\mathcal{CNF}$ to ϕ and returns the resulting clause set. By default, Leo-III uses definitional clausification [NW01] as opposed to plain CNF calculation.

convertDefinedEqs(t) applies the rules (LEQ) and (AEQ) to the clause t and returns all results. Note that both rules do not, by default, replace the original clause but merely add instances of primitive equality. Hence, it holds that $t \in$ `convertDefinedEqs(`t`)`.

liftEq(t) applies rule (LiftEq) to clause t and returns the result.

Figure 3: Pre-processing procedure of Leo-III. The input is a list I of well-typed formulas. The procedure returns a set of proper clauses U in $\beta\eta$-normal form that is equisatisfiable to I.

```
U := $\emptyset$
while not empty($I$) do
  $T$ := $\emptyset$
  $\phi$ := head($I$)
  $I$ := tail($I$)
  if $\phi$ is a conjecture then
    $\phi$ := $\neg\phi$
  endif
  $\phi$ := propSimp(expandDefs($\phi$)$\downarrow_{\beta\eta}$)
  $T$ := $\{\phi\} \cup$ heuristicInstances($\phi$)
  $T$ := $\bigcup_{t\in T}$ cnf(miniscope($t$))
  $T$ := $\bigcup_{t\in T}$ convertDefinedEqs($t$)
  $T$ := $\{$simplify(liftEq($t$))$\,|\, t \in T\}$
  $T$ := $T \cup \bigcup_{t\in T}$ detectInjectivity($t$)
  $U$ := $U \cup \{t \in T\,|\,$not trivial($t$)$\}$
end
return $U$
```

simplify(t) applies the normalization rules (DD), (DR) to t as well as the simplification routine simp and returns the result.

detectInjectivity(t) applies rule (INJ) to t and returns the result, if applicable.

trivial(t) returns true if and only if t can be shown to be tautological using (TD1) and (TD2).

Note that the input of this procedure is a list of formulas I. For standard list operations the following functions are used: empty(L) is a predicate that is true if and only if L is the empty list, head(L) returns the first element of L and tail(L) gives a list that is constructed from L by removing its first element. Syntax errors and parsing errors are handled prior to pre-processing, in particular each formula in I is assumed to be well-typed in the following. Leo-III aborts its proof search early if any syntax error occurs.

Fig. 3 displays the pre-processing procedure implemented by Leo-III. The pre-processing routine works as follows: Each input formula ϕ is successively transformed into an equisatisfiable set of clauses. Each of these sets is processed by multiple normalization procedures and the results are aggregated into the final input set U that is subject to subsequent saturation. Since Leo-III is a refutation-based theorem prover, the conjecture, if existent, is negated prior to further processing (cf. lines 6-8). Given an input formula ϕ, first every defined constant symbol occurring in ϕ is expanded and straight-forward simplification techniques are applied to the result. A miniscoping procedure minimizes the number of dependencies of existentially quantified subformulas which, in turn, will in some sense simplify the Skolem terms introduced later. The resulting formula is then exhaustively clausified using definitional clausification [NW01, WSKB16] in order to reduce the number of resulting clauses. Additionally, further heuristically determined instances of ϕ are considered for clausification (cf. line 10). As a next step, recognized defined equalities are instantiated by primitive equality (cf. line 12). Also, each of the clauses is scanned for occurrences of injectivity specifications (cf. line 14). If such a function is found, a fresh function symbol (serving as left-inverse) of appropriate type is generated, added to Σ, and a corresponding axiom included to the result set. Finally, the partial pre-processing result for ϕ is constructed by again simplifying the intermediate clauses and removing the contained tautologies (cf. lines 13,15).

The output of the above pre-processing procedure is a set U of proper clauses in $\beta\eta$-normal form. Currently, pre-processing in Leo-III does not include aggregation of statistical data or other forms of characteristic information about the input problem. However, recent experiments suggest that various analysis methods for such information can fruitfully be employed for improving search heuristics [FB16, KU15a, LISK17, GKU17] and relevance filtering [USPV08, WTWD17]. Recent work experiments with suitable representations of Leo-III's proof certificates as data types for utilization in learning or optimization procedures [Zie18]. However, concrete improvements to Leo-III using these procedures remain further work.

4.3.2. Saturation Procedure

A calculus does not define the order in which the inference rules should be applied, i.e. any arbitrary (chaotic) application scheme of inference rules is, in theory, suitable. For completeness, however, it needs to be ensured that every possible inference is eventually generated by the application scheme (cf. notion of *fairness*). In practice a class of loop procedures often referred to as *given-clause algorithms* [Sch02] has been established for saturating a set of clauses with respect to a set of inference rules.

In these saturation procedures, the clausal search space is structured using two sets U and P of *unprocessed* clauses and *processed* clauses, respectively. Initially, P is empty and U contains all clauses generated from the input problem. Intuitively, the algorithm iteratively selects an unprocessed clause g (the *given clause*) from U. If g is the empty clause, the initial clause set is shown to be inconsistent and the algorithm terminates. If g is not the empty clause, all inferences involving g and (possibly) clauses in P are generated and inserted into U. The resulting invariant is that all inferences between clauses in P have already been performed.

Most current saturation-based ATP systems implement a variant of the given-clause procedure. Two particular shapes of this algorithm have been popularized in the past, called the *Otter loop* and the *DISCOUNT loop*. Each procedure is thereby named after the first well-known ATP system that implemented it [McC94, DKS97]. While both variants share the same iterative approach, they differ in the way how simplification inferences are applied. Otter loops usually use various different subsets of the search space for different simplification procedures. In contrast, the DISCOUNT loop only employs simplifications induced by P and g.

Leo-III uses a variant of the DISCOUNT loop that has its intellectual roots in the E prover. Hence, the general layout of Leo-III's saturation algorithm is quite similar to the one of E. Nevertheless, some modifications are necessary to address the specific requirements of reasoning in HOL (as opposed to first-order logic with equality in the E prover). The concrete saturation procedure of Leo-III is displayed in Figure 4, its components as well as the differences between E's DISCOUNT loop are discussed in the following. It is assumed that the pre-processed clauses (see above) have been added to U prior to the saturation procedure invocation.

The saturation loop works as follows: A clause g is heuristically selected and simplified with respect to P (lines 5,7). Note that since `simp` also applies simplification using unit clauses, simplification with respect to P may create non-CNF clauses even if g was initially in clause normal form. Non-CNF results are reinserted into U for later selection whereas all other (non-redundant) results are used for backward simplification of P (lines 16,18). Finally, the generated inferences are normalized, unified (if possible) and simplified using only inferences that are particularly efficiently implemented (lines 20-22).

Additionally to the internal reasoning capabilities of Leo-III, the employment of external ATP systems is tightly integrated into the proof loop procedure. At every iteration of the main loop it is checked whether any previous invocation of external reasoners was successful (line 2-4), i.e. if any external reasoner

Figure 4: Saturation procedure of Leo-III. The input of this procedure is a set U of clauses that originate from pre-processing (cf. Fig. 3). The algorithm succeeds and terminates if the empty clause is derived. The saturation loop is externally interrupted and a failure result is returned if a user-specified time limit is exceeded.

```
while $U \neq \emptyset$ do
  if checkExternal() then
    return success; initial $U$ is unsatisfiable
  endif
  $g$ := selectBest($U$)
  $U$ := $U \setminus g$
  $g$ := simp($g$,$P$)
  if not clauseNormal($g$) then
    $U$ := $U \cup$ cnf($g$)
  else if empty($g$) then
    return success; initial $U$ is unsatisfiable
  else if choiceSpec($g$) then
    registerChoiceFunction($g$)
  else
    if not redundant($g$,$P$) then
      $P$ := $P \setminus \{p \in P \mid$ redundant$(p, \{g\})\}$
      submit($P \cup \{g\}$)
      $T$ := $\{p \in P \mid$ simp($p, \{g\}$) $\neq p\}$
      $P$ := $(P \setminus T) \cup \{g\}$
      $T$ := $T \cup$ generate($g$,$P$)
      $T$ := $\bigcup_{t \in T}$ cheapSimp(cnf(unify($t$)), $P$)
      $U$ := $U \cup \{t \in T \mid$not trivial($t$)$\}$
    endif
  endif
end
return failure;
```

was able to establish the unsatisfiability of the submitted clause set. Fresh proof obligations for external reasoners are heuristically produced and submitted (line 17). Checking for external results as well as submitting fresh proof obligations is non-blocking and therefore does not stall the internal saturation loop.

The functions used within the saturation procedure are:

selectBest(U) heuristically selects a clause from U and removes it from this set.

`simp(`g`,`P`)` exhaustively applies all normalization rules to g where potential side premises come from P. This includes the rules (DD), (DR), (RW), (PSR) and (NSR). Note that g is the only clause that is being modified during this operation.

`cheapSimp(`T`,`P`)` similarly applies a subset of normalization rules that are implemented particularly efficiently, including (DD), (DR) and (RW).

`clauseNormal(`g`)` returns `true` if and only if g is in clause normal form (i.e. cannot be further normalized with respect to $\mathcal{CNF}$).

`empty(`g`)` returns `true` if and only if $g \equiv \Box$ (recall that g may also consists only of flex-flex unification literals).

`choiceSpec(`g`)` returns `true` if and only if g is a specification of a choice operator, i.e. if rule (ACD) can be applied.

`registerChoiceFunction(`g`)` adds the specified choice operator represented by g to the set of registered choice functions `CF` as described in §4.2.

`redundant(`g`,`P`)` returns `true` if and only if g can be shown redundant with respect to P using (CS), (TD1) and (TD2).

`trivial(`t`)` likewise returns `true` if and only if g can be shown to be tautological using (TD1) and (TD2). Similar to `cheapSimp`, `trivial` is a specialized variant of `redundant` that uses only a efficiently implemented subset of the redundancy detection rules. In particular, it is independent on the size of P.

`generate(`g`,`P`)` applies all generating inferences where g is the main premise and potential other premises come from P. This includes the rules (Para), (Fac), (Prim), (PBE), (NBE), (PFE) and (NFE). Additionally, if enabled, rules (LEQ), (AEQ), (ACI) and (FuncSpec) are applied.

`unify(`t`)` applies unification rules to t. If unification literals are recognized as higher-order patterns, pattern unification is employed. Bounded higher-order pre-unification is applied if the unification literals are non-patterns.

`cnf(`T`)` exhaustively applies all clausification rules from $\mathcal{CNF}$ to every clause $t \in T$ and return the union of all results.

`checkExternal()` returns `true` if any previously submitted external prover call terminated with a helpful result, i.e. if the unsatisfiability of the submitted clause set was established. This operation is non-blocking, hence it returns `false` if there is no such answer available yet.

`submit(`T`)` requests the invocation of every registered external reasoning system on the clause set T. The maximal number of parallel externals calls is limited (and configurable by the user) and hence `submit` might simply do nothing if that limit is exceeded or a call on T is otherwise deemed unnecessary (cf. §4.4). Depending on the characteristics of the systems, the actual transmitted proof obligation might differ from T. For example, it may be necessary to encode T to a first-order proof obligation, to eliminate polymorphic symbols from the problem or to explicitly axiomatize THF language constructs that are not supported within the target prover's language.

There are several differences between the procedure in Fig. 4 and first-order DISCOUNT loop variants. Most of these differences are due to two main characteristics of higher-order saturation: Firstly, since formulas can occur within subterm positions and, in particular, within proper equalities, many of the generating and modifying inferences may produce non-CNF clauses albeit having proper clauses as premises. This implies that, during a proof loop iteration, potentially every clause needs to be re-normalized after almost every application of inferences. Secondly, since higher-order unification is undecidable, unification procedures cannot be used as an eager inference filtering mechanism (e.g., for paramodulation and factoring) nor can they be integrated tightly into the inference generation operations as done in first-order procedures. Even if higher-unification was decidable, it could not be used as a filtering mechanism for the generating inferences. This is due to the fact that if a unification constraint cannot be solved syntactically, it might still be solvable semantically modulo extensionality (which is also undecidable).

These aspects are reflected within Leo-III's saturation loop in several locations. At every iteration of the loop the freshly generated clauses potentially need to be re-clausified before being inserted into U (line 21) since rules such as (Prim), (Para), (Fac), (PBE) and (NBE) may generate non-proper clauses. Also, the simplification procedures (including rewriting) may introduce non-proper clauses (line 7). Even if a clause $\mathcal{C}$ was simplified before being inserted into U, it may be the case that further rewrite rules (between formulas) have been generated before $\mathcal{C}$ was selected by `selectBest` and hence rewriting, again, may require re-normalization of the result with respect to $\mathcal{CNF}$ (lines 8,9).

Unification is included into the proof loop as an explicit procedure, detached from the application of the generating inferences (line 21). If unification constraints that were introduced to freshly created clauses by `generate` are higher-order patterns, pattern unification is used to solve the constraint. Otherwise, a higher-order pre-unification procedure is used. As opposed to the first-order case, clauses that contain unification literals that could not be solved are not discarded but nevertheless inserted into the search space. This is necessary in order to retain completeness. However, as discussed below, certain search strategies might change this behavior for practical considerations.

The saturation loop does not necessarily need to find the empty clause as defined in first-order theorem proving, but rather a clause which contains only negative flex-flex literals (cf. §3.1). Given a clause $\mathcal{C}$ of the form

$$\mathcal{C} \equiv [X^1\, \bar{s}^1 \simeq Y^1\, \bar{t}^1]^{\mathrm{ff}} \vee \ldots \vee [X^n\, \bar{s}^n \simeq Y^n\, \bar{t}^n]^{\mathrm{ff}}$$

where the X^i and Y^i are free variables, it is known that there always exists a substitution σ such that $\mathcal{C}\sigma$ only contains solved unification constraints [Hue72, Hue73a] and hence can be simplified to $\square$.

4.3.3. Search Control

Even though the saturation procedure presented in the previous section brings a certain amount of structure to the proof search process, there are several parameters to the saturation loop that have significant influence on the success rate of the reasoning system. It is well-known that clause selection, i.e. the process of selecting the next clause to be used by the saturation loop, is one of the most crucial points [Sch02]. It is not surprising that major effort has been invested in enhancing and improving clause selection techniques, involving sophisticated multi-priority selection strategies [Sch02], syntax-guided heuristics [SM16] and, recently, the employment of machine learning and AI related techniques for generating good selection methods [FB16, LISK17, JU18]. Besides clause selection, there are further parameters that influence the effectivity of the ATP system: Relevance pruning is used the eliminate irrelevant facts prior to proof search [MP09, BGK$^+$16, PU18], literal selection can be used to restrict the number of inferences generated during a single proof loop iteration, and sophisticated redundancy detection methods try to reduce the search space during saturation [Sch02].

In a higher-order setting, there are even more aspects that need to be handled for practical applications (cf. challenges from §4.1). Higher-order inferences such as paramodulation and factoring exhaustively generate all possible conclusions from its premises without any effective syntactical restrictions. In first-order theorem proving, unification procedures can be used as a filter routine for such inferences. Furthermore, some inference rules may generate infinitely many conclusions. We hence need to find suitable strategies to prevent an explosive growth of the search space while, at the same time, retaining reasoning effectivity.

In Leo-III, adjustment of search control parameters can be done quite flexibly due to the strict separation of the saturation loop, the inference rule applications, and their control.

Clause selection. As pointed out by Schulz [Sch02], clause selection is of particular importance for theorem proving systems built around variants of the DISCOUNT loop. This is due to the fact that unprocessed clauses are truly passive and do not contribute to the proof search at all until selected. Clause selection in Leo-III is inspired by E's usage of multiple priority queues aligned in a weighted round-robin scheme [Sch02]. In contrast to clause selection in E, clause selection functions in Leo-III consist of arbitrary lexicographical combinations of primitive clause orders. Primitive clause orders roughly correspond to the notion of evaluation functions from E. Furthermore, the use of explicit priority functions by E which prefer certain classes of clauses is subsumed by the above approach since priority functions can be seen as special primitive clause orders. A corresponding prioritization of clauses is then implemented by using the respective clause orders as first element in the lexicographical composition of the clause selection function. Of course, each clause selection function itself defines an ordering on clauses that can again be used for constructing more complex selection functions. This is, however, currently not exploited by Leo-III. The most important clause order primitives available in Leo-III are:

Age: Leo-III assigns each clause a unique monotonously increasing number $\mathrm{id}(\mathcal{C})$ to each generated clause $\mathcal{C}$. This information can be used to define a clause ordering that simply prefers older clauses to newer ones: Let $\mathcal{C} \lesssim_{\text{age}} \mathcal{D}$ whenever $\mathrm{id}(\mathcal{C}) \leq \mathrm{id}(\mathcal{D})$.

Literal count: Intuitively, smaller clauses, that is clauses with fewer literals, should be preferred as the proof search process is aimed to infer the empty clause $\square$ which is the smallest possible clause. Furthermore, smaller clauses generate fewer consequences and, if only containing one literal, can be used for unit simplification procedures. To that end, let $\mathcal{C} \lesssim_{\text{litCount}} \mathcal{D}$ if and only if $|\mathcal{C}| \leq |\mathcal{D}|$.

SOS: The set of support search strategy was developed in the 1960s by Wos et al. [WRC65] for improving the performance of saturation procedures in first-order theorem proving. Informally, the set of support (SOS) is a set of clauses that contains every clause which is derived from inferences where at least one premise is contained in the SOS. Initially, only the conjecture is in the set of support. The SOS search strategy then restricts proof search to only allow inferences where at least one clause is contained in the set of support. In order to simulate this behavior, Leo-IIIs offers the clause ordering $\lesssim_{\text{SOS}}$ where $\mathcal{C} \lesssim_{\text{SOS}} \mathcal{D}$ if and only if $\mathcal{C}$ is contained in the SOS. $\sim_{\text{SOS}}$ effectively partitions the set U in two distinct subsets.

Symbol weight: Symbol counting can be used to prefer clauses with fewer symbols. This is due to the fact that clauses with fewer symbols have fewer positions and hence generate fewer consequences. In E, variables, function symbols and predicate symbols can be assigned different weights which are then added and result in the weight of the clause. In HOL, there is no strict separation between formulas and terms, hence in Leo-III there are only two parameters to this ordering: A weight for non-variable symbols and a weight for variables. Formally, for a symbol weight order weight$(w_{\text{var}}, w_{\text{fun}})$ it holds that $\mathcal{C} \lesssim_{\text{weight}(w_{\text{var}},w_{\text{fun}})} \mathcal{D}$ if and only if $w(\mathcal{C}) \leq w(\mathcal{D})$, where

$$\begin{aligned}
w(\mathcal{C}) &= \Sigma_{\ell \in \mathcal{C}} w(\ell) \\
w([s \simeq t]^{\alpha}) &= w(s) + w(t) \\
w(\lambda X.t) &= w(t) \\
w(s\,t) &= w(s) + w(t) \\
w(X) &= w_{\text{var}} \quad \text{if } X \text{ is a variable} \\
w(c) &= w_{\text{fun}} \quad \text{if } c \in \Sigma \text{ is a constant symbol.}
\end{aligned}$$

Conjecture relative symbol weight: Similar to symbol weight, conjecture relative symbol weight prefers clauses with fewer symbols. Additionally, $\lesssim_{\text{conjRelWeight}(w_{\text{var}},w_{\text{fun}},c)}$ takes a real multipler $c > 0$ which is multiplied to the weight of function symbols that also occur in the conjecture. Usually, c is chosen such that $0 < c < 1$ in order to prefer clauses in which relevant symbols from the conjecture occur. This order is a simplified variant of Schulz' *RefinedWeight* as implemented in E [Sch02].

Prefer goals: Negative literals can be regarded as goals to be solved e.g. by unification. $\lesssim_{\text{goals}}$ prefers goals by having $\mathcal{C} \lesssim_{\text{goals}} \mathcal{D}$ whenever $1 - \frac{|\mathcal{C}^-|}{|\mathcal{C}|} \leq 1 - \frac{|\mathcal{D}^-|}{|\mathcal{D}|}$ for non-empty clauses $\mathcal{C}$ and $\mathcal{D}$. It is extended to empty clauses by having $\square \lesssim_{\text{goals}} \mathcal{C}$, for every clause $\mathcal{C}$.

Prefer non-goals: Analogously, $\lesssim_{\text{nonGoals}}$ prefers goals by having $\mathcal{C} \lesssim_{\text{nonGoals}} \mathcal{D}$ whenever $1 - \frac{|\mathcal{C}^+|}{|\mathcal{C}|} \leq 1 - \frac{|\mathcal{D}^+|}{|\mathcal{D}|}$ for non-empty clauses $\mathcal{C}$ and $\mathcal{D}$. Again, empty clauses are always smaller.

Groundness: Ground clauses may be utilized by simplification methods and tend to generate fewer consequences. To that end, have $\mathcal{C} \lesssim_{\text{ground}} \mathcal{D}$ whenever $\mathcal{C}$ is ground.

The last three clause orders correspond to frequently used priority functions within E. Further clause orders can easily be added to Leo-III. Formally, the above primitive clause orders correspond to total preorders (or preference relations). Every total preorder $\lesssim$ defines an equivalence relation $\sim$ with $\mathcal{C} \sim \mathcal{D}$ whenever $\mathcal{C} \lesssim \mathcal{D}$ and $\mathcal{D} \lesssim \mathcal{C}$. Equivalent clauses $\mathcal{C} \sim \mathcal{D}$ can be seen are identical with respect to the clause order $\lesssim$ (e.g. having the same number of literals etc.). By convention, a clause $\mathcal{C}$ is strictly smaller than $\mathcal{D}$ with respect to a given clause ordering $\lesssim$, denoted $\mathcal{C} < \mathcal{D}$, whenever $\mathcal{C} \lesssim \mathcal{D}$ but not $\mathcal{C} \sim \mathcal{D}$.

Concrete clause selection functions are then constructed by lexicographical combinations of the above clause order primitives in Leo-III. For some index set I and $i_j \in I$ for $1 \leq j \leq n$, $n \geq 1$, a lexicographical combination $(\lesssim_{i_1}, \dots, \lesssim_{i_n})$ again defines a (strict) total preorder $<_{(\lesssim_{i_1},\dots,\lesssim_{i_n})}$ given by

$$\mathcal{C} <_{(\lesssim_{i_1},\dots,\lesssim_{i_n})} \mathcal{D} \text{ if and only if either } \mathcal{C} <_{i_1} \mathcal{D} \text{ or } \mathcal{C} \sim_{i_1} \mathcal{D} \text{ and } \mathcal{C} <_{(\lesssim_{i_2},\dots,\lesssim_{i_n})} \mathcal{D}$$

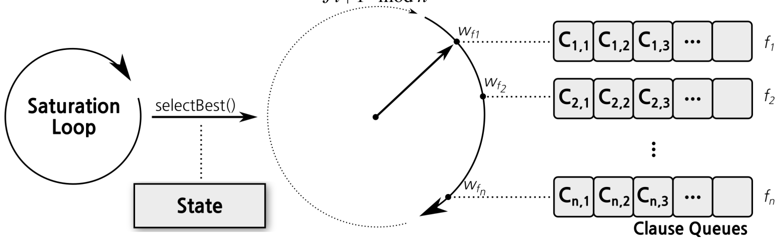
For the border case $<_{(\lesssim_{i_n})}$, let $\mathcal{C} <_{(\lesssim_{i_n})} \mathcal{D}$ if and only if $\mathcal{C} <_{i_n} \mathcal{D}$. Currently implemented clause selection functions within Leo-III include:[6]

- $(\lesssim_{\text{litCount}}, \lesssim_{\text{conjRelWeight}(.)})$: Prefer small clauses that also carry few symbols or symbols contained in the conjecture.
- $(\lesssim_{\text{goals}}, \lesssim_{\text{weight}(.)})$: Prefer clauses that have many negative literals and carry few symbols.
- $(\lesssim_{\text{nonGoals}}, \lesssim_{\text{weight}(.)})$: Prefer clauses that have many positive literals and carry few symbols.
- $(\lesssim_{\text{SOS}}, \lesssim_{\text{conjRelWeight}(.)})$: Prefer clauses that originate from inferences with the conjecture (the set of support) and additionally carry few symbols or symbols contained in the conjecture.
- $(\lesssim_{\text{age}})$: Prefer early clauses unconditionally.

These clause selection functions are mostly inspired by work of Schulz and others in the context of E [SM16, SS15]. It is easy to see that the selection process is fair if $\lesssim_{\text{age}}$ is included into the portfolio, as every clause will be eventually selected. The exact combinations of clause orderings originate from experimentation with Leo-III of a problem set from CASC-J8 [Sut16a]. There is certainly a lot of room for improvement, in particular for identifying clause orders involving higher-order features such as order of argument's types, nesting of formulas,

[6] Every clause selection function in Leo-III uses $\lesssim_{\text{age}}$ as last, implicit, primitive clause ordering to break ties in favor of older clauses. Since clause ages are unique, every resulting clause selection function is total and strict.

Figure 5: Round-robin clause selection within Leo-III. The $C_{i,j}$ are the unprocessed clauses from U ordered with respect to clause selection function f_i. Up to w_{f_i} clauses are selected from the same queue until `selectBest()` advances to the next clause selection function $f_{i+1 \mod n}$.


Saturation Loop
selectBest()
State
w_{f1}
w_{f_2}
w_{f_n}
$C_{1,1}$ $C_{1,2}$ $C_{1,3}$ ···
$C_{2,1}$ $C_{2,2}$ $C_{2,3}$ ···
$C_{n,1}$ $C_{n,2}$ $C_{n,3}$ ···
f_1
f_2
f_n
Clause Queues

variables at head position etc. A structured approach for generating and evaluating improved selection functions remains further work; cf. current discussions about this topic [Sch17].

A weighted clause selection function is then a pair (f, w_f) where f is a clause selection function and $w_f \in \mathbb{N}$ is a weight. The weights are used by Leo-III to prefer certain selection functions in a round-robin selection procedure. Figure 5 displays a schematic presentation of Leo-III's clause selection procedure. Each invocation of `selectBest()` selects the smallest clause of U with respect to the current weighted clause selection function, removes this clause from U and returns the result. If n selection functions are used, the selection process works as follows: If (f_i, w_{f_i}) is the current selection function, w_{f_i} successive calls to `selectBest()` use f_i for selecting the next clause. After w_{f_i} calls, the next call to `selectBest()` uses $f_{i+1 \mod n}$ for $w_{f_{i+1 \mod n}}$ selections. This process is continued until the empty clause is found or the reasoning process is otherwise terminated.

Premise selection. Premise selection is increasingly relevant for reasoning in practical relevant applications, e.g. in mathematical reasoning and reasoning in large knowledge databases. For many small and medium sized problems, even relatively simple approaches can already significantly increase the reasoning performance. The idea of relevance goes back to Robinson who identified pure literals to be of no avail in a first-order resolution reasoning process [Rob65]. Here, premise selection focuses on the selection of axioms prior to proof search.

Leo-III implements a relevance filtering mechanism due to Meng and Paulson [MP09] which produces fair improvements for small and medium sized problems. Inclusion of an higher-order adaption of a SInE selection routine [HV11] is ongoing work. The latter approach seems to be promising as observed by the major improvement in the performance of Satallax 3.2 that is, at least partly, due to a SInE-like axiom selection [Sut17a].

Restriction of inferences. Leo-III uses several heuristics to restrict the number of inferences, including a higher-order term ordering called *computability path ordering* (CPO) [BJR15]. While these restrictions sacrifice completeness in general, an evaluation in §6 confirms that these approaches are indeed practical.

The restrictions implemented in Leo-III are:

- As pointed out in §4.1, unification cannot be used as filtering procedure for paramodulation and factoring inferences. If however an unrestricted application of such rules is allowed, every type-correct combination of subterms from every literal of all premises generates a conclusion, leading to an tremendous explosion of the search space. In contrast, LEO-II is based on resolution and hence only allows top-level combinations of literals which results in a much smaller set of conclusions.
  To mitigate this situation, Leo-III uses a filtering procedure that can be regarded an under-approximation of unifiability. Intuitively, this filter predicate permits paramodulation and factoring between two terms if the deterministic unification rules (Decomp), (Bind) or (Triv) can be applied at least once to the resulting unification literal or either side contains a variable at head position. This restriction however sacrifices completeness of Leo-III but seems to be reasonable for practical applications as later evaluations indicate. The filtering mechanism can be disabled by the user using a command-line parameter.
- In order to guarantee termination of higher-order pre-unification, Leo-III limits the search depth of the unifier enumeration procedure. The depth of a node in the search graph is hereby defined as the number of subsequent applications of imitation and projection bindings on the current search branch. It turns out that a reasonable small search depth of eight suffices for almost all unification tasks originating from TPTP problems. Of course, this parameter may be changed by the user if required. Since Leo-III implements pattern unification, it can additionally be expected that plain (pre-)unification is only used in comparatively rare cases.
  The implementation of the Huet's higher-order unification procedure employed by Leo-III is originally due to Tomer Libal. It has been substantially reworked and reimplemented in order to guarantee termination and support polymorphic HOL by the author. In particular, the imple-

Table 1: Intensity levels of primitive substitution (Prim) within Leo-III. For every intensity level i, the set S^i of symbols serves as carrier set for the approximating bindings $\mathcal{GB}_\tau^{S^i}$ used by (Prim), where $\tau \in \mathcal{T}$ is the goal type for the binding.

Level i	**Description**	**Symbols** S^i
0	Primitive substitution disabled	$\emptyset$
1	Logical connectives	$\{\neg, \vee, \top, \bot\}$
2	Further connectives	$\{=^\mu, \neq^\mu \mid \tau \equiv o\mu\mu \text{ for some } \mu \in \mathcal{T}\}$
3	Problem-specific symbols	$\{c \in \Sigma \mid \text{goal}(\text{ty}(c)) \equiv o\}$
4	Locally bound variables	$\{X \in \text{fv}(\mathcal{C}) \mid \text{goal}(\text{ty}(X)) \equiv o\}$

mentation of a single unification procedure call returns an infinite lazy stream of unifiers that can be stored and re-used later on if necessary, e.g., for iterative deepening approaches. This provides the possibility of eliminating or at least mitigating the incompleteness source due to the search depth limit; but is not yet exploited by Leo-III. Additionally, it is planned to augment Leo-III's pattern unification procedure with specialized treatment of regular patterns [Lib15].

- Leo-III classifies several *intensity levels* for applications of primitive substitution by rule (Prim). As described further above, exhaustive application of (Prim) is infinitely branching and leads to a search space explosion if not restricted. The intensity levels of Leo-III allow to control the concrete connectives and constant symbols that are used to construct general bindings by (Prim). Table 1 displays the intensity levels of Leo-III and their associated sets of constant symbols used for primitive substitution. Leo-III is configured to use the set $S := \bigcup_{j<i} S_j$ for a given intensity level i which is, by default, set to one. The exact set of symbols used for primitive substitution be configured more fine-grained by the user if necessary.
- Leo-III implements a variant of the higher-order term order CPO [BJR15]. The induced ordering is used to orient the sides of equational literals as well as determine the maximal literals of a clause. This information can be used by Leo-III to select only a subset of all literals (and their sides) to be eligible for paramodulation and factoring inferences. Since CPO originates from research in higher-order termination, it is not clear whether the properties of the ordering actually suffice for being used for some variant of a higher-order superposition equivalent and hence no completeness guarantees have been established yet. As for the previously discussed restrictions, the conducted evaluations seems

however to confirm the practical benefits of the orderings' employment.

Extensionality and unification. As already sketched in §4.1, the difference between syntactical unification and semantic proof search disappears in extensional HOL. In principle, every invocation of a unification procedure could be considered to be a separate new proving attempt that unfolds to an independent proof search for syntactically or semantically discharging the unification constraint. Since a complete proof search procedure is anything but an efficient subroutine for unification, such a solution is far from practical.

Leo-III tackles this problem by interleaving unification and extensionality inferences at the proof search level, preferring syntactical unification whenever possible. This is in contrast to LEO-II where extensionality inferences are hard-wired into its monolithic and fairly complex unification procedure. In Leo-III extensionality inferences and unification inferences are strictly separated routines. There are multiple benefits of approach:

- The unification procedure is simpler, faster and more maintainable.
- Application of extensionality inference rules can be explicitly controlled in a fine-grained fashion.
- Extensionality principles can even be completely disabled, effectively turning Leo-III into an ATP system for elementary type theory (ETT). This can be handy e.g. for experiments where no extensionality inferences are desired or known to be unnecessary.

The interplay between unification and extensionality is tackled as follows in Leo-III: Every occurrence of equivalence between formulas is replaced by proper equality during parsing (if not disabled for experiments in ETT or similar fragments). Hence, equivalence is regarded as a defined notion (or syntactic sugar) instead of a language primitive. First, unification (or pattern unification) is applied to every negated equality literal, regardless if the equality is between formulas or non-Boolean terms. If unification fails, a dedicated routine that combines the deterministic unification rules (Triv), (Bind) and (Decomp) is exhaustively applied to the clause. The resulting clause is then eligible for extensionality inferences and the potential results and added into the search space for further processing. This way, the unification procedure is applied as far as possible avoiding early search space explosions caused by excessively generating extensionality inferences.

Strategy scheduling. As discussed further above, Leo-III uses various heuristic restrictions that are intended to increase the reasoning effectivity. Most of these restrictions can be adjusted using various parameters, where each combination of such parameter settings may have a strongly different behavior on a given problem or problem set. An abstraction of the uniform search approach that uses a fixed parameter setting is the notion of a *search strategy* and the employment of *strategy scheduling* [RV03, Wol98]. A search strategy can be seen as a concrete combination of search parameter settings, usually including a time limit that defines the time the ATP system will spend executing proof search using that particular search parameters. Strategy scheduling means here the execution of a portfolio of different search strategies, in parallel or sequentially, on a single input problem. The first strategy, if any, to solve the problem then contributes the overall solution of the ATP system.

In the context of Leo-III, strategy scheduling allows to use both incomplete but often effective search strategies and complete but often ineffective approaches to yield a combination that yields a complete and hopefully effective overall search approach. As an example, such a complete fallback strategy could disable all above presented search restrictions such as ordering constraints. Leo-III currently implements a simple variant of strategy scheduling that allows a sequential execution of different search strategies during a single invocation. However, the support for strategies in Leo-III is still rudimentary and requires more development work in order to result in a robust benefit to its reasoning effectivity.

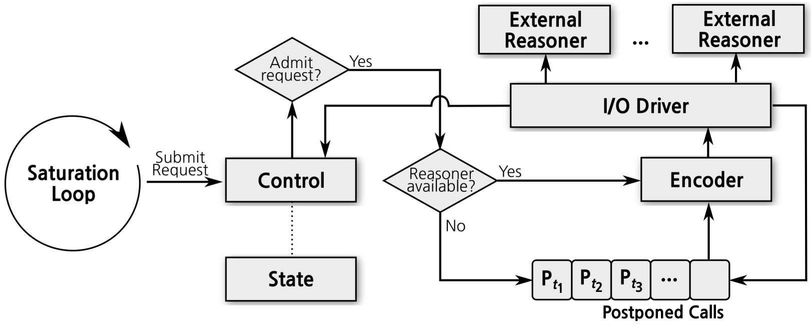
4.4. External Cooperation

In the tradition of the cooperative nature of the LEO prover family, Leo-III collaborates during proof search with external reasoning systems, in particular, with first-order ATPs such as E [Sch02], iProver [Kor08] and Vampire [RV02] as well as SMT solvers like CVC4 [B$^+$11]. Unlike LEO-II, which translates proof obligations into untyped first-order languages, Leo-III, by default, translates its higher-order clauses to (polymorphic or monomorphic) many-sorted first-order formulas.

Leo-III accumulates higher-order clauses during proof search and repeatedly invokes all cooperating systems on the generated proof obligations. For that purpose, various translation mechanisms from HOL to different variants of first-order logic are implemented. If any external (first-order) reasoning system finds the submitted proof obligation to be unsatisfiable, the original HOL problem is unsatisfiable as well and a proof for the original conjecture is found.

The integration of external cooperation into the reasoning process of Leo-III is described in §4.4.1. The translation techniques that are used for cooperating with first-order systems are sketched in §4.4.2. An evaluation of the effect of external cooperation within Leo-III is given in §6.1.

Figure 6: Invocation of external reasoning systems during proof search. The solid arrows denote data flow through the respective modules of Leo-III. A dotted line indicates use of auxiliary information. Postponed calls are selected by the I/O driver after termination of outstanding external calls.



4.4.1. *Utilization during Proof Search*

Leo-III's saturation procedure may, during any loop iteration, invoke external reasoning systems for discharging proof obligations that originate from its current search space state. To that end, Leo-III includes an encoding module that translates its higher-order clauses to polymorphic and monomorphic typed first-order clauses. While LEO-II relied on cooperation with untyped first-order provers, Leo-III exploits the relatively young support of types in first-order ATP systems using the associated TPTP language dialect TFF. By making use of TFF's type system, the translation of higher-order proof obligations does not require encoding types as terms, e.g. by type guards or type tags [BBPS16]. This approach reduces clutter and hence promises more effective cooperation.

Cooperation within Leo-III is by no means limited to first-order ATPs. Multiple different systems, including first-order and higher-order ATPs and model finders, can be used simultaneously provided that they comply with some common TPTP language standard. Currently, Leo-III supports encoding and output of proof obligations to four different TPTP dialects:[7]

1. TF0: Monomorphically typed first-order logic [SSCB12]
2. TF1: Polymorphically typed first-order logic [BP13b]
3. TH0: Monomorphically typed higher-order logic [SB10]

[7] Note that the TPTP TF0 (TH0) language has generally been referred to as TFF (THF) prior to the development of its extension to polymorphism. Sometimes TF0 (TH0) is also called TFF0 (THF0).

4. TH1: Polymorphically typed higher-order logic [KSR16]

Figure 6 displays a schematic diagram of the interaction between Leo-III and external reasoning systems. During the saturation, the control of Leo-III is periodically requested to submit a new external reasoning task. The generated proof obligations originate from the current set of processed clauses P. The decision of whether the control will admit the cooperation request is a predicate involving the current workload of the external systems, the current time, the time of the last admitted call and a base frequency that can be specified by the user.[8] This predicate is designed in such a way that, for each employed external reasoner, a cooperation request is always admitted if the system would otherwise be idle or the last admitted request was made long enough in past (with respect to loop iteration count or actual time). Note that an admitted cooperation request does not necessarily lead to an immediate invocation of the respective system. Leo-III limits the number of parallel open requests (that is requests where the external reasoner is still running) per registered cooperating system to some number in order to avoid swamping the CPU with hundreds of parallel processes. As a consequence, if the current limit is already reached, admitted cooperation requests are inserted into a collection of postponed requests. If an external call is completed, some element of this collection is sent immediately without further requests from the saturation loop. The selection strategy of postponed requests balances between picking most recent requests and those which have been delayed for the longest time. This approach tries to mediate between situations where early (small) clause sets are easier to solve for external reasoners and situations where difficult higher-order reasoning steps are necessary for enabling the external systems to establish unsatisfiability.

4.4.2. Translation to First-Order Logic

In Leo-III, translation of higher-order clauses to first-order logic consists of various independent steps that can be flexibly combined and configured. As a first step, regardless of the target logic, language features that are not supported by the cooperating system are eliminated from the problem. Then, proof obligations are converted into polymorphically typed first-order formulas by eliminating all higher-order constructs such as anonymous functions, partial applications, variables at function positions and higher-order types [MP08, BBPS16]. Subsequently, these polymorphic first-order formulas are translated to monomorphic

[8] By default, every 15th loop iteration will trigger an external cooperation call (assuming that it has not been triggered by further conditions).

first-order logic by heuristic monomorphization [Böh12, BBPS16]. The resulting typed first-order problems may be reduced to untyped first-order logic using appropriate type encodings [BBPS16] as well. Leo-III however refrains from this effort since many of the usual cooperating first-order provers meanwhile natively support monomorphic types.

Of course, each intermediate result of the sketched translation pipeline may already be given to an appropriate external reasoning systems. As an example, an obvious approach would be to use first-order ATP systems that already support polymorphic types. This way the expensive and incomplete monomorphization routine can be eliminated from the translation process. However, experiments with relevant provers indicate that currently available first-order ATP systems for TF1 are still less effective for cooperation than using sophisticated TF0 ATPs in conjunction with manual monomorphization.

Similarly, there are multiple techniques for eliminating anonymous functions (λ-abstractions) from the problem. It is not generally clear what approach, if any, is superior in practice. On the contrary, an evaluation in the context of Isabelle's Sledgehammer tool suggests that all approaches have reasonable advantages and disadvantages, strongly depending on what cooperating system is used [SBP13].

Each translation step used by Leo-III for interacting with first-order ATPs is briefly outlined next. An evaluation of different cooperating ATP systems as well as different encoding techniques employed by Leo-III is presented in §6.1.

Elimination of unsupported language features. The TPTP THF (TH0) syntax offers several beneficial syntax features that allow a more concise formulation of problem statements. Unfortunately, some of these language features are not widely accepted or known in the respective community and are therefore only supported by a few ATP systems. These include[9]

- *if-then-else* conditionals given by formulas of the form `$ite_f(p,f1,f2)`,
- *let-bindings* given by formulas and terms of the form `$let_tf(local_defs,f1,f2)` and `$let_ff(local_defs,t1,t2)`, respectively, and in the context of higher-order cooperating systems
- *choice and description operators* given by terms `@+ [X:ty]: t` and `@- [X:ty]: t`, respectively, and

[9] Note that some TPTP languages currently undergo some modifications for simplification and uniformity. This, in particular, includes the use of the combinator connectives (e.g. `!!` etc.) in THF. It is possible that the presented syntax examples are in this context not syntactically or semantically valid anymore.

- *combinator variants* of usual connectives such as universal quantification, choice and equality, given by `!! @ t`, `@@+ @ t`, and `@= @ t`, respectively.

Leo-III comes with a data base of known reasoning systems and their capabilities [SWSB17] and decides per employed cooperating system whether specific language features in the problem to be submitted need to be eliminated. In the case that if-then-else conditionals are not supported, an appropriate axiomatization for a freshly introduced conditional predicate is included in the problem. Local let-bindings are simply unfolded if not recognized by the target system. For choice and description terms, an associated fresh operator is postulated together with an higher-order specification of that operator. If the target system is first-order, this axiom is then converted to a first-order formula by the usual translation process. Leo-III does not distinguish internally between prefix applications of connectives and the standard syntax. After parsing and interpretation (cf. §4.1), these cases are handled uniformly by the following translation procedures.

Elimination of higher-order features. Additional to the eponymous use of higher-order quantification, there are several further language constructs in HOL that cannot be directly expressed in first-order logic. This includes

1. *higher-order quantification* as in $\forall P_{o\tau}. P\, a \longrightarrow P\, b$,
2. *higher-order arguments* that can be passed as function arguments, e.g. apply $f_{\nu\tau}\, X_\tau$,
3. *variables at head position*, i.e. terms of the form $X\, \overline{s^i}$ for some variable X,
4. *partially applied functions* that can be used as proper terms, e.g. as $f = apply\, f$,
5. *formulas within terms* as in $f_{\iota o}\, (s_o \longleftrightarrow t_o)$,
6. *terms as formulas*, i.e. terms in which some terms t_o serve as formula as well,
   e.g. $s_o \leftrightarrow (s_o = t_o)$, and
7. *anonymous functions* by λ-abstraction as in $\lambda X_\tau. \lambda Y_\tau. f\, X$.

All of the above higher-order features, except for occurrences of anonymous functions, can be eliminated by using two additional operators hApp and hBool[10] that re-introduce the strict separation of formulas and terms [MP08,

[10] The names of the operators are chosen here to coincide with the actually implemented translation in Leo-III. This translation, in turn, follows Meng and Paulsons naming convention within the

Rey98a]. hApp is a binary polymorphic first-order logic function symbol of type $\mathrm{fun}(\alpha,\beta) \times \alpha \to \beta$ where fun is a new sort symbol of arity two, $\mathrm{fun}(\alpha,\beta)$ representing the type of functions from type α to type β. hBool is a predicate symbol of type $\mathrm{bool} \to o$ where bool is a new type reflecting terms of Boolean type (i.e. terms that are used as formulas).

Intuitively, hApp serves as explicit application operator such that terms $s\, t^1 \, \cdots \, t^n$ are translated to $\mathsf{hApp}(\ldots(\mathsf{hApp}(\dot{s},\dot{t}^1),\ldots),\dot{t}^n)$ where $\dot{s}$ and the $\dot{t}^i$ denote the translations of s and the t^i, respectively. Hence, function application is encoded by successive applications of hApp where its first argument is an encoded function symbol of type $\mathrm{fun}(\tau,\nu)$, for some types τ,ν, and the second argument of type τ is the argument to the passed function. Analogously, higher-order function types are represented by appropriate left-nesting, e.g. $\mathrm{fun}(\mathrm{fun}(\tau,\nu),\mu)$ represents the higher-order type $\mu(\nu\tau)$.

Higher-order terms s_o of Boolean type that occur as proper subterms are encoded as terms $\dot{s}_{\mathrm{bool}}$ of a dedicated type bool which is distinct from o [JP09]. If such a term is then used as (top-level) formula, the meta-predicate hBool converts it to a first-order formula $\mathsf{hBool}(\dot{s})$. Within Boolean typed subterms, logical connectives are replaced by so-called *proxy symbols* [Böh12]. Additional axioms relate these proxy symbols with the properties of the underlying connective.

When disregarding anonymous functions, the above presented techniques already convert a HOL problem to a polymorphically typed first-order problem. However, in order to allow higher-order equivalent reasoning within the first-order reasoner some more information needs to be included [Ker94]. An example of built-in reasoning principles of HOL is functional and Boolean extensionality. Extensionality axioms need to be stated explicitly for each constant symbol in the context of first-order logic. In order to enable full higher-order reasoning, appropriate extensionality axioms need to be added to the resulting problem. Similarly, comprehension principles require additional explicit axioms [Ker94].

Leo-III does not include any of these axioms related to higher-order reasoning principles into the translated problem. This is due to the fact that the idea underlying Leo-III is that higher-order reasoning is done by Leo-III internally, whereas generated (first-order) consequences are sent to external provers for discharging. Also, rigorously augmenting a problem with such axioms dramatically increases its size and may potentially reduce the effectivity of first-order reasoners.

implementation of Sledgehammer. hApp is also often denoted @, whereas hBool is also referred to as B [MP08]. There are certainly many more naming variants originating from concrete translation implementations.

Elimination of anonymous functions. In HOL, functions can be constructed using λ-abstraction. These functions have no direct equivalent in FOL. There are at least two different popular techniques in use for eliminating anonymous functions:

1. Elimination of anonymous functions by *λ-lifting* [Joh85] proceeds by successively replacing λ-abstractions by fresh function symbols. The problem formulation is then augmented with appropriate definitions of the introduced function symbols, sometimes also referred to as *supercombinators*. This process is related to *defunctionalization* [Rey98a] and used in various applications in functional programming, programming languages and compiler theory.
2. Another possibility is to use a small but complete set of combinators and replace the anonymous functions by adequate applications of such combinator terms. The use of combinators has early roots and goes back to Schönfinkel and Curry [Sch24, Cur30, Bar81]. The use of Curry combinators may, however, increase the size of the problem quadratically [PJ87].

In addition to the two techniques, there exists an extended set of combinators due to Turner [Tur79] that aims at further decreasing the size of the resulting output. However, use of Turner combinators seems to be of no major improvement in practice [MP08].

Curry combinators are used by metis [Hur03], LEO-II relies on λ-lifting [BS13], and Isabelle's Sledgehammer system allows to configure which of the above elimination techniques (of a combination of both) is used and chooses heuristically if not specified by the user [SBP13]. Leo-III uses, by default, a translation scheme using Curry combinators but also supports λ-lifting. This setting can also be changed for each cooperating system.

Monomorphization. Since Leo-III is designed to cooperate with typed external reasoning systems, the translation of proof obligations does not include encoding of types, as done in LEO-II. It is possible, however currently not planned, to extend Leo-III to use encoding mechanisms for polymorphic and monomorphic types to reduce the polymorphically typed first-order problem to untyped first-order logic [BBPS16]. The proposed methods involve adding type arguments, type tags or type guards [MP08, BBPS16] to each formula and term of the problem in order to retain soundness. The number of necessary typing atoms can however be dramatically reduced using monotonicity inference mechanisms [CLS11, BBPS16]. These sophisticated encoding techniques are, for example, successfully used by Isabelle's Sledgehammer system [BBPS16].

Leo-III makes use of so-called *heuristic monomorphization* [Böh12] which generates a monomorphically typed problem from a polymorphic input problem. This approach is quite pragmatic: Universally quantified type variables are successively instantiated with heuristically chosen ground types and the results are merged into a single problem. It is strongly related to research in compilers for translations between different programming languages [TO98]. In the context of automated reasoning, monomorphization is known to be necessarily incomplete [CL07, BP11] and may increase the size of the problems representation by magnitudes [CL07] as the number of ground types eligible for instantiation is infinite.

Nevertheless, heuristic monomorphization can be used to exploit the native type support in external reasoning systems and proved to be of great value in practice [BBPS16, SBP13] for bridging between various logic formalisms used by different reasoning systems. Moreover, it enables provers to "detect unprovability of small problems much faster than before" [BBPS16] in the context of the Sledgehammer system. If the cooperating system supports polymorphic first-order logic, the monomorphization process is not required and will not be executed by Leo-III.

4.5. Proof Output

The output of Leo-III depends on whether or not the conjecture could be found to be a theorem. In the latter case, this might be due to one of several reasons: Either the conjecture is, de facto, not a theorem and hence it is not possible to find a sound derivation of the empty clause. Another possibility is that sources of incompleteness within Leo-III makes the saturation procedure terminate prematurely without establishing any validity result. Finally, the saturation procedure of Leo-III can diverge and, eventually, forcefully terminate if a given resource limit is exceeded. All of the above scenarios can also occur if no conjecture but only a collection of axioms is given in the input problem. For the result output format, Leo-III rigorously implements the machine-readable TSTP result standard [Sut10] and hence outputs appropriate SZS ontology values [Sut08]. The use of the TSTP output format allows for simple means of communication and exchange of reasoning results between different reasoning tools and, consequently, eases the employment of Leo-III within external tools (e.g., the SystemOnTPTP infrastructure [Sut17b]). Figure 7 displays a schematic template of Leo-III's output. The first 15 lines print statistical information about the proving attempt which are not captured by existing TSTP standards and are hence printed as comments. Here, t_Σ and $t_{\text{effective}}$ denote the total time and the time spent with-

Figure 7: Schematic template of Leo-III's proof output. All but the lines starting with `% SZS` denote additional information that is not captured by TSTP standards. The output enclosed in `<.>` are placeholders for the concrete values.

```
% Time passed: <tΣ>
% Effective reasoning time: <teffective>
% Solved by strategy<...>
% Axioms used in derivation (<nAx>): <axiom list>
% No. of inferences in proof: <...>
% No. of processed clauses: <...>
% No. of generated clauses: <...>
% No. of forward subsumed clauses: <...>
% No. of backward subsumed clauses: <...>
% No. of ground rewrite rules in store: <...>
% No. of non-ground rewrite rules in store: <...>
% No. of positive (non-rewrite) units in store: <...>
% No. of negative (non-rewrite) units in store: <...>
% No. of choice functions detected: <...>
% No. of choice instantiations: <...>
% SZS status <result> for <problem>
% SZS output start CNFRefutation for <problem>
  <proof>
% SZS output end CNFRefutation for <problem>
```

out parsing, respectively, by Leo-III on the input problem `<problem>`. Further information include the number of used axioms n_{Ax}, the number of processed and generated clauses and the number of subsumed clauses.

The result status value `<result>` used in the proof is a textual standardized representation of a SZS result ontology value. However, only a subset of all possible SZS results are printed by Leo-III:

`Theorem` is printed, if a problem containing a conjecture was found to be a theorem.

`ContradictoryAxioms` is printed, if a problem containing a conjecture was trivially found to be a theorem because its axioms are inconsistent.

`GaveUp` is printed, if Leo-III was not able to establish a result value due to incompleteness sources and hence terminated on its own accord.

`Timeout` is printed, if Leo-III was not able to establish a result value yet, but was terminated due to violation of time limits.

`Unsatisfiable` is printed, if a problem without conjecture was found to be inconsistent/unsatisfiable.
`Inappropriate` is printed, if a problem uses syntax features that are not supported by Leo-III (e.g., arithmetic operations).
`Forced` is printed, if Leo-III was terminated by external sources.
`SyntaxError` is printed, if a problem contains syntactical errors and can therefor not be parsed correctly.
`TypeError` is printed, if a problem uses conflicting type information.
`InputError` is printed, if a problem is otherwise malformed or missing essential information (e.g. explicit type information for constant symbols).
`Error` is printed, if any internal error occurred during execution of Leo-III. This is, in general, an implementation flaw and does not imply any statement on the quality of the input problem.

Currently, Leo-III is not capable of printing results that indicate non-validity of a given conjecture (e.g. `CounterSatisfiable`). Novel to the list of possible return values for the Leo prover family is `ContradictoryAxioms`. Up to the author's knowledge, no other (higher-order) ATP system currently checks the consistency of the generated proof on-the-fly. In Leo-III, this is implemented using a simple traversal of the generated proof object: If the negated conjecture is never used within the generated refutation, the axioms must be inconsistent and hence `ContradictoryAxioms` is returned. This feature can help to clarify errors within the problem's formalization as it points out a potential weakness of the given axioms. The implementation comes with small-to-none costs at run time as only one single traversal of the proof object is conducted.

Additional to the above described result value, Leo-III produces machine-readable proof certificates (cf. the output between "`% SZS output start`" and "`% SZS output end`" in Fig 7) if the empty clause could be derived and such a certificate has been requested (using the `-p` command-line option). The proof certificate is an ASCII encoded, linearized, directed acyclic graph (DAG) of inferences that refutes the negated input conjecture by ultimately generating the empty clause. The root sources of the inference DAG are hereby the given conjecture (if any) and all axioms that have been used for the refutation.

The proof output records all intermediate inferences, with the exception of modifying inferences (simplifications) that effectively did not change the clause. The representation again follows the TSTP format and records the inferences using annotated THF formulas. While the clause itself is encoded as THF formula using disjunctions, the information about which inference and which premise clause generated the result is recorded in the annotation field. In the case of

Table 2: Annotations used within Leo-III proofs and their corresponding calculus rules. Annotations for structural rules or specialized routines may have no calculus equivalent (marked with —).

Annotation	**Calculus rule(s)**	**Additional Information**
`neg_conjecture`	—	Negates the original conjecture.
`miniscope`	—	Applies miniscoping as described in §4.3.1.
`simp`	(Simp), (DD), (TD1), (TD2), (Triv), (Bind)	—
`cnf`	All of $\mathcal{CNF}$	—
`paramod_ordered`	(Para)	—
`eqfactor_ordered`	(Fac)	—
`prim_subst`	(Prim)	Substitution is given as additional output.
`func_ext`	(PFE), (NFE)	—
`bool_ext`	(PBE), (NBE)	—
`pre_uni`	All of $\mathcal{UNI}$	Unifier is given as additional output.
`pattern_uni`	—	Unifier is given as additional output.
`replace_leibeq`	(LEQ), (AEQ)	Substitution is given as additional output.
`simplifyReflect`	(PSR), (NSR)	—
`rewrite`	(RP), (RN)	—
`instance`	(HeuInst)	Substitution is given as additional output.
`choice`	(ACI)	—
`funcspec`	(FS)	Substitution is given as additional output.

primitive substitution and unification inferences, the applied substitutions are included as well. Table 2 contains an overview over the annotations that are used within Leo-III proofs. For each annotation, the corresponding inference rules of the underlying calculus are given. The verbose proof object can be reduced with the option `-proofcompression`. This mode skips less informative non-branching steps in the proof and compresses them into a sequence of inference applications.

Due to the fine granularity of Leo-III proofs, it is often possible to verify them step-by-step using external tools such as GDV [Sut06]. In fact, many proofs found by Leo-III without external cooperation can successfully be verified by

GDV. Note that, however, when using external cooperation it is more likely that Leo-III proofs cannot be verified this way. This is due to the proof reconstruction procedure of Leo-III which includes those formulas as inference parents that were used as premises of the inference. For external prover calls all (relevant) clauses of the search space are sent to the external system and can hence be seen as potential premises relevant to the inference. As a consequence, Leo-III over-approximates and outputs all submitted clauses as inference parents. This in turn makes it quite hard for external systems to verify the whole proof as the proof step contains more (potentially useless) formulas and only outputs a very general big-step inference.

This situation can be mitigated in parts by inspecting the proofs produced by the external systems and extracting which submitted clauses were actually used in the given certificate. Another possibility is to translate the external proof into a Leo-III refutation and to integrate it in the remaining proof object of Leo-III. The latter approach seems preferable as it produces self-contained and therefore more trustworthy proofs. However, it is debatable if the development and implementation of such translations is possible (or feasible) when considering that Leo-III is able to cooperate with nearly all available first- and higher-order systems and that these systems partly use completely different proof calculi and proof output formats.

Examples of Leo-III proofs are given at Appendix C. An analysis of the verifiability of these proofs is given in §6.4.

4.6. Reasoning in Polymorphic HOL

Interactive theorem proving systems for higher-order logic such as Isabelle/HOL [NWP02] and Coq [Pau11] are based on expressive type systems that extend simple types (as used by most higher-order ATP systems) with e.g., polymorphism, type classes, dependent types and further type concepts [Pie02]. Expressive type systems also allow to structure knowledge in terms of reusability [Tho97].

There exist increasingly many reasoning systems for first-order logic with rank-1 polymorphism, starting from SMT solvers such as Alt-Ergo [B$^+$08] and CVC4 [B$^+$11], tableaux based ATP systems such as Zenon [BCH15], up to superposition based ATP systems including Zipperposition [Cru15] and Pirate (that is, SPASS with polymorphism) [Wan14]. Some of these systems already support a recent extension of the TPTP typed first-order format TFF to rank-1 polymorphism [BP13b], called TF1, as input format. However, in a higher-order setting there are, up to the author's knowledge, no stand-alone automated theorem provers that natively support polymorphism. Hence middleware techniques such as monomorphization [BBPS16] are still required to enable a fruitful interaction between these higher-order reasoning systems and proof assistants (cf. §4.9.3).

The only exceptions concerning automation of polymorphic HOL are HOL(y)Hammer [KU15b] and the `tptp_isabelle` mode of Isabelle/HOL [NWP02] that schedule proof tactics within HOL Light [Har09] and Isabelle/HOL, respectively.

Since version 1.1, and the substantially improved in version 1.2, Leo-III supports reasoning in first- and higher-order logic with rank-1 polymorphism. The support for polymorphism has been strongly influenced by the recent development of the TH1 format for representing problems in rank-1 polymorphic HOL [KSR16] extending the de-facto standard THF syntax [SB10] for HOL. The TF1 language is closely related to TH1 and can, since it is a fragment of TH1, also be used as an input format for Leo-III.

4.6.1. Adjustments to Leo-III

The existing term and type data structures of Leo-III are already expressive enough to represent TH1 problems (cf. §4.8.1). Although currently not exploited, these data structures can capture full System F whose unrestricted use would, however, render the system inconsistent [Coq94]. Nevertheless, conservative extensions to higher-rank polymorphism – beyond TF1 or TH1 – would already be supported on a syntactic basis.

The adaption of Leo-III to TH1 reasoning did not require modifying the general proof loop as presented in Fig 4. This is primarily due to the strict separation of the loop and its underlying abstraction layers. On the control layer, however, careful modification of most of the inference control mechanisms were necessary. Here, careful handling of (free) type variables and employment of type unification were added. Additionally, on the lowest abstraction layers (the inference rule implementations) these aspects need to be considered as well. Also, an implementation of type unification for simple types with type variables is added. Three particular changes or additions are surveyed in the following.

Unification. Central to the polymorphic adaption of Leo-III's calculus is the notion of *type unification*. Analogous to the unification of two terms s and t that returns, if existent, a substitution σ such that $s\sigma \equiv t\sigma$, type unification between two types τ and ν yields a substitution σ such that $\tau\sigma \equiv \nu\sigma$, if such a substitution exists. Such a type substitution σ is then called *most general type unifier* (*mgtu*) of τ and ν. Type unification can formally be described as a rewrite procedure; the type unification algorithm employed by Leo-III is given by the transforma-

Figure 8: Type unification by transformation.

$$
\begin{array}{rcll}
\langle \tau \doteq \tau :: U, \sigma \rangle & \longrightarrow & \langle U, \sigma \rangle & \text{(TyDelete)} \\
\langle \alpha \doteq \tau :: U, \sigma \rangle & \longrightarrow & \langle U\{\tau/\alpha\}, \sigma \circ \{\tau/\alpha\} \rangle & \text{(TyBind)} \\
 & & \quad \text{if } \alpha \notin \mathrm{fv}(\tau) & \\
\langle \zeta(\overline{\tau_i}) \doteq \zeta(\overline{\nu_i}) :: U, \sigma \rangle & \longrightarrow & \langle \overline{\tau_i \doteq \nu_i} :: U, \sigma \rangle & \text{(TyDecomp)} \\
\langle \tau_1 \to \tau_2 \doteq \nu_1 \to \nu_2 :: U, \sigma \rangle & \longrightarrow & \langle \tau_1 \doteq \nu_1 :: \tau_2 \doteq \nu_2 :: U, \sigma \rangle & \text{(TyFunDecomp)}
\end{array}
$$

tion system displayed in Fig. 8. Here, the type unification problem is represented as a tuple $\langle U, \sigma \rangle$ where U is a list of unsolved type equations (the double colon concatenates list entries, the empty list is denoted $[]$) and σ is a type substitution. Since unification on types is essentially a first-order unification problem, it is decidable and unitary, i.e. yields a unique most general unifier if one exists. More precisely, if $\tau, \nu \in \mathcal{T}_{\text{pre}}$ are monotypes, τ and ν are unifiable if and only if $\langle \tau \doteq \nu :: [], \mathrm{id} \rangle \longrightarrow^{\star} \langle [], \sigma \rangle$, where id denotes the identity substitution and σ is some type substitution. In particular, it then holds that $mgtu(\tau, \nu) := \sigma$ is a most general type unifier of τ and ν.

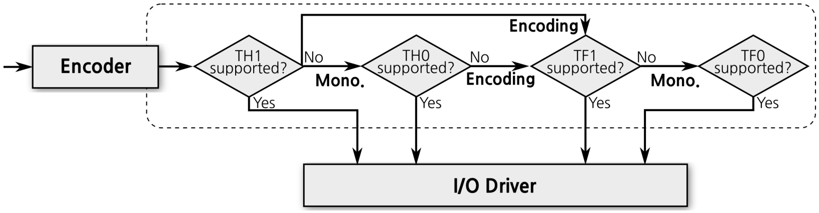
Intuitively, whenever a calculus rule requires two premises to have the same type, in the polymorphic adaption it suffices that the two terms' types are unifiable. For a concrete inference, the type unification is then applied first to the clauses; followed by the standard inference rule itself.

Skolemization. In the context of polymorphic HOL, Skolemization needs to be adapted to account for free type variables in the scope of existentially quantified variables. As a consequence, Skolem constants that are introduced e.g. during clausification are polymorphically typed symbols sk that are applied to the free type variables $\overline{\alpha^i}$ followed by the free term variables $\overline{X^i}$, yielding the final Skolem term (sk $\overline{\alpha^i}\ \overline{X^i}$), where sk is the fresh Skolem constant. A similar construction is used for general bindings that are employed by primitive substitution or projection bindings during unification.

4.6.2. External Cooperation

The use of external cooperation for polymorphically typed HO clauses is straightforward due to the nature of the existing encoding framework (cf. §4.4). Recall that the translation of (monomorphic) proof obligations first eliminates higher-order constructs by using polymorphic first-order operators. Using this translation pipeline, polymorphic HO clauses can be dealt with in two different ways: The clauses can be translated directly into polymorphically typed first-order

Figure 9: Cooperation schemes for polymorphic problems. Polymorphic problems can be reduced to monomorphic HOL by heuristic monomorphization (Mono.) or, alternatively, be translated directly to polymorphic first-order formulas for further processing.



problems. Here, only the higher-order constructs are eliminated using the usual techniques. The result then contains not only polymorphic first-order auxiliary functions introduced by the translation, but also the polymorphic symbols from the original problem. Regardless of the source of polymorphism in the first-order problem, the result is then transformed into monomorphic first-order logic using heuristic monomorphization as usual.

Another possibility is to do heuristic monomorphization first, yielding monomorphic HOL clauses. In this case, the resulting clauses can then be translated using the usual encoding procedure to first-order clauses, or be directly sent to external HO reasoners (without any further translation). The differences between both approaches have been highlighted in the context of the Sledgehammer system [BBPS16].

A schematic representation of the possible encodings for external cooperation is displayed in Fig. 9. Leo-III uses, by default, the first encoding variant that is sketched above. Currently, external prover cooperation for TH1 is functional albeit experimental. In particular for large problem inputs (e.g., produced by Isabelle/HOL's Sledgehammer tool) the current monomorphization procedure needs to be improved.

Examples of reasoning with Leo-III in polymorphic HOL are displayed at §5.2. Additionally, a detailed evaluation of Leo-III's reasoning strength on TH1 problems is conducted in §6.2.

4.7. Modal Logic Reasoning

One long-term goal of Leo-III is to provide means for reasoning within (and about) non-classical logics including, e.g., free logic, quantified conditional logic, and quantified modal logic. Non-classical logics have many topical applications in mathematics, computer science and beyond. In particular, quantified (multi-)modal logics [BvBW06] which can be fruitfully applied in the context of artificial intelligence, computational linguistics and rule-based reasoning. They also play an important role in various areas of philosophy, including ontology, (computer-)ethics, philosophy of mind and philosophy of science. Many challenging applications, however, as recently explored in metaphysics [BWP16, BWWP17, FB17], require quantified and in particular higher-order quantified modal logics (HOMLs). But even for first-order non-classical logics only few implemented systems are available, and the situation is even worse for higher-order quantified logics. There are numerous ATP systems for propositional modal logics.[11] For (first-order) quantified modal logics, however, only few reasoning systems exist. Well-known exceptions are MleanSeP[12], MleanTAP[13], GQML [TCM02], MleanCoP [Ott14] and f2p-MSPASS [HS00] that each support a different restricted subset of semantic variants of first-order modal logic. In particular, the development of ATP systems for HOMLs is still in its infancy, hence impeding more complex computer-assisted studies of relevant topics.

To overcome this situation, Leo-III bridges the above gap by providing native means[14] for flexible reasoning within every normal higher-order quantified modal logic. This is realized using a shallow semantical embedding approach [BP13a] in which formulas of modal logic are identified with specific terms of classical higher-order logic such that a notion of modal validity can be defined within HOL that coincides with the desired modal logic semantics.

A difficulty of quantified modal logic automation is that there exist multiple different notions of semantics, most of which usually used in different application

[11] A list of modal logic systems is curated by Renate Schmidt at `www.cs.man.ac.uk/~schmidt/tools`.

[12] MleanSeP can be downloaded at `www.leancop.de/mleansep`

[13] MleanTAP can be downloaded at `www.leancop.de/mleantap`

[14]The fact that modal logics as such can be translated to classical logic is well known [BVB07, BP13a]. Nevertheless, the translation process is cumbersome, error-prone and needs to be done manually by the user prior to the use of an ATP system for classical logic. The expression "native means" here highlights the fact that no such translation process or embedding procedure is necessary when using Leo-III for reasoning in HOMLs. Earlier experimental results of Benzmüller et al. used a dedicated external pre-processing system [Ben15b, BR12], this is not necessary anymore in Leo-III.

domains. The exact semantics of a given quantified modal logic can be regarded a product of multiple individual semantical parameters, including:

(i) Modal axiomatization: *What properties hold for each modality?*
The properties range from axiom scheme K alone to the strong assumptions of logic S5, and any intermediate system (cf. the modal logic cube [BvBW06]).

(ii) Quantification semantics: *What are the domains of quantified variables?*
Usual choices include so-called cumulative, decreasing, constant and varying domain semantics.

(iii) Rigidity: *Is the meaning of a symbol the same in every possible world?*
Possible choices include rigid and world-dependent constant symbols.

Also, there exist different choices for logical consequence relations, including at least so-called local and global consequence [BvBW06]. When taking all possible parameter combinations into account this amounts to more than 120 different HOMLs (cf. §4.7.2 further below).

As of version 1.2, Leo-III is capable of reasoning within all such HOMLs without the use of any external pre-processing tools. This is in contrast to other special-purpose modal logic reasoning systems that are limited to a small number of semantic variants. This makes Leo-III the first ATP system to natively support full higher-order modal logic reasoning for a large range of modal logics.

In this section, higher-order (multi-)modal logic (HOML) is sketched informally together with an overview of its many different semantic variations. Subsequently, a semantic embedding procedure for HOML into HOL is discussed. This procedure is included into Leo-III as a pre-processing module, and enables its extensive modal logic reasoning capabilities. Finally, an extension of the standard TPTP THF problem input syntax is presented. This augmented syntax is designed for formulating HOML problems as well as specifying modal semantics in which these problems are to be evaluated. The here presented procedure for modal reasoning support is based on earlier work [GSB17, WSB16] and originates from the study of embedding methods for various non-classical logics [BP13a, Ben15c].

Related approaches to generic theorem proving for various propositional modal logics include the tableau-based theorem systems LoTReC [dC$^+$01], MeTTeL2 [TSK12] and tableau workbench [AG09]. However, it is unclear whether these approaches scale for first-order or higher-order quantified modal logics.

Table 3: Popular modal axiom schemes and their corresponding frame condition

Name	**Axiom scheme**	**Condition on r^i**	**Meta-logical specification of r^i**
K	$\Box^i(s \longrightarrow t) \longrightarrow (\Box^i s \longrightarrow \Box^i t)$	—	—
B	$s \longrightarrow \Box^i \Diamond^i s$	symmetric	$r^i\ w\ v \longrightarrow r^i\ v\ w$
D	$\Box^i s \longrightarrow \Diamond^i s$	serial	$\exists v.\, r^i\ w\ v$
T/M	$\Box^i s \longrightarrow s$	reflexive	$r^i\ w\ w$
4	$\Box^i s \longrightarrow \Box^i \Box^i s$	transitive	$\left(r^i\ w\ v \land r^i\ v\ u\right) \longrightarrow r^i\ w\ u$
5	$\Diamond^i s \longrightarrow \Box^i \Diamond^i s$	euclidean	$\left(r^i\ w\ v \land r^i\ w\ u\right) \longrightarrow r^i\ v\ u$

4.7.1. Higher-Order Modal Logic

Higher-order multi-modal logics (HOML) can, roughly speaking, be regarded an extension of classical higher-order logic, augmented with a set of modal operators $\Box^i$, commonly denoting necessity, $i \in I$, for some index set I. The dual operators $\Diamond^i$, denoting possibility, can then be defined as usual for normal modal logics by taking $\Diamond^i := \lambda X_o.\, \neg\Box^i\neg X$.

Semantics of HOML is given by an appropriate adaption of Henkin semantics for HOL to Kripke semantics (also referred to as *possible world semantics*) [BvBW06]. Here, the semantics of the modal sentences $\Box^i s_o$ and $\Diamond^i s_o$, for some formula s_o, is defined in terms of accessibility relations r^i between possible worlds, where each possible world w is associated an individual HOL frame $\mathcal{D}_w$ and interpretation function $\mathcal{I}_w$. The remaining (non-modal) sentences are interpreted as usual. The resulting notions of models for HOML are a generalization of those of (first-order) quantified modal logic [Gol11] to full higher-order quantification. Note that an important aspect of modal logic is that $\Box^i s_o$ can be derived whenever s_o is a valid formula (this is called *necessitation*). Nevertheless, $s_o \longrightarrow \Box^i s_o$ is in general not a theorem of HOML. Additionally, in normal modal logics, is it agreed upon that the box operators $\Box^i$ respect the axiom scheme K from Table 3. Further properties of $\Box^i$ can be required, depending on the application scenario (cf. §4.7.2 below).

A thorough introduction to modal logics and its intricate structure can be found in the literature [BvBW06, Gar16].

4.7.2. Semantics Variations

Higher-order quantified modal logic can be equipped with many different semantics. This is due to the existence of various subtle but meaningful variations in multiple individual facets of which each combination potentially yields a distinct modal logic. Many of those variations have their particular applications, hence there is no reasonably small subset of modal logics to which a system should be restricted. This, of course, poses a major challenge to theorem proving systems for modal logics. Leo-III does, however, admit such a diverse reasoning capability for modal logics and in fact supports every normal modal logic. In contrast, popular other ATP systems for (first-order) quantified modal logics such as MleanCoP, MleanSep and MleanTab only support a comparably small subset of all possible variants. The most prominent semantic facets that can be adjusted are surveyed in the following.

Modality Axiomatizations. The most common variation for a concrete modal logic at hand is the choice of the $\Box^i$-operator's axiomatization. A subset of popular axiom schemes is displayed in Table 3. It is a well-known fact that certain modal logic formulas correspond to first-order accessibility relation conditions (particularly the so-called Sahlqvist formulas [Sah75]), also displayed in Table 3.

Modal logic systems (denoted by bold-faced names) consist of one or more axiom schemes. As an example, modal logic **K** only consists of axiom scheme K. More complex systems are then constructed by adding further axiom schemes, e.g. **M** consists of K and M; **S4** consists of K, M and 4; and **S5** consists of K, B, M and 5. As an example, the modal logic is usually chosen to be **S4** or **S5** when used in an epistemic context.

In a multi-modal logic, the choice of a specific axiomatization can be made for every different $\Box^i$-operator independently.

Quantification semantics. An intuitive, unrestricted, definition of HOML models yields so-called *varying domains* semantics. Here, we have the situation that denotations from $\mathcal{D}_w$ that exist at a particular world w may not exist in another world v. This is often called the *actualist* interpretation of quantification [Gol11] and states that everything there is (actually) exists, i.e., that there are no merely possible things.

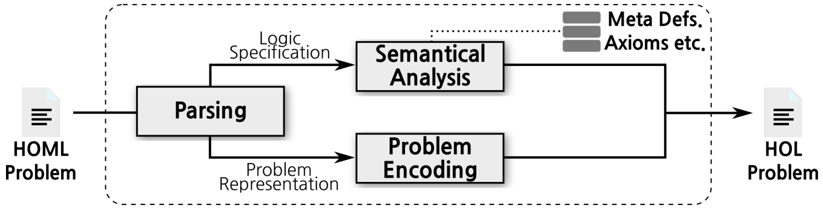
This setting may however not be adequate for all applications of modal logic, in particular in computer science, and is also criticized in the context of metaphysics from so-called *possibilist* positions. The here proposed variant of *constant domain* quantification assumes that the frames of all worlds coincide, i.e. $\mathcal{D}_w \equiv \mathcal{D}_v$ for all worlds w and v.

The remaining two settings of *cumulative domains* and *decreasing domains* are intermediate variants that assume that a frame $\mathcal{D}_w$ is a superset resp. subset of $\mathcal{D}_v$ whenever w is connected to v by the accessibility relation.

All of the above variants co-exist and there is still an ongoing dispute about the desired notion of quantification in modal logic [Wil13, Sta12].

Rigid and flexible constants. A further dimension of modal logic semantics deals with the dependency of the denotation of constants on the current world: In a so-

Figure 10: Bird's eye perspective of the automated embedding process



called *flexible* setting, two interpretation functions $\mathcal{I}_w$ and $\mathcal{I}_v$ at different worlds $w \not\equiv v$ may assign a constant symbol different denotations (except for the logical connectives such as $\neg$, $\vee$, etc. which are always denoting as usual). In a *rigid* setting, however, the interpretation functions at every world coincide, i.e. it holds that $\mathcal{I}_w(c) \equiv \mathcal{I}_v(c)$ for all worlds w, v and all constants c_τ, $\tau \in \mathcal{T}$. Aspects of rigidity play an important role for applications in paraconsistent reasoning and when dealing with vagueness.

Consequence. There is no single meaningful notion of consequence in modal logics. At least two versions of consequence relations have been discussed in the literature [FM98], including those often referred to as *global* and *local* consequences, $\models^{\text{global}}_{\text{HOML}}$ and $\models^{\text{local}}_{\text{HOML}}$, respectively. For brevity, the exact definitions are omitted here.

4.7.3. Automation of HOML in Leo-III

Automation of HOML within Leo-III is realized using an indirection: The key is to find equivalent formulations of HOML sentences in classical HOL. Recall that this translation has, in particular, to preserve the necessitation inference rule (from s infer $\Box^i s$) while not, in general, validating $s \longrightarrow \Box^i s$ for any formula s_o. In order to capture this non-trivial behavior of the modal operators, the relevant fragments of HOML's Kripke semantics are encoded into HOL. To that end, all logical connectives of HOML as well as relevant meta-logical notions such as validity (in HOML) are encoded first. Subsequently, the original modal problem is reformulated using the previously encoded (meta-)logical notions and a translation scheme for HOML terms. The result of this embedding process is a problem formulation using only constructs of classical HOL as can, hence, be dealt with using any common HO ATP system.

Embedding scheme. The key idea of the translation is that formulas of HOML s_o are encoded as HOL predicates on possible words. To that end, possible worlds

Table 4: Embedding of modal logic connectives and meta-logical notions in HOL

Lifting	HOL abbrev.	HOL term
$\lceil \Box^i_{o\to o} \rceil$	$\boldsymbol{\Box}^i_{o\mu(o\mu)}$	$\lambda S_\sigma.\lambda W_\mu.\forall V_\mu.\, \neg(r^i\, W\, V) \vee S\, V$
$\lceil \neg_{o\to o} \rceil$	$\boldsymbol{\neg}_{o\mu(o\mu)}$	$\lambda S_\sigma.\lambda W_\mu.\, \neg(S\, W)$
$\lceil \vee_{o\to o\to o} \rceil$	$\boldsymbol{\vee}_{o\mu(o\mu)(o\mu)}$	$\lambda S_\sigma.\lambda T_\sigma.\lambda W_\mu.\, (S\, W) \vee (T\, W)$
$\lceil \Pi^\tau_{o(o\tau)} \rceil$	$\boldsymbol{\Pi}^{\text{co},\tau}$	$\lambda P_{\tau\to\sigma}.\lambda W_\mu.\forall X_\tau.\, P\, X\, W$
	$\boldsymbol{\Pi}^{\text{va},\tau}$	$\lambda P_{\tau\to\sigma}.\lambda W_\mu.\forall X_\tau.\, \neg(\text{eiw}\, X\, W) \vee (P\, X\, W)$
$\models^{\text{global}}_{\text{HOML}}$	$\text{valid}_{o(o\mu)}$	$\lambda S_\sigma.\, \forall W_\mu.\, S\, W$
$\models^{\text{local}}_{\text{HOML}}$	$\mathcal{A}_{o(o\mu)}$	$\lambda S_\sigma.\, S\, w_{\text{actual}}$

are included to the problem as explicit objects of a fresh type μ. The translation of s_o is then a predicate $\boldsymbol{s}_{o\mu}$ on worlds. This encodes the idea of Kripke-semantics that the truth-value of a sentence is no longer intrinsic to the sentence itself, but also dependent of the world it is evaluated in.

The translation of HOML problems proceeds by first translating the general HOML constructs, i.e. encoding the semantics of HOML connectives etc., followed by the embedding of the actual problem itself. The translation is formalized as a function $\lceil . \rceil$ from HOML terms to HOL terms. It is extended to also denote the encoding of HOML types to HOL types in the following.

For the first part, for each modal index $i \in I$, an accessibility relation $r^i_{o\mu\mu}$ between possible worlds is postulated. Depending on the desired axiomatization of $\Box^i$, additional restrictions (axioms) of r^i are postulated. These restrictions (cf. Table 3) make use of the correspondence between modal axiom schemes and accessibility relation properties [BvBW06]. HOML types are embedded by:

$$\begin{aligned} \lceil \nu\tau \rceil &:= \lceil \nu \rceil \lceil \tau \rceil \\ \lceil o \rceil &:= o\mu \\ \lceil \tau \rceil &:= \tau\mu \qquad \text{if } \tau \text{ is base type and } (*) \end{aligned}$$

where $(*)$ only applies if in a world-dependent constants context. For rigid constants, only type o is translated to a predicate type. The logical connectives of HOML can be encoded as displayed in Table 4. The abbreviations of the respectively embedded connectives are written in bold face symbols. The encoding of universal quantification depends on whether this quantification is intended to be using constant domain or varying domain semantics (the latter case subsumes cumulative and decreasing domain semantics). Here, $\mathbf{\Pi}^{\mathrm{co},\tau}$ and $\mathbf{\Pi}^{\mathrm{va},\tau}$ denote the translation of the modal logic quantifier for constant domains and varying domains, respectively. The predicate eiw (for *exists-in-world*) guards against quantification over objects that do not exist in the world at hand. The setting of cumulative and decreasing domains is implemented by adding appropriate restrictions of the eiw predicate used in the definition of $\mathbf{\Pi}^{\mathrm{va},\tau}$.

The translation of the original problem then recursively replaces the components of the original HOML formula by the embedded HOL equivalents:

$$\lceil c_\tau \rceil := c_{\lceil \tau \rceil} \quad \lceil X_\tau \rceil := X_{\lceil \tau \rceil} \quad \lceil \lambda X_\tau . s_\nu \rceil := \lambda \lceil X_\tau \rceil . \lceil s_\nu \rceil \quad \lceil s_{\tau \to \nu}\, t_\tau \rceil := \lceil s_{\tau \to \nu} \rceil \lceil t_\tau \rceil$$

Finally, the notions of HOML validity resp. consequence is encoded with aid of two further meta-logical definitions $\mathrm{valid}_{o(o\mu)}$ and $\mathcal{A}_{o(o\mu)}$. Both function symbols are grounding terms of type $o\mu$ (i.e., embedded formulas of HOML) to type o (formulas of HOL) and assert that the respective formula is valid. The difference between these two notions is that the first one encodes global consequence semantics while the latter can be used to encode local consequence. The definitions of both operators are displayed in Table 4. Here, w_{actual} is a fresh world constant.

Details on the embedding procedure and the underlying theoretical considerations can be found in previous work [GSB17, BP13a].

Utilization within Leo-III. The refutation process and associated operations remain unchanged by this approach since the problem encoding is automatically done in a pre-processing step, transparent to the underlying reasoning modules. A schematic invocation of this pre-processing step is displayed at Fig. 10. The input of this translation pipeline is a modal problem statement formulated in a language specifically created for HOML. After parsing, the meta-logical contents of the problem statement (the semantical specification, cf. further below) are processed. This produces HOL encodings of corresponding logical and meta-logical notions of HOML. In a second step, the problem itself is embedded as sketched above. The results of this pre-processing operation is then passed to Leo-III just like any regular (non-modal) problem input.

Since the syntax of HOML is a conservative extension of that of classical HOL, the THF representation language can easily be augmented by introducing the modal operators `$box` and `$dia` as new primitive connectives, representing the modal connectives $\Box$ and $\Diamond$, respectively (in a mono-modal settings). For example, the HOML formula $\forall X_\iota . \Box (p\ X)$ then corresponds to `! [X:$i] : ( $box @ ( p @ X ) )` in the proposed syntax extension. For multi-modal logics, there exist analogous operators that are additionally given

Figure 11: Layout of a general modal logic problem. The first statement (ll. 2 - 4) specifies a concrete modal logic, the remaining statements (ll. 6 - 8) formulate the problem itself. The ⟨name$_i$⟩ serve as syntactic identifier for that statement, a ⟨role$_i$⟩ (usually set to `axiom` or `conjecture`) tells the reasoning system how to interpret the ⟨formula$_i$⟩ formulated in the presented augmented THF syntax. Lines starting with `%` are comments.

```
% Begin of logic specification
thf(⟨name_0⟩, logic, ($modal := [
      $constants := ⟨const_spec⟩, $quantification := ⟨domain_spec⟩,
      $consequence := ⟨conseq_spec⟩, $modalities := ⟨modal_spec⟩ ]
     )).
% End of logic specification, begin of problem statement
thf(⟨name_1⟩, ⟨role_1⟩, ⟨formula_1⟩).
...
thf(⟨name_n⟩, ⟨role_n⟩, ⟨formula_n⟩).
```

an index as first argument and the formula as second argument. The remaining syntax coincides with standard THF and is described in the literature [SB10].

As mentioned earlier, there is no single semantics of quantified modal logics. As a consequence, there is additional need for explicitly stating the semantical setting in which a problem is to be assessed by the reasoning system. This is realized by including a meta-logical specification into the problem header which is represented as an input statement of type `logic` that assigns the desired semantics for each discussed semantic dimension (cf. §4.7.2). A prototypical specification statement is displayed in Fig. 11: The identifiers `$constants`, `$quantification` and `$consequence` specify the exact semantical settings for the rigidity of constant symbols, the quantification semantics and the consequence relation, respectively. Finally, `$modalities` specifies the properties of the modal connectives by means of fixed modal logic system names or, alternatively, a list of individual modal axiom schemes. The valid parameter values are given in Table 5.

The remaining placeholders of Fig. 11, ⟨name⟩, ⟨role⟩ and ⟨formula⟩, are standard and given by the TPTP language definition [Sut17b]. This logic specification approach was fostered in earlier work [WSB16] and subsequently improved and enhanced to a work-in-progress TPTP language extension proposal.[15] Due to the flexibility of the semantics specification of this input syntax and the

[15] See `http://www.cs.miami.edu/~tptp/TPTP/Proposals/LogicSpecification.html`.

Table 5: Semantic specification parameters. The parameter placeholders, written in angles $\langle\cdot\rangle$, refer to the values for the logic specification of Fig. 11. The names of the modal logic system parameters (such as `$modal_system_K` or `$modal_system_S5`) refer to the respective systems from the modal logic cube [BvBW06]. The individual modal axiom schemes names (such as `$modal_axiom_T` or `$modal_axiom_5`) are named similarly.

Parameter	Valid values
⟨const_spec⟩	`$rigid`, `$flexible`
⟨domain_spec⟩	`$constant`, `$varying`, `$cumulative`, `$decreasing`
⟨conseq_spec⟩	`$local`, `$global`
⟨modal_spec⟩	`$modal_system_X` for $\mathtt{X} \in \{\mathtt{K},\mathtt{KB},\mathtt{K4},\mathtt{K5},\mathtt{K45},\mathtt{KB5},\mathtt{D},\mathtt{DB},\mathtt{D4},\mathtt{D5},\mathtt{D45},\mathtt{T},\mathtt{B},\mathtt{S4},\mathtt{S5},\mathtt{S5U}\}$ *or* `[$modal_axiom_X`$_1$`,$modal_axiom_X`$_2$`, ...]` for $\mathtt{X}_i \in \{\mathtt{T},\mathtt{B},\mathtt{D},\mathtt{4},\mathtt{5},\mathtt{CD},\mathtt{C4},\mathtt{C}\}$

flexibility of Leo-III's underlying semantical embedding approach, Leo-III also supports multi-modal logics, user-defined combinations of rigid and non-rigid constants and different quantification semantics for each type.

Fig. 12 displays an example modal logic formula that is an instance of a corollary of Becker's postulate [Bec30]. It essentially expresses that everything that is possibly necessary is, in fact, necessary. It is known from the literature, that Becker's postulate is indeed valid in **S5** modal logics but not in any weaker logic systems. Even without this knowledge, Leo-III can be used to experimentally reproduce these results. To that end, each semantic setting can be formulated as logic specification and then assessed by Leo-III.

Experiments show that the automation approach presented above is indeed competitive with modal logic reasoners that are specifically implemented for that purpose [GSB17, BR13]. An example of modal logic reasoning within Leo-III is displayed in §5.3.

4.8. Implementation Details

Data structure choices are a critical part of a theorem prover and permit reliable increases of overall performance when implemented and exploited properly. Key aspects for efficient theorem proving such as term representations and indexing data structures have been an intensive research topic and have reached a convinc-

Figure 12: A corollary of Becker's postulate formulated in modal THF, representing the formula $\forall P_{\iota\to o} \forall F_{\iota\to\iota} \forall X_\iota \exists G_{\iota\to\iota} (\Diamond\Box P(F(X)) \Rightarrow \Box P(G(X)))$.

```
thf(s5_spec, logic, ($modal := [
      $constants := $rigid, $quantification := $constant,
      $consequence := $global, $modalities := $modal_system_S5 ])).
thf(becker,conjecture,( ! [P:$i>$o,F:$i>$i, X:$i]: (? [G:$i>$i]:
      (($dia @ ($box @ (P @ (F @ X)))) => ($box @ (P @ (G @ X))))))).
```

ing degree of maturity within first-order ATP systems [SRV01]. In the context of the Leo-III prover, quite some effort was invested into designing appropriate data structures for HOL reasoning procedures. While their development is still far from being as mature and optimized as their first-order counterparts, the current data structures seem suited for practical applications.

A collection of basic data structures and algorithms for the implementation of higher-order reasoning systems has been isolated from the implementation of Leo-III into a dedicated framework called LEOPARD (Leo's Parallel Architecture and Datastructures) [WSB15] which is freely available at GitHub.[16] This framework provides many useful stand-alone components, including a term data structure for polymorphic λ-terms, unification and subsumption procedures, parsers for all TPTP languages and further utility procedures and pretty printers for TSTP-compatible system output.

In the remainder of this section, some implementation aspects of Leo-III and LEOPARD are highlighted.

4.8.1. Term Data Structure

In automated reasoning systems, terms are the most general and common pieces of information that are accessed, manipulated and created by most routines of the reasoning system. It is therefore not surprising that the internal representation of terms is a crucial detail which has direct consequences on the efficiency of the whole system.

Leo-III implements a combination of term representation techniques that aims at providing terms that admit an expressive typing system, efficient basic term operations and reasonable memory consumption. This is in contrast to the usual curried representation of λ-terms which comes with a major drawback for

[16] Details about the LEOPARD system can be found at `https://github.com/leoprover/LeoPARD`.

automation: There are many logical procedures where the head symbol of a term needs to be accessed or the individual arguments of an application need to be examined in the left-to-right reading order (e.g. unification or matching procedures). However, in a naive curried representation of a λ-term the head symbol of a term is deeply buried under several layers of applications. Hence for each head access a linear number of traversal operations need to be performed. A similar observation holds for left-to-right argument traversal.

To overcome this weakness of a classical term representation, Leo-III uses a so-called spine notation [CP03], which imitates first-order-like terms in a higher-order setting. Here, terms are either type abstractions, term abstractions or applications of the form $f \cdot (s_1; s_2; \ldots)$ where the head f is either a constant symbol, a bound variable or a complex term and the spine $(s_1; s_2; \ldots)$ is a linear list of arguments that are, again, spine terms. Note that if a term is β-normal, f cannot be a complex term. This observation led to a internal implementation distinction between β-normal and (possibly) non-β-normal spine terms where the first kind has an optimized representation whose head having only associated a reference to an integer representing the constant symbol or variable.

Additionally, the term representation employs explicit substitutions [ACCL90]. In a setting of explicit substitutions, substitutions are part of the term language and can thus be postponed and composed before being applied to the term. This technique admits more efficient β-normalization and substitution operations as terms are traversed only once, regardless of the number of substitutions applied.

The term data structure implements a locally nameless representation using de Bruijn-indices [Bru72]. In the setting of polymorphism, types may also contain variables. Consequently, the nameless representation of variables is extended to type variables [KRTU99]. The definition of De Bruijn indices for type variables is analogous to the one for term variables. In fact, since only rank-1 polymorphism is used, type indices are much easier to manage than term indices. This is due to the fact that there are no type quantifications except for those on top level. One of the most important advantages of nameless representations over representations with explicit variable names is that α-equivalence is reduced to syntactical equality, i.e. two terms are α-equivalent if and only if their nameless representation is equal.

The techniques presented here are to some varying degree used by state-of-the-art reasoning systems, such as Teyjus λProlog [NM99] (which is based on explicit substitutions of the Suspension Calculus [Nad99]), the logical frameworks TWELF [PS99], Beluga [PD10], and the interactive Abella prover [Gac08]. Nevertheless, the combination of techniques for term data structures presented here is, up to the author's knowledge, unique in the context of higher-order ATP systems.

The type abstraction mechanism ($\Lambda.\ s$) is due to Girard and Reynolds, who independently developed a polymorphically typed λ-calculus today widely known as System F [Gir72, Rey74]. Note that Leo-III does not exploit the full typing flexibility of the presented term and type language. This is due to the fact that unrestricted use of polymorphism renders the underlying system inconsistent [Coq94] and hence of no use for an employment within an ATP system.

Term sharing. Terms are perfectly shared within Leo-III, meaning that each term is only constructed once and then reused between different occurrences. This not only reduces memory consumption in large knowledge bases, but also allows constant-time term comparison for syntactic equality using the term's pointer to its unique physical representation. For fast basic term retrieval operations (e.g. head symbol, subterm occurrences) terms are kept in β-normal η-long form.

β-normalization schemes. Leo-III comes with a number of different β-normalization strategies that adjust the standard leftmost-outermost strategy with different combinations of strict and lazy substitution composition resp. normalization and closure construction. This research is motivated by previous observations that suggest that there is no single best normalization strategy [SB15].

4.8.2. Indexing Techniques

Indexing data structures are used, in particular in first-order theorem proving, for speeding up queries regarding terms, literals or clauses with respect to certain query relations. Employment of indexing methods improve the efficiency of operations that are frequently invoked by the reasoning system, such as finding unifiable terms or subsumed clauses subsumption [SRV01].

In higher-order theorem proving, there exist only few indexing data structures. This is due to the fact that most operations used for generating and maintaining term or clause indexes are not decidable, e.g. computing the most specific generalization or higher-order unification. For the latter case, however, there exists an indexing approach for the restricted unification fragment of linear higher-order patterns which uses substitution trees in a setting of contextual modal type theory [Pie09]. In practice, the algorithms used by this indexing approach are very complicated and require considerable adaption work for the term language used by Leo-III. Additionally, this technique seems not to be very effective in practice [PP03].

Following LEO-II, term indexing mechanisms for low-level operations are available in Leo-III [TB06, Ste14]. These operations include subterm retrieval, symbol-based look-up queries for occurs checks and improvements of β-normalization procedures by early term subtree cutoffs. In contrast to LEO-II, head symbol indexing is not required anymore due to the efficient (constant-time) head symbol access in the term representation presented above.

Additionally, Leo-III uses a higher-order adaption of feature vector indexing [Sch13]. It is used to reduce the number of matching tests during the subsumption procedure of Leo-III. The adapted feature vector index has limitations when indexing terms with variables at head positions but nevertheless seems suitable in practice. A formal, more thorough investigation about its practical benefits in this setting is further work.

4.8.3. Management of External Reasoners

Automated reasoning systems such as theorem provers or model finders significantly differ in scope and supported language features. While projects such as TPTP [Sut17b] and SMT-LIB [BFT16] provide a widely accepted infrastructure for automated reasoning systems within their domain, including standard language representations for input problems and generated outputs (e.g. proof objects), not all of the postulated representation languages are supported by a given system.

Knowledge about a system's supported logic and language features within that particular representation language is of special importance to Leo-III which is designed to cooperate with such systems, in particular first-order ATP systems, during proof search. Since it is of no importance which (first-order) prover is employed as long as it supports input and output according to TPTP standards, Leo-III can cooperate with any reasoning system that a user may register (e.g. via command-line options). Nevertheless, depending on the features of the ATP system, the mode of cooperation between Leo-III and that system differs in terms of necessary translations of higher-order proof obligations (cf. Fig. 9).

In earlier work [SWSB17], the use of a dedicated descriptive artifact that enumerates a system's capabilities (e.g., its supported logics) was sketched such that Leo-III can decide during run-time how to communicate with that external system. Internal representations of capability objects for external reasoning systems such as CVC4, E, iProver, Vampire, Nitpick, Isabelle and Satallax are included into Leo-III and used during proof search for identifying the concrete mode of cooperation per system. Static capability description files for the LEO-

II and Leo-III prover can be accessed at the Leo-III project website[17]. Alternatively, Leo-III supports the `-caps` switch to print its capability description on the command-line.

4.9. Additional Features

Additional to the capabilities that have been discussed above, the Leo-III systems provides further, more pragmatically motivated, features. In the spirit of its predecessor LEO-II, Leo-III also provides an *guided mode*[18] that allows experienced users to guide the proof search. Also, Leo-III can be used via a graphical user interface (GUI) that aims at the tight integration of ATP system invocation to a sophisticated *what-you-see-is-what-you-get* text editor. Furthermore, in order to contribute to the well-justified collaboration of interactive and automatic theorem proving systems, Leo-III has been integrated into Isabelle/HOL's Sledgehammer system [BN10b, BBP13]. These and further aspects are described in more detail in the following.

4.9.1. Interactively Guided Refutation

Leo-III is layed-out as an automated theorem prover, i.e. it searches for a refutation of the input problem without any user interaction. As pointed out in §4.3.2, this involves the use of various heuristics for e.g. the selection of clauses, the selection of literals within clauses and the restriction of certain inferences. These heuristics, however, may also misdirect the proof search for certain problems, thus hindering or slowing down a successful refutation attempt. On the other hand, expert knowledge of humans may supersede syntactic heuristics in some cases. This is why Leo-III includes a so-called *guided mode* that allows users to influence the otherwise fully automatic refutation process by e.g. producing certain inferences by hand or preferring clauses that would otherwise never[19] be selected.

This guided mode is enabled by a range of commands that an expert may use to analyze the search space and heuristic evaluations, inspect certain clauses and their properties, manually apply inferences to existing clauses and to alter

[17]See `http://inf.fu-berlin.de/~lex/leo3/#downloads`.

[18] LEO-II called this *interactive*, but since interactive theorem proving system offer far more functionality and, more importantly, a quite different motivation and aim, the analogous mode of Leo-III is called "guided" to emphasize the difference.

[19]Of course, the clause selection heuristics of Leo-III are fair and, in theory, every clause will be selected after an arbitrary but finite amount of time. This may, however, in practice not happen within specified time resource limits.

Table 6: Commands for Leo-III's guided mode

	Command	**Effect**
Inspection/Control	`sig`	Print the current signature.
	`queue`	Print a summary of all clause queues.
	`peek`	Print the clause that will be selected next and the queue id from which it is selected from.
	`eval`	Print the priority value of `<cl>` for the heuristic evaluation function associated with `<queue>` (lower is better).
	`take`	Enqueue `<cl>` for selection. All clauses enqueued by `take` are selected before any clause from the regular queues.
	`sleep`	Skip the next `<n>` iterations of the proof loop before asking for user interaction again.
	`insert`	Insert `<cl>` from temporary clause store into the unprocessed set (if not already member of that set).
	`remove`	Remove `<cl>` from every queue or from the processed set (if already selected before).
	`vars`	Print all free variables occurring in `<cl>`.
	`typeVars`	Print all free type variables occurring in `<cl>`.
	`trivial`	Inspect if `<cl>` is trivially true.
	`redundant`	Inspect if `<cl>` can be shown redundant with respect to the current processed set.
Inferences	`para`	Apply paramodulation with `<cl1>` (using `<lit1>` for rewriting) into `<cl2>` (selecting `<lit2>`).
	`fac`	Apply factoring to `<cl>`, selecting `<lit1>` and `<lit2>`.
	`primsubst`	Apply primitive substitution to `<cl>`, substituting `<var>` by an approximating binding for `<atom>`.
	`funcext`	Apply functional extensionality rules to `<cl>`
	`boolext`	Apply Boolean extensionality rules to `<cl>`
	`unify`	Exhaustively apply all (pattern/pre-)unification rules on `<cl>`.
	`cnf`	Exhaustively calculate the CNF of `<cl>`.
	`simp`	Simplify the clause `<cl>` with all simplification rules that do not depend on other clauses.
	`rewrite`	Rewrite `<cl1>` using unit clause `<cl2>`.
Utility	`write`	Write the current processed set as TPTP problem to `<path>` using `<format>` (default: thf).
	`pretty`	Display the internal representation of `<cl>`.
	`tptp`	Display the TPTP THF representation of `<cl>`.
	`help`	Print a help message or help to a specific commands.
	`exit`	Terminate Leo-III.

the search space traversal. See Table 6 for a complete list of available commands within Leo-III's guided mode. The parameters to each command are explained within the commands description as typewriter font identifiers, as in `<cl>`. Here, parameters of the form `<cl>` denote specific clauses (identified by their unique numeric id), `<lit>` denote literal indexes (starting at 0) where `<atom>` denotes an numeric id of a signature entry. All other parameters are strings. The invocation of a command may generate one or more new clauses that are stored in a temporary set. A user may choose to insert specific clauses to the unprocessed set using `insert`. Only clauses contained in the unprocessed or processed set may be used for executing inferences.

Of course, the guided mode of Leo-III can also be used for educational purposes, e.g., for lectures on theorem proving. However, admittedly, using this mode is quite technical and requires some expert knowledge. In order to fully exploit the educational possibilities of Leo-III, the user interface as well as the documentation would need to be improved and adapted to the target audience.

4.9.2. Editor Front-end for Leo-III

Automated theorem proving systems are, from the perspective of usability and user experience aspects, not as mature, robust and well-developed as current proof assistants like Isabelle/HOL. Usage of such ATPs still require a considerable amount of (partly system-related) expert knowledge and tedious legwork for formulating input problems correctly and, in some cases, interpreting error messages and problem reports.

In order to mitigate this drawbacks, a graphical user interface (GUI) has been developed that serves as a text editor for logical problems as well as communication and invocation platform for ATP systems. The GUI has been developed in the context of an university project that has been co-supervised by the author. The text editor offers eager syntax checking with in-place error reporting, syntax highlighting for the problem formulation and, natively included, means for invoking ATP systems on the hereby produced problem within the GUI. It has been primarily developed for use in conjunction with Leo-III but is not fixed to any one system. However, it requires the provers to comply with TPTP standards.

4.9.3. Integration to Isabelle/HOL

Isabelle/HOL is an interactive theorem prover for higher-order logic. It offers, among many other features, a subsystem called Sledgehammer [BN10b, BBP13] that bridges between the realm of interactive theorem proving on the one side and the fully automated theorem proving approach on the other side. That is, a user may, at any point within the construction of a proof, call the Sledgehammer system to have many different ATP systems trying to solve the current goal automatically. If this succeeds, a proof (method) may be reconstructed from the proof output of one the successful ATP calls. This approach proved very effective in practice [PB10].

In order to foster the use of powerful higher-order ATP within proof assistants, Leo-III 1.2 has been integrated into Isabelle/HOL's Sledgehammer system (as of Isabelle/HOL 2018). The integration itself has mainly been of simple implementation work since most of the routines that were used to integrate LEO-II could be reused. Also, since Leo-III is maximally compatible with TPTP specifications, standard routines for generating TPTP problems and parsing TSTP proofs could be employed. Although a more extensive evaluation of this integration is future work, first experimental results look promising (cf. §6).

4.9.4. Further Minor Features

Leo-III supports various additional functionality regarding aspects of which not all are directly connected to proof search itself. Features utilizing Leo-III as a useful pre-processing tool include plain syntax and type checks for any TPTP language up to TH1, translation from any TPTP language to THF syntax and the translation of higher-order problems to (polymorphically) typed first-order problems in TFF syntax using the encoding displayed in §4.4. Leo-III offers a dedicated mode for checking for inconsistencies within an input problem's axioms. Here, the problem's conjecture is dropped internally (if existent) and the remaining problem consisting of type definitions, definitional declarations and axioms is saturated. If the empty clause can be derived without use of the problem's conjecture, the axioms are obviously unsatisfiable and Leo-III will report so. Regardless of the use of the consistency mode, an occurrence check of the (negated) conjecture in a proof derivation is always enabled in Leo-III (see discussion of `CAX` values in §6). Nevertheless, it is possible that a derivation of the empty clause is found by Leo-III which uses the conjecture although there exists a derivation that does not include it. For such (speculative) cases, the above consistency mode can be tried. The usefulness of such an option is confirmed by an evaluation of inconsistencies within the TPTP library (cf. discussion of `CAX` values in §6). A more complex approach for detecting inconsistencies in (first-order) theories is presented by Schulz et al. [SSUP17]. It can be expected that such a specialized technique finds more inconsistencies than the simple routine described above. The detection of contradictory axioms within Leo-III does not aim at providing such a powerful routine but merely a pragmatic sanity check

Table 7: Command-line options for pre-processing features of Leo-III

Parameter	Options	Effect
`--syntaxCheck`	*none*	Perform a syntax check on the input problem. Returns `SUCCESS` if the problem is a syntactically valid TPTP problem statement, `SyntaxError` otherwise.
`--typeCheck`	*none*	Perform a type check on the input problem. Returns `SUCCESS` if the problem is well-typed, `TypeError` otherwise.
`--toTHF`	`mono` *or* `poly`	Translate the input problem to an equivalent TPTP THF problem statement. If `mono` is specified (default), the result will consist of plain TH0 formulas (this may require heuristic monomorphization if polymorphic parts are contained within the input problem). If `poly` is specified, polymorphic constructs (TH1 features) are kept. Returns `SUCCESS` and the translated problem as SZS output if successful.
`--toTFF`	`mono` *or* `poly`	Similar to `--toTHF`, but higher-order features of the problem are additionally encoded into first-order logic using explicit application operators and some lambda elimination technique (SKI combinators by default).
`--filterAxioms`	*none*	Applies relevance filtering to the input problem and returns `SUCCESS` with an SZS output containing only the relevant parts of the input problem as result.
`--consistency`	*none*	Performs a proof search on the input problem without considering the conjecture (if existent). Returns `Unsatisfiable` if successful.

that can be executed with almost no runtime penalty. Due to the flexible co-operation scheme implemented by Leo-III it is comparably simple to include a (higher-order) counter-model finder such as Nitpick that could then be executed in parallel to the actual proof search to check the axioms for (in-)consistency.

An overview of these features together with their corresponding command-line parameter is displayed in Table 7.

5. Application Examples

In this chapter, a survey of various application scenarios of Leo-III is presented. Leo-III is primarily targeted at reasoning within classical higher-order logic as specified by the TPTP THF (TH0) standard.[1] Nevertheless, as described in more detail before, the Leo-III ATP system can also be applied to further reasoning tasks. This includes reasoning in (typed and untyped) first-order logic, polymorphic HOL as well as various higher-order quantified modal logics. This makes Leo-III, up to the author's knowledge, to the most widely applicable theorem proving system. Additionally, due to the flexibility of its cooperation mechanisms, Leo-III can also be employed as a meta-prover. These application scenarios are exemplarily discussed in the context of small case studies that focus on interesting aspects of Leo-III's reasoning capabilities.

This chapter is organized as follows: Examples from classical higher-order reasoning are given in §5.1. This includes equality reasoning, reasoning with choice and applications of Leo-III's function synthesis techniques. Subsequently, applications of Leo-III on polymorphic reasoning tasks are studied in §5.2. Modal logic reasoning examples are then presented in §5.3. Finally, meta-prover capabilities of Leo-III are sketched in §5.4.

All of the proofs that were generated by Leo-III on problem examples from §5.1 have been verified by GDV; see §6.4 for a more thorough discussion.

5.1. Higher-Order Reasoning

In the context of higher-order reasoning, challenging problems as well as the corresponding proofs that are found by Leo-III are studied. Most of the following problems have previously not been solved by any other HO ATP system. The problems displayed here are chosen in order to demonstrate some reasoning features that are specific to Leo-III. A more thorough evaluation of the effectivity of Leo-III is then presented in §6.1.

In the following, roman font is used within surveyed proofs by convention for denoting reasoning steps of Leo-III that can be seen as direct applications of the corresponding calculus rules. Sans font is used when referring to an inference step of Leo-III that typically includes multiple (low-level) calculus rules

[1] The TPTP infrastructure requires THF reasoners to assume Henkin semantics with choice.

which are nevertheless bundled for pragmatic reasons (e.g., clausification and unification).

Throughout the application examples, the X^i denote fresh free variables of appropriate type.

Equational reasoning. There are many hard and interesting problems that require equational reasoning. Four problems as well as their proofs generated by Leo-III are exemplarily discussed in this setting.

E1 *Surjective Cantor Theorem. Cantor's Theorem* states that, given a set A, the power set of A has a strictly greater cardinality than A itself. Despite being of fundamental importance to elementary set theory, it allows a most elegant application of higher-order formalism and a suitable case study for higher-order reasoning. The core argument of the proof can be formalized as follows:

$$\neg\exists f_{o\iota\iota}.\, \forall Y_{o\iota}.\, \exists X_{\iota}.\, f\, X = Y \tag{1}$$

This formula states that there exists no surjective function f from a set to its power set[2], encoded using only primitives of HOL such as quantification about functions and sets. Due to the the use of surjective functions this particular argument is also referred to as *surjective Cantor theorem.*
Let $\mathbf{E}_1$ denote formula (1) from above. The proof of $\mathbf{E}_1$ that found by Leo-III contains applications of functional extensionality, Boolean extensionality, primitive substitution as well as non-trivial higher-order pre-unification. The proof is given below; the Skolem symbols sk^1 and sk^2 used therein have type $o\iota\iota$ and $\iota(o\iota)$, respectively.

$\mathsf{CNF}(\neg\mathbf{E}_1), \text{LiftEq}:$	$\mathcal{C}_1: [sk^1\ (sk^2\ \mathrm{X}^1) \simeq \mathrm{X}^1]^{\mathtt{tt}}$
$\text{PFE}(\mathcal{C}_1):$	$\mathcal{C}_2: [sk^1\ (sk^2\ \mathrm{X}^1)\ \mathrm{X}^2 \simeq \mathrm{X}^1\ \mathrm{X}^2]^{\mathtt{tt}}$
$\text{PBE}(\mathcal{C}_2):$	$\mathcal{C}_3: [sk^1\ (sk^2\ \mathrm{X}^1)\ \mathrm{X}^2]^{\mathtt{tt}} \vee [\mathrm{X}^1\ \mathrm{X}^2]^{\mathtt{ff}};$
	$\mathcal{C}_4: [sk^1\ (sk^2\ \mathrm{X}^3)\ \mathrm{X}^4]^{\mathtt{ff}} \vee [\mathrm{X}^3\ \mathrm{X}^4]^{\mathtt{tt}}$
$\text{Prim}(\mathcal{C}_3, \mathcal{GB}^{\neg}_{o\iota}), \mathsf{CNF}:$	$\mathcal{C}_5: [sk^1\ \big(sk^2\ (\lambda Z_\iota.\, \neg(\mathrm{X}^5\ Z))\big)\ \mathrm{X}^2]^{\mathtt{tt}} \vee [\mathrm{X}^5\ \mathrm{X}^2]^{\mathtt{tt}}$
$\text{Prim}(\mathcal{C}_4, \mathcal{GB}^{\neg}_{o\iota}), \mathsf{CNF}:$	$\mathcal{C}_6: [sk^1\ \big(sk^2\ (\lambda Z_\iota.\, \neg(\mathrm{X}^6\ Z))\big)\ \mathrm{X}^4]^{\mathtt{ff}} \vee [\mathrm{X}^6\ \mathrm{X}^4]^{\mathtt{ff}}$
$\text{Fac}(\mathcal{C}_5), \mathsf{UNI}:$	$\mathcal{C}_7: [sk^1\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\, Z)\big)\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\, Z)\big)]^{\mathtt{tt}}$
$\text{Fac}(\mathcal{C}_6), \mathsf{UNI}:$	$\mathcal{C}_8: [sk^1\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\, Z)\big)\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\, Z)\big)]^{\mathtt{ff}}$
$\text{Para}(\mathcal{C}_7, \mathcal{C}_8), \text{Triv}:$	$\square$

[2] Note that, without loss of generality, only the (power) set of objects of type ι is considered. Also, given a type τ, sets A of objects of type τ are represented in HOL as predicates $\chi^A_{o\tau}$ of type $o\tau$, also known as the *characteristic function* of A, hence the type of f and Y in formula (1).

The unifier $\sigma_{\mathcal{C}_7}$ calculated by UNI for producing $\mathcal{C}_7$ is given by (analogously for $\mathcal{C}_8$):

$$\sigma_{\mathcal{C}_7} \equiv \left\{ sk^2 \left(\lambda Z_\iota . \neg(sk^1\, Z\, Z)\right)/\mathrm{X}^2, \lambda Z_\iota . sk^1\, Z\, Z/\mathrm{X}^5 \right\}$$

Together with the substitution $\sigma_{\mathcal{C}_3} \equiv \left\{\lambda Z_\iota . \neg(\mathrm{X}^5\, Z)/\mathrm{X}^1\right\}$ generated by approximating $\neg_{oo}$ by (Prim) on $\mathcal{C}_3$, the free variable X^1 in $\mathcal{C}_1$ is instantiated within the refutation procedure by $\sigma_{\mathcal{C}_7} \circ \sigma_{\mathcal{C}_3}(\mathrm{X}^1) \equiv \lambda Z_\iota . \neg(sk^1\, Z\, Z)$. Intuitively, this instantiation encodes the so-called *diagonal set of* sk^1, given by $\{x \mid x \notin sk^1(x)\}$, as used in the traditional proofs of Cantor's Theorem.
This problem is part of the TPTP problem library as problem `SET557^1.p` and can be solved by most HOL ATP systems, excluding Isabelle/HOL. The full proof of Leo-III is displayed at Appendix C.1. Note that Leo-III implements an optimized variant of factorization that allows factoring two literals with different polarity if one of that literals is flexible. Formally, this step can be derived by instantiating the flexible head by an approximation of negation via primitive substitution, followed by regular factorization (the literals now have the same polarity after re-clausification). The above proof explicitly displays the primitive substitution of negation while the proof of Leo-III incorporates the contracted variant (cf. inferences `199` and `18` in C.1).

E2 *Injective Cantor Theorem*
Another argument for the proof of Cantor's Theorem can be formalized using the concept of injectivity instead of surjectivity:

$$\neg\exists f_{\iota(o\iota)} . \forall X_{o\iota}, Y_{o\iota} . f\, X = f\, Y \longrightarrow X = Y \tag{2}$$

Here, the existence of an injective function from a set's power set to the original set is negated. This particular version of the argument is referred to as the *injective Cantor theorem.*
While both formulas can be used to express the same fact, only the surjective variant can commonly be automatically solved by contemporary HO ATPs. Leo-III is, however, able to automatically prove both the surjective as well as the injective cantor theorem in a few seconds.
Let $\mathbf{E}_2$ denote formula (2) from above. Similar to the proof of $\mathbf{E}_1$, Leo-III applies functional extensionality, Boolean extensionality and unification procedures to $\mathbf{E}_2$. However, unlike to the surjective case, this refutation makes use of Leo-III's capability to infer the existence of left inverses for injective functions. Using this inverse function (denoted sk^2 below), the

remaining proof follows an analogous diagonalization approach as for $\mathbf{E}_1$. The introduced Skolem symbols sk^1 and sk^2 are of type $\iota(o\iota)$ and $o\iota\iota$, respectively.

$\mathsf{CNF}(\neg\mathbf{E}_2)$, LiftEq:	$\mathcal{C}_0: [sk^1\ X^1 \simeq sk^1\ X^2]^{\mathit{ff}} \vee [X^1 \simeq X^2]^{\mathit{tt}}$
$\mathrm{PFE}(\mathcal{C}_0)$:	$\mathcal{C}_1: [sk^1\ X^1 \simeq sk^1\ X^2]^{\mathit{ff}} \vee [X^1\ X^3 \simeq X^2\ X^3]^{\mathit{tt}}$
$\mathrm{INJ}(\mathcal{C}_0)$:	$\mathcal{C}_2: [sk^2\ (sk^1\ X^4) \simeq X^4]^{\mathit{tt}}$
$\mathrm{PFE}(\mathcal{C}_2)$:	$\mathcal{C}_3: [sk^2\ (sk^1\ X^4)\ X^5 \simeq X^4\ X^5]^{\mathit{tt}}$
$\mathrm{Para}(\mathcal{C}_3,\mathcal{C}_1)$:	$\mathcal{C}_4: [sk^1\ X^1 \simeq sk^1\ X^2]^{\mathit{ff}} \vee [X^1\ X^3 \simeq X^4\ X^5]^{\mathit{tt}} \vee$ $[sk^2\ (sk^1\ X^4)\ X^5 \simeq X^2\ X^3]^{\mathit{ff}}$
$\mathsf{UNI}(\mathcal{C}_4)$:	$\mathcal{C}_5: [sk^2\ \big(sk^1\ (X^7\ X^3)\big)\ (X^6\ X^3) \simeq X^7\ X^3\ (X^6\ X^3)]^{\mathit{tt}}$
$\mathrm{PBE}(\mathcal{C}_3)$:	$\mathcal{C}_6: [sk^2\ (sk^1\ X^4)\ X^5]^{\mathit{ff}} \vee [X^4\ X^5]^{\mathit{tt}}$
$\mathrm{Prim}(\mathcal{C}_6,\mathcal{GB}^{\neg}_{o\iota})$, CNF:	$\mathcal{C}_7: [sk^2\ (sk^1\ (\lambda Z_\iota.\neg(X^6\ Z)))\ X^5]^{\mathit{ff}} \vee [X^6\ X^5]^{\mathit{ff}}$
$\mathrm{Fac}(\mathcal{C}_7)$, UNI, CNF:	$\mathcal{C}_8: [sk^2\ \big(sk^1\ \lambda Z_\iota.\neg(sk^2\ Z\ Z)\big)\ \big(sk^1\ \lambda Z_\iota.\neg(sk^2\ Z\ Z)\big)]^{\mathit{ff}}$
$\mathrm{Para}(\mathcal{C}_4,\mathcal{C}_8)$, UNI, CNF:	$\mathcal{C}_9: [sk^2\ \big(sk^1\ \lambda Z_\iota.\neg(sk^2\ Z\ Z)\big)\ \big(sk^1\ \lambda Z_\iota.\neg(sk^2\ Z\ Z)\big)]^{\mathit{tt}}$
$\mathrm{Para}(\mathcal{C}_9,\mathcal{C}_8)$, Triv:	$\square$

This problem is part of the TPTP library as problem `SYO037^1.p` and could, up to the author's knowledge, not be solved by any existing HO ATP system. This is probably due to the fact that the above proof requires a sort of non-analytic variable instantiation that introduces the left inverse of f. In contrast, in the proof of the surjective Cantor theorem the clausification procedure explicitly introduces a Skolem functions that can be used to reason about the pre-image of a given function (ultimately leading to the contradiction). Leo-III is able to infer the existence of an left inverse to f by its special treatment of injective functions (cf. §4.2.5). Nevertheless, such an inverse function could also be postulated by using appropriate choice instantiations but still the generation of such an instance by a general purpose routine seems challenging in practice. The full proof of Leo-III is displayed in Appendix C.2.[3]

E3 *Pigeonhole principle.*
The well-known pigeonhole principle can be stated as follows: Suppose there are n holes and more pigeons than holes. If every pigeon is assigned a hole then there is at least one hole that is assigned to two pigeons. An exemplary instance is given by the following problem that assumes 4 pigeons and 3 holes. The original (propositional) formalization goes back to Cook [Coo76]. The formalization used here is due to Chad Brown.

[3] The exact invocation of Leo-III for generating this proof was ”`leo3 SYO037^1.p -t 60 -p --sos`”.

Let pigeon and hole be two new types and let p^i_{pigeon}, $1 \leq i \leq 4$, be the four distinct pigeons, i.e., it holds that $p^i \neq p^j$ whenever $i \neq j$. Let further h^j_{hole}, $1 \leq j \leq 3$, be three holes and $\text{ph}_{\text{hole(pigeon)}}$ the function assigning pigeons to holes. Given that $\forall P_{o(\text{hole})}. (P\, h^1 \wedge P\, h^2 \wedge P\, h^3) \longrightarrow \forall X_{\text{hole}}. P\, X$, denoted $\star$ in the following, it holds that there exist two different pigeons $X_{\text{pigeon}}, Y_{\text{pigeon}}$, $X \neq Y$, such that X and Y are assigned the same hole by ph. Note that, as pointed out further below, $\star$ already implies that there are at most three holes.
After pre-processing, the problem $\mathbf{E}_3$ consists of the following clauses:

$$\begin{array}{ll}
\mathcal{C}_0: & [\mathrm{X}^1\, h^1]^{\mathfrak{ff}} \vee [\mathrm{X}^1\, h^2]^{\mathfrak{ff}} \vee [\mathrm{X}^1\, h^2]^{\mathfrak{ff}} \vee [\mathrm{X}^1\, \mathrm{X}^2]^{\mathfrak{tt}} \\
\mathcal{C}_1: & [p^1 \simeq p^2]^{\mathfrak{ff}} \\
\mathcal{C}_2: & [p^1 \simeq p^3]^{\mathfrak{ff}} \\
\mathcal{C}_3: & [p^1 \simeq p^4]^{\mathfrak{ff}} \\
\mathcal{C}_4: & [p^2 \simeq p^3]^{\mathfrak{ff}} \\
\mathcal{C}_5: & [p^2 \simeq p^4]^{\mathfrak{ff}} \\
\mathcal{C}_6: & [p^3 \simeq p^4]^{\mathfrak{ff}} \\
\mathcal{C}_7: & [\mathrm{ph}\, \mathrm{X}^3 \simeq \mathrm{ph}\, \mathrm{X}^4]^{\mathfrak{ff}} \vee [\mathrm{X}^3 \simeq \mathrm{X}^4]^{\mathfrak{tt}}
\end{array}$$

A proof for $\mathbf{E}_3$ is found by Leo-III in approx. 40 s and consists of 172 inferences. Due to its large number of inferences required, the refutation is not displayed here.[4] However, it should be pointed out that the generated proof consists of various non-trivial reasoning steps, including:

- A large number of paramodulation and (pattern) unification inferences.
- A large number of equational simplification steps using (RP), (RN), (PSR) and (NSR).
- Instantiation of negation by (Prim) on $\mathcal{C}_0$, followed by (Fac) , producing
  $\mathcal{C}_8: [\mathrm{X}^6\, \mathrm{X}^5]^{\mathfrak{ff}} \vee [\mathrm{X}^6\, h^1]^{\mathfrak{tt}} \vee [\mathrm{X}^6\, h^2]^{\mathfrak{tt}} \vee [\mathrm{X}^6\, h^3 \simeq \neg(\mathrm{X}^6\, \mathrm{X}^5)]^{\mathfrak{ff}}$.
- Instantiation of primitive equality by (LEQ) on $\mathcal{C}_8$, (Triv) :
  $\mathcal{C}_9: [\mathrm{X}^7 \simeq h^1]^{\mathfrak{tt}} \vee [\mathrm{X}^7 \simeq h^2]^{\mathfrak{tt}} \vee [\mathrm{X}^7 \simeq h^3]^{\mathfrak{tt}}$
  Note that this clause follows from $\star$ alone and asserts that there exist at most three pigeonholes.
- Postulating the existence of a left-inverse function sk^1 of type pigeon(hole) to ph:
  $\mathcal{C}_{10}: [sk^1\, (\mathrm{ph}\, \mathrm{X}^8) \simeq \mathrm{X}^8]^{\mathfrak{tt}}$

[4] It can, however, be found at `www.alexandersteen.de/phd/` under section ”Application examples”.

> This function can then be used to derive a contradiction when assigning two pigeons to one hole while assuming that ph is injective.

As can be seen, the proof uses a broad variety of different inference procedures. The success of Leo-III for this example can be traced to the use of sophisticated equational simplification procedures that use positive and negative unit equations to reduce the search space. Furthermore, this proof exploits Leo-III's capability to recognize injective functions and, in turn, uses the so inferred existence of a corresponding left-inverse.
The above presented problem is part of the TPTP library as problem `MSC007^1.003.004.p`. It can currently not be solved by any system other than Leo-III.

E4 *Function approximation using if-then-else*
The following problem conjectures the existence of a function h such that $h(0) \equiv 1$, $h(1) \equiv 0$ and $h(2) \equiv 1$ using an axiomatization of if-then-else. The goal is to find the if-then-else term that expresses this function h. This problem is considered hard since it requires the ATP system to approximate step-by-step the correct if-then-else function that is functionally equivalent to h. It originates from the PhD studies of Martin Riener.[5]
Let ite be a constant of type $\iota\iota\iota o$ that denotes the conditional (if-then-else) operator. Let further be zero and s the constants representing the number zero (of type ι) and the successor function (of type $\iota\iota$), respectively. The problem $\mathbf{E}_4$ is then stated as follows:

$$\begin{aligned}&\big(\forall X_o, Y_\iota, Z_\iota.\ \ X \longrightarrow \mathrm{ite}\, X\ Y\ Z = Y \wedge \\ &\ \forall X_o, Y_\iota, Z_\iota.\, \neg X \longrightarrow \mathrm{ite}\, X\ Y\ Z = Z \wedge \\ &\ \forall N_\iota.\, \mathrm{s}\, N \neq N\big) \longrightarrow \\ &\ \exists H_{\iota\iota}.\, H\ \mathrm{zero} = \mathrm{s\ zero} \wedge H\ (\mathrm{s\ zero}) = \mathrm{zero} \wedge H\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) = \mathrm{s\ zero}\end{aligned} \tag{3}$$

Roughly speaking, an axiomatization of a conditional operator and a specification of the desired function h is given. An reasoning system can now use the axiomatization of ite to derive a concrete conditional term expressing h. Leo-III finds a proof for the above problem in approx. 6 s:

[5] The problem and its origin is described in more detail at Martin Riener's web site under `https://www.logic.at/staff/riener/hol-problems/`.

$\mathsf{CNF}(\neg\mathbf{E}_4)$:	$\mathcal{C}_0$: $[\mathrm{ite}\ \mathrm{X}^1\ \mathrm{X}^2\ \mathrm{X}^3 \simeq \mathrm{X}^2]^{tt} \lor [\mathrm{X}^1]^{ff}$;
	$\mathcal{C}_1$: $[\mathrm{ite}\ \mathrm{X}^4\ \mathrm{X}^5\ \mathrm{X}^6 \simeq \mathrm{X}^6]^{tt} \lor [\mathrm{X}^4]^{tt}$;
	$\mathcal{C}_2$: $[\mathrm{s}\ \mathrm{X}^7 \simeq \mathrm{X}^7]^{ff}$;
	$\mathcal{C}_3$: $[\mathrm{X}^8\ \mathrm{zero} \simeq \mathrm{s\ zero}]^{ff} \lor [\mathrm{X}^8\ (\mathrm{s\ zero}) \simeq \mathrm{zero}]^{ff} \lor$
	$[\mathrm{X}^8\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) \simeq \mathrm{s\ zero}]^{ff}$
$\mathrm{Para}(\mathcal{C}_0,\mathcal{C}_3),\mathsf{UNI}$:	$\mathcal{C}_4$: $[\mathrm{ite}\ (\mathrm{X}^9\ \mathrm{zero})\ (\mathrm{X}^{10}\ \mathrm{zero})\ (\mathrm{X}^{11}\ \mathrm{zero}) \simeq \mathrm{s\ zero}]^{ff} \lor$
	$[\mathrm{ite}\ (\mathrm{X}^9\ (\mathrm{s\ zero}))\ (\mathrm{X}^{10}\ (\mathrm{s\ zero}))\ (\mathrm{X}^{11}\ (\mathrm{s\ zero})) \simeq \mathrm{zero}]^{ff} \lor$
	$[\mathrm{X}^{10}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) \simeq \mathrm{s\ zero}]^{ff} \lor [\mathrm{X}^9\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big)]^{ff}$
$\mathrm{Para}(\mathcal{C}_0,\mathcal{C}_4),\mathsf{UNI}$:	$\mathcal{C}_5$: $[\mathrm{X}^{10}\ \mathrm{zero} \simeq \mathrm{s\ zero}]^{ff} \lor$
	$[\mathrm{ite}\ (\mathrm{X}^9\ (\mathrm{s\ zero}))\ (\mathrm{X}^{10}\ (\mathrm{s\ zero}))\ (\mathrm{X}^{11}\ (\mathrm{s\ zero})) \simeq \mathrm{zero}]^{ff} \lor$
	$[\mathrm{X}^{10}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) \simeq \mathrm{s\ zero}]^{ff} \lor [\mathrm{X}^9\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big)]^{ff} \lor$
	$[\mathrm{X}^9\ \mathrm{zero}]^{ff}$
$\mathrm{Prim}(\mathcal{C}_1,\mathcal{GB}_o^{\neg})$:	$\mathcal{C}_6$: $[\mathrm{ite}\ (\neg \mathrm{X}^{12})\ \mathrm{X}^{13}\ \mathrm{X}^{14} \simeq \mathrm{X}^{14}]^{tt} \lor [\mathrm{X}^{12}]^{ff}$
$\mathrm{Para}(\mathcal{C}_6,\mathcal{C}_5),\mathsf{UNI}$:	$\mathcal{C}_7$: $[\mathrm{X}^{10}\ \mathrm{zero} \simeq \mathrm{s\ zero}]^{ff} \lor [\mathrm{X}^{11}\ (\mathrm{s\ zero}) \simeq \mathrm{zero}]^{ff} \lor$
	$[\mathrm{X}^{10}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) \simeq \mathrm{s\ zero}]^{ff} \lor [\neg\big(\mathrm{X}^{15}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big)\big)]^{ff} \lor$
	$[\neg\big(\mathrm{X}^{15}\ \mathrm{zero}\big)]^{ff} \lor [\neg\big(\mathrm{X}^{15}\ (\mathrm{s\ zero})\big)]^{tt}$
$\mathsf{UNI}(\mathcal{C}_7),\mathsf{CNF}$:	$\mathcal{C}_8$: $[\mathrm{X}^{15}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big)]^{tt} \lor [\mathrm{X}^{15}\ \mathrm{zero}]^{tt} \lor [\mathrm{X}^{15}\ (\mathrm{s\ zero})]^{ff}$
$\mathrm{Fac}(\mathcal{C}_8),\mathrm{Triv}$:	$\mathcal{C}_9$: $[\mathrm{X}^{15}\ \mathrm{zero}]^{tt} \lor [\mathrm{X}^{15}\ (\mathrm{s\ zero})]^{ff} \lor$
	$[\neg\big(\mathrm{X}^{15}\ (\mathrm{s\ zero})\big) \simeq \mathrm{X}^{15}\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big)]^{ff}$
$\mathsf{LEQ}(\mathcal{C}_9),\mathsf{Simp}$,	
CNF:	$\mathcal{C}_{10}$: $[\mathrm{s\ zero} \simeq \mathrm{zero}]^{tt} \lor [\mathrm{s}\ (\mathrm{s\ zero}) \simeq \mathrm{s\ zero}]^{tt}$
$\mathrm{NSR}(\mathcal{C}_2,\mathcal{C}_{10})$:	$\mathcal{C}_{11}$: $[\mathrm{s}\ (\mathrm{s\ zero}) \simeq \mathrm{s\ zero}]^{tt}$
$\mathrm{NSR}(\mathcal{C}_2,\mathcal{C}_{11})$:	$\square$

In this refutation, an general instance of if-then-else is approximated by clause $\mathcal{C}_4$ using projection bindings and subsequently refined using primitive substitution (cf. clause $\mathcal{C}_6$) and further applications of paramodulation and unification.
The problem is part of the TPTP library as problem `NUN025^1.p`. The proof is displayed in Appendix C.3. Similar problems are `NUN023^1.p` and `NUN024^1.p`. They vary in the number of specified function input-output pairs. All of these variants can be solved by Leo-III in approx. 5 s while only the slightly simpler version (`NUN023^1.p`) can be solved by further ATP systems. There exist, however, even simpler variants of these three problems that additionally include a witness term. Here, the problem is merely to verify that the given witness term is indeed a correct solution to the problem. These versions can be solved by any current HO ATP system.

Choice problems. Increasingly many HOL ATP systems natively support reasoning with choice, that is, handling of an interpreted symbol $\varepsilon^{\tau}_{\tau(o\tau)}$, for each type τ, that chooses an arbitrary element satisfying a given predicate (if such an element exist). Leo-III adapted the choice handling from its predecessor and augmented the reasoning process with heuristic instantiation of free variables with special functions including choice. These instances can help to find refutations where otherwise the proof search would require deeply nested, complicated chains of primitive inferences.

E5 *Existence of choice.*

Let τ be a type. An instance of the axiom scheme of choice for type τ can be stated as follows:

$$\exists e_{\tau(o\tau)}.\, \forall P_{o\tau}.\, (\exists X_\tau.\, P\, X) \longrightarrow P\, (e\, P) \tag{4}$$

It postulates the existence of a function (the *choice function*) that, given a predicate $p_{o\tau}$, produces an element c_τ such that $p\ c$ – if such an object exists.

Of course, every ATP system that supports reasoning with choice should be able to prove this fundamental assumption for any given type τ. It is nevertheless included to demonstrate Leo-III's proof mechanisms for handling choice. The proof below exemplarily shows the validity of (4), the axiom of choice for objects of type ι. To that end, let $\mathbf{E}_5$ denote this choice instance. The introduced Skolem constants sk^1 and sk^2 are hereby of type $o\iota(\iota(o\iota))$ and $\iota(\iota(o\iota))$, respectively.

$\mathsf{CNF}(\neg\mathbf{E}_5)$:	$\mathcal{C}_0: [sk^1\ \mathrm{X}^1\ (sk^2\ \mathrm{X}^1)]^{\mathfrak{tt}}$;
	$\mathcal{C}_1: [sk^1\ \mathrm{X}^2\ (\mathrm{X}^2\ (sk^1\ \mathrm{X}^2))]^{\mathfrak{ff}}$
$\mathsf{ACD}(\mathcal{C}_1)$:	$\mathcal{C}_2: [sk^1\ \mathrm{X}^3\ \mathrm{X}^4]^{\mathfrak{ff}} \vee [sk^1\ \mathrm{X}^3\ (\varepsilon Z_\iota.\, sk^1\ \mathrm{X}^3\ Z)]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_2,\mathcal{C}_1)$:	$\mathcal{C}_3: [sk^1\ \mathrm{X}^3\ \mathrm{X}^4]^{\mathfrak{ff}} \vee$
	$[sk^1\ \mathrm{X}^3\ (\varepsilon Z_\iota.\, sk^1\ \mathrm{X}^3\ Z) \simeq sk^1\ \mathrm{X}^2\ (\mathrm{X}^2\ (sk^1\ \mathrm{X}^2))]^{\mathfrak{ff}}$
$\mathsf{UNI}(\mathcal{C}_3)$:	$\mathcal{C}_4: [sk^1\ \varepsilon^\iota\ \mathrm{X}^4]^{\mathfrak{ff}}$
$\mathrm{Para}(\mathcal{C}_0,\mathcal{C}_4)$:	$\mathcal{C}_5: [sk^1\ \mathrm{X}^1\ (sk^2\ \mathrm{X}^1) \simeq sk^1\ \varepsilon^\iota\ \mathrm{X}^4]^{\mathfrak{ff}}$
$\mathsf{UNI}(\mathcal{C}_5)$:	$\square$

The key inference here is that the subterm $\mathrm{X}^2\ (sk^1\ \mathrm{X}^2)$ of clause $\mathcal{C}_1$ is used for the application of (ACD) to produce the necessary concrete AC instance.

This problem is part of the TPTP library as problem `SYN997^1.p` and can be solved by most (but not all) current HO ATP systems. The full proof of Leo-III is displayed in Appendix C.4.

E6 *Special choice function*

A more challenging problem regarding choice functions is given in the following: There is an operator that chooses an element *not* satisfying the supplied predicate, if such an element exists. It is formalized quite similarly to the previous problem:

$$\exists e_{\iota(o\iota)}.\, \forall P_{o\iota}.\, (\exists X_\iota.\, \neg P\, X) \longrightarrow \neg P\, (e\, P) \tag{5}$$

In order to solve this problem, an instance combining a choice operator with negation needs to be inferred. Since this includes non-trivial primitive substitutions, the problem is more difficult to solve than the version further above.
Making use of its heuristic instantiation routine during pre-processing, Leo-III can solve this problem quite easily. Let, for that purpose, $\mathbf{E}_6$ denote the formula from equation (5) above. The introduced Skolem constants sk^1 and sk^2 are hereby of type $o\iota$ and ι, respectively.

$$\begin{array}{ll}
\text{HeuInst}(\neg\mathbf{E}_6): & \mathcal{C}_0: [\neg\forall P_{o\iota}.\, (\exists X_\iota.\, \neg P\, X) \longrightarrow \neg P\, (\varepsilon Z_\iota.\, \neg P\, Z)]^{\mathtt{tt}} \\
\mathsf{CNF}(\mathcal{C}_0): & \mathcal{C}_1: [sk^1\, sk^2]^{\mathtt{ff}}; \\
 & \mathcal{C}_2: [sk^1\, \left(\varepsilon Z_\iota.\, \neg(sk^1\, Z)\right)]^{\mathtt{tt}} \\
\mathsf{ACD}(\mathcal{C}_2), \mathsf{CNF}: & \mathcal{C}_3: [sk^1\, \mathrm{X}^1]^{\mathtt{ff}} \vee [sk^1\, \left(\varepsilon Z_\iota.\, \neg(sk^1\, Z)\right)]^{\mathtt{ff}} \\
 & \vdots \\
 & \square
\end{array}$$

The rest of the proof is then analogous to the proof of (4) from example *E5* above. The key aspect within the given proof of Leo-III is the instantiation of the universally quantified variable (referred to by e in (4)) by $\lambda X_\iota.\varepsilon Z_\iota.\, \neg(X\, Z)$. This instantiation is generated by Leo-III using its heuristic instantiation routine as described in §4.2.
This problem is part of the TPTP library as problem `SY0548^1.p`. It was studied in the context of Tableau calculi for higher-order logic [Bac10]. Other state-of-the-art systems including LEO-II, Isabelle/HOL and Zipperposition are currently not able to solve this problem. The full proof of Leo-III is displayed in Appendix C.5.

Function synthesis. Problems conjecturing the existence of functions with given properties are often very difficult for current HOL ATPs as they cannot be solved with sophisticated unification alone but regularly require the addition of suitable choice terms to the search space.

Leo-III features a novel procedure that allows the synthesis of simple functions based on given input-output specifications (cf. §4.2). Although this procedure may dramatically increase the search space, there are some problems that can be solved using this technique that could not be solved before by any ATP system.

E7 *Swapping function*
A simple yet challenging problem for current ATP systems is given by the following conjecture:

$$\forall X_\iota, Y_\iota . \exists f_{\iota\iota} . (f\, X = Y) \land (f\, Y = X) \tag{6}$$

This formula, denotes $\mathbf{E}_7$ in the following, postulates the existence of a function $f_{\iota\iota}$ that, given two elements a_ι, b_ι, swaps its input. More formally, it holds that $f\ a = b$ and $f\ b = a$.
Leo-III can make use of its synthesis procedure for functional specifications and, as a consequence, generates the following proof where sk^1 and sk^2 are again Skolem functions introduced by Leo-III, each of type ι:

$\mathsf{CNF}(\neg\mathbf{E}_7)$, LiftEq:	$\mathcal{C}_0 : [\mathrm{X}^1\ sk^1 \simeq sk^2]^{ff} \lor [\mathrm{X}^1\ sk^2 \simeq sk^1]^{ff}$
Fac$(\mathcal{C}_0)$, UNI:	$\mathcal{C}_1 : [sk^1 \simeq sk^2]^{ff}$
FuncSpec$(\mathcal{C}_0)$:	$\mathcal{C}_2 : [\varepsilon Z_\iota . \big(Z = sk^2 \land (sk^1 = sk^2 \longrightarrow Z = sk^1)\big) \simeq sk^2]^{ff} \lor$ $[\varepsilon Z_\iota . \big((sk^2 = sk^1 \longrightarrow Z = sk^2) \land Z = sk^1\big) \simeq sk^1]^{ff}$
$\mathsf{ACD}(\mathcal{C}_2)$, CNF, UNI:	$\mathcal{C}_3 : [\varepsilon Z_\iota . \big(Z = sk^2 \land (sk^1 = sk^2 \longrightarrow Z = sk^1)\big) \simeq sk^2]^{tt} \lor$ $[sk^2 \simeq sk^1]^{tt}$
Para$(\mathcal{C}_3, \mathcal{C}_1)$, Triv:	$\mathcal{C}_4 : [\varepsilon Z_\iota . \big(Z = sk^2 \land (sk^1 = sk^2 \longrightarrow Z = sk^1)\big) \simeq sk^2]^{tt}$
Para$(\mathcal{C}_4, \mathcal{C}_2)$, Triv:	$\mathcal{C}_5 : [\varepsilon Z_\iota . \big((sk^2 = sk^1 \longrightarrow Z = sk^2) \land Z = sk^1\big) \simeq sk^1]^{ff}$
$\mathsf{ACD}(\mathcal{C}_2)$, CNF, UNI:	$\mathcal{C}_6 : [\varepsilon Z_\iota . \big((sk^2 = sk^1 \longrightarrow Z = sk^2) \land Z = sk^1\big) \simeq sk^1]^{tt} \lor$ $[sk^2 \simeq sk^1]^{tt}$
Para$(\mathcal{C}_6, \mathcal{C}_1)$, Triv:	$\mathcal{C}_7 : [\varepsilon Z_\iota . \big((sk^2 = sk^1 \longrightarrow Z = sk^2) \land Z = sk^1\big) \simeq sk^1]^{tt}$
Para$(\mathcal{C}_7, \mathcal{C}_5)$, Triv:	$\square$

The specification of the desired function is supplied by clause $\mathcal{C}_0$. It is used to generate the concrete function description satisfying the specification, if existent. The hereby produced instance is given by $\lambda X . \varepsilon Z_\iota . \big((X = sk^1 \longrightarrow Z = sk^2) \land (X = sk^2 \longrightarrow Z = sk^1)\big)$. Using this instance, and the fact that $sk^1 \neq sk^2$, the contradiction is then inferred by using the choice instantiation procedure described further above and a number of subsequent paramodulation steps.
This problem is part of the TPTP library as problem `SYO519^1.p`. This problem remained unsolved since TPTP v4.1.0 (released approx. in 2009) and can now be solved by Leo-III. The full proof of Leo-III is displayed in Appendix C.6.

E8 *Function specification*

Example *E4* postulated the existence of a function with a given specification using a formulation of an if-then-else operator. Although this problem is hard as it is, it technically still includes redundant information that should not be necessary for an ATP system to find a proof to the conjecture. This is because the HOL semantics targeted by Leo-III is Henkin with choice (which is also assumed by default by the TPTP THF standard). Hence, the validity of the problem should not be depending on the explicit postulation of an conditional operator ite (cf. (3)) since such a function can easily defined via choice.

Consequently, an even more reduced (and complicated) problem can be formulated analogously to $\mathbf{E}_4$, however leaving out the axioms of the conditional operator:

$$\begin{aligned}&\forall N_\iota .\, \mathrm{s}\, N \neq N \longrightarrow \\ &\quad \exists H_{\iota\iota} .\, H\ \mathrm{zero} = \mathrm{s\ zero} \wedge H\ (\mathrm{s\ zero}) = \mathrm{zero} \wedge H\ \big(\mathrm{s}\ (\mathrm{s\ zero})\big) = \mathrm{s\ zero}\end{aligned} \tag{7}$$

Let $\mathbf{E}_8$ denote this modified variant of the problem.

Leo-III solves $\mathbf{E}_8$ in under 4 s, even faster than the presumably simpler variant from example *E4*. Since the proof is again quite long and rather technical, it is not surveyed here in full length. The functional specification derived from Leo-III's synthesis method is analogous to the one described in example *E7*. The full proof output of Leo-III is displayed at Fig. C.7.

5.2. Polymorphic Higher-Order Reasoning

Another application of Leo-III is reasoning in rank-1 polymorphic higher-order logic. Leo-III is compatible with the recent TF1 and TH1 extensions of first-order and higher-order syntax, respectively, from TPTP. Two problems are exemplarily presented in this section. A comprehensive evaluation of Leo-III's reasoning capabilities in polymorphic HOL is subsequently analyzed in §6.2.

E9 *Cantor's Theorem, revisited*

The use of polymorphism allows more general and more reusable formalizations of knowledge. In this example, Cantor's (surjective) Theorem from example *E1* is generalized to not depend on a certain type that is used to specify the type of the sets and functions. Consequently, the type ι is replaced by a type variable α, using an explicit type quantification.

The remaining formalization is then analogous to problem $\mathbf{E}_1$ from example *E1*:

$$\forall\alpha.\, \neg\exists f_{o\alpha\alpha}.\, \forall Y_{o\alpha}.\, \exists X_{\alpha}.\, f\, X = Y \tag{8}$$

A proof of this conjecture is generated by Leo-III in less than 3 s and is similar to the monomorphic one. A new aspect here is the generation of a fresh Skolem type skt that serves as a witness to the abstract type variable α. As a consequence, the introduced Skolem symbols sk^1 and sk^2 are now of type $o(\mathrm{skt})(\mathrm{skt})$ and $(\mathrm{skt})(o(\mathrm{skt}))$, respectively. After clausification, the proof is identical to the previous one.
The full proof of Leo-III is displayed in Appendix C.8. The injective case is analogous and therefore omitted here.

E10 *Finiteness of Booleans*
Another interesting problem originates from the HOL Light core library [Har09]. It introduces the notion of sizes of sets, relates it to finiteness and then postulates the finiteness of the set of Booleans.
The problem is formalized as follows[6]: Let num denote a type for (natural) numbers. Let further hasSize, finite, card be three constants of type $\forall\alpha.o(\mathrm{num})(o\alpha)$, $\forall\alpha.o(o\alpha)$ and $\forall\alpha.\,\mathrm{num}(o\alpha)$, respectively. Intuitively, hasSize denotes a predicate that, given a set S and a number n, is true iff S is finite (represented by finite) and has a cardinality (represented by card) equal to n. Let additionally univ be a polymorphic constant symbol of type $\forall\alpha.o\alpha$ that represents the universe of all objects of the given type argument. Using this symbol, the size of the universe of Booleans is postulated to have size 2. Formally, the relevant facts for this problems are encoded as two axioms A^1 and A^2:

$$A^1 := \forall\alpha.\forall X_{o\alpha}, Y_{\mathrm{nat}}.\, \mathrm{hasSize}\;\alpha\; X\; Y = (\mathrm{finite}\;\alpha\; X \wedge \mathrm{card}\;\alpha\; X = Y)$$
$$A^2 := \mathrm{hasSize}\; o\; (\mathrm{univ}\; o)\; 2$$

The problem $\mathbf{E}_{10}$ is then encoded as follows:

$$(A^1 \wedge A^2) \longrightarrow \mathrm{finite}\; o\; (\mathrm{univ}\; o) \tag{9}$$

Leo-III finds a proof for this problem in under 2 s. The generated proof is straight-forward but requires multiple applications of type unification for

[6] For reasons of legibility, the problems' formulation is slightly simplified.

paramodulation inferences between clauses with (free) type variables.

$$\begin{array}{lll}
\mathsf{CNF}(\neg\mathbf{E}_{10}): & \mathcal{C}_0: & [(\text{finite}\ \alpha^1\ \mathrm{X}^1) \wedge (\text{card}\ \alpha^1\ \mathrm{X}^1 = \mathrm{X}^2) \simeq \text{hasSize}\ \alpha^1\ \mathrm{X}^1\ \mathrm{X}^2]^{tt}; \\
 & \mathcal{C}_1: & [\text{hasSize}\ o\ (\text{univ}\ o)\ 2]^{tt}; \\
 & \mathcal{C}_2: & [\text{finite}\ o\ (\text{univ}\ o)]^{ff} \\
\text{PBE}(\mathcal{C}_0): & \mathcal{C}_3: & [(\text{finite}\ \alpha^1\ \mathrm{X}^1) \wedge (\text{card}\ \alpha^1\ \mathrm{X}^1 = \mathrm{X}^2)]^{tt} \vee [\text{hasSize}\ \alpha^1\ \mathrm{X}^1\ \mathrm{X}^2]^{ff} \\
\mathsf{CNF}(\mathcal{C}_3): & \mathcal{C}_4: & [\text{finite}\ \alpha^1\ \mathrm{X}^1]^{tt} \vee [\text{hasSize}\ \alpha^1\ \mathrm{X}^1\ \mathrm{X}^2]^{ff} \\
\text{Para}(\mathcal{C}_1,\mathcal{C}_3), & & \\
\mathsf{TyUNI}: & \mathcal{C}_5: & [\text{finite}\ o\ \mathrm{X}^1]^{tt} \vee [\text{hasSize}\ o\ (\text{univ}\ o)\ 2 \simeq \text{hasSize}\ o\ \mathrm{X}^1\ \mathrm{X}^2]^{ff} \\
\mathsf{UNI}(\mathcal{C}_5): & \mathcal{C}_6: & [\text{finite}\ o\ (\text{univ}\ o)]^{tt} \\
\text{Para}(\mathcal{C}_6,\mathcal{C}_2), & & \\
\text{Triv}: & \square &
\end{array}$$

This problem is part of the most recent TPTP library as problem `SEV485^1.p`. The full proof of Leo-III is displayed in Appendix C.9.

5.3. Modal Logic Reasoning

The semantical embedding approach allows the employment of common HOL ATP systems for reasoning in (and about) many different modal logics. The utilization of an ATP system for that purpose involves, however, quite technical embedding procedures that need to be performed prior to the reasoning process. As described in §4.7, Leo-III offers an automation of this embedding process such that it can be applied to modal logic problems without any extra effort or technical issues. In this section, this claim is supported by giving several modal example problems and outlining how Leo-III handles these problems. Modal logic K with rigid constants and constant domains as well as a global consequence relation is assumed if not stated otherwise.

E11 *Barcan formula*
One of the most prominent controversial consequence of the possibilists's interpretation of modal logic semantics (constant domain quantification semantics) is the validity of all instances of the Barcan Formula (BF) [Men16]. For type ι, the instances are given by

$$\forall X_\iota . \square p\, X \longrightarrow \square \forall X_\iota . p\, X \qquad (10)$$

where $p_{o\iota}$ is some predicate symbol. As an example, the validity of (BF) gives so-called *Possibilia* debatable traits, e.g., that the mere possibility of the existence of an object with some given property implies the actual existence of those object having possibly that property. On a model theoretic

level, there is a strong connection between (BF) and properties of the respective frame classes in which it is valid. In particular, (BF) is valid if and only if the modal logic frames are non-increasing (a detailed discussion can be found in relevant literature on modal logic [FM98]).
Modal logic problems can be formalized using the proposed modal THF input language as described in §4.7. The above (BF) instance can, for example, be encoded as follows:

```
(! [X:$i]: ($box @ (p @ X))) => ($box @ (! [X:$i]: (p @ X)))
```

Let $\mathbf{E}_{11}$ denote the Barcan Formula instance (10) in the context of decreasing domain semantics. Let further $\lceil . \rceil$ be the embedding operator, as sketched in §4.7, that encodes modal logic problems into equivalent HOL problems. Leo-III can prove $\mathbf{E}_{11}$ in under 3 s.[7] The proof is presented below. Let, as introduced before, the bold identifiers (e.g. $\boldsymbol{\forall}$) denote the respectively lifted connectives or constant sybmols of the modal logic problem. In order to clarify the embedding process, the expansion of the embedded modal logic formula into the final HOL formula is displayed step-by-step as equation $\star$ below the refutation. The auxiliary symbols valid, $\boldsymbol{\Box}$, etc. that are used by the embedding process have been introduced further above (cf. §4.7). The symbols sk^1, sk^2 and sk^3 are Skolem symbols of type μ, μ and ι, respectively.

$\mathsf{Embed}(\lceil \neg \mathbf{E}_{11} \rceil)$:	$\mathcal{C}_0: [r\,\mathrm{X}^1\,\mathrm{X}^2 \longrightarrow (\mathrm{eiw}^\iota\,\mathrm{X}^3\,\mathrm{X}^2 \longrightarrow \mathrm{eiw}^\iota\,\mathrm{X}^3\,\mathrm{X}^1)]^{\mathfrak{tt}}$;
	$\mathcal{C}_1: [\mathrm{valid}\,\big(\boldsymbol{\forall X_\iota}.\boldsymbol{\Box}\,(\boldsymbol{p\,X}) \boldsymbol{\longrightarrow} \boldsymbol{\Box}\,(\boldsymbol{\forall X_\iota}.\boldsymbol{p\,X})\big)]^{\mathfrak{ff}}$
	$\overset{\star}{\equiv} [\forall W_\mu.\big(\forall X_\iota.(\mathrm{eiw}^\iota\,X\,W) \longrightarrow \forall V_\mu.(r\,W\,V \longrightarrow p\,X\,V)\big)$
	$\longrightarrow \forall V_\mu.\big((\mathrm{r}\,W\,V) \longrightarrow \forall X_\iota.(\mathrm{eiw}^\iota\,X\,V) \longrightarrow p\,X\,V\big)]^{\mathfrak{ff}}$
$\mathsf{CNF}(\mathcal{C}_0)$:	$\mathcal{C}_2: [r\,\mathrm{X}^1\,\mathrm{X}^2]^{\mathfrak{ff}} \vee [\mathrm{eiw}^\iota\,\mathrm{X}^3\,\mathrm{X}^2]^{\mathfrak{ff}} \vee [\mathrm{eiw}^\iota\,\mathrm{X}^3\,\mathrm{X}^1]^{\mathfrak{tt}}$
$\mathsf{CNF}(\mathcal{C}_1)$:	$\mathcal{C}_3: [p\,sk^3\,sk^2]^{\mathfrak{ff}}$;
	$\mathcal{C}_4: [\mathrm{eiw}^\iota\,sk^3\,sk^2]^{\mathfrak{tt}}$;
	$\mathcal{C}_5: [r\,sk^1\,sk^2]^{\mathfrak{tt}}$;
	$\mathcal{C}_6: [\mathrm{eiw}^\iota\,\mathrm{X}^4\,sk^1]^{\mathfrak{ff}} \vee [r\,sk^1\,\mathrm{X}^5]^{\mathfrak{ff}} \vee [p\,\mathrm{X}^4\,\mathrm{X}^5]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_4,\mathcal{C}_2),\mathsf{UNI}$:	$\mathcal{C}_7: [r\,\mathrm{X}^1\,sk^2]^{\mathfrak{ff}} \vee [\mathrm{eiw}^\iota\,sk^3\,\mathrm{X}^1]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_5,\mathcal{C}_6),\mathsf{UNI}$:	$\mathcal{C}_8: [\mathrm{eiw}^\iota\,\mathrm{X}^4\,sk^1]^{\mathfrak{ff}} \vee [p\,\mathrm{X}^4\,sk^2]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_7,\mathcal{C}_8),\mathsf{UNI}$:	$\mathcal{C}_9: [r\,sk^1\,sk^2]^{\mathfrak{ff}} \vee [p\,sk^3\,sk^2]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_5,\mathcal{C}_9),\mathrm{Triv}$:	$\mathcal{C}_{10}: [p\,sk^3\,sk^2]^{\mathfrak{tt}}$
$\mathrm{Para}(\mathcal{C}_{10},\mathcal{C}_3),\mathrm{Triv}$:	$\Box$

[7] The Leo-III options used are: `--assume-modal-domains decreasing --assume-modal-system K -p`. Rigid constants and global consequence are used by default, if not overridden using further command-line arguments.

The proof uses straight-forward paramodulation and unification procedures. Note, however, that the initial formula $\mathbf{E}_{11}$ is translated by $\lceil . \rceil$ into multiple formulas that not only contain the lifted equivalent of (BF) but also additional meta-logical information about the semantic context. The clause $\mathcal{C}_0$ here represents the fact that the quantification semantics assumes decreasing domains (cf. §4.7 for a more detailed explanation). Without this information, the refutation could not be found.
The expansion of clause $\mathcal{C}_1$ by $\star$ is justified by the embedding process of modal logic formulas into HOL. In this example, the embedding proceeds as follows:

$$
\begin{aligned}
(\star)\quad & \text{valid}\,\big(\forall \boldsymbol{X_\iota}. \Box\,(\boldsymbol{p\,X}) \longrightarrow \Box\,(\forall \boldsymbol{X_\iota}.\boldsymbol{p\,X})\big)\\
&\equiv \text{valid}\,\big((\lambda W_\mu.\forall X_\iota.\text{eiw}^\iota\;X\;W \longrightarrow \Box\,(p_{o\mu\iota}\;X)\;W) \longrightarrow \Box\,(\forall \boldsymbol{X_\iota}.\boldsymbol{p\,X})\big)\\
&\equiv \text{valid}\,\big((\lambda W_\mu.\forall X_\iota.\text{eiw}^\iota\;X\;W \longrightarrow \Box\,(p\,X)\;W)\\
&\qquad\qquad \longrightarrow \Box\,(\lambda W_\mu.\forall X_\iota.\text{eiw}^\iota\;X\;W \longrightarrow p\,X\,W)\big)\\
&\equiv \text{valid}\,\big(\lambda W_\mu.(\forall X_\iota.\text{eiw}^\iota\;X\;W \longrightarrow \Box\,(p\,X)\;W)\\
&\qquad\qquad \longrightarrow \Box\,(\lambda W_\mu.\forall X_\iota.\text{eiw}^\iota\;X\;W \longrightarrow p\,X\,W)\;W\big)\\
&\equiv \text{valid}\,\Big(\lambda W_\mu.\big(\forall X_\iota.(\text{eiw}^\iota\;X\;W) \longrightarrow \forall V_\mu.(r\;W\;V \longrightarrow\;(p\,X\,V))\big)\\
&\qquad\qquad \longrightarrow \forall V_\mu.\big((r\;W\;V) \longrightarrow \forall X_\iota.(\text{eiw}^\iota\;X\;W) \longrightarrow p\,X\,W\big)\Big)\\
&\equiv \forall W_\mu.\big(\forall X_\iota.(\text{eiw}^\iota\;X\;W) \longrightarrow \forall V_\mu.(r\;W\;V \longrightarrow\;(p\,X\,V))\big)\\
&\qquad\qquad \longrightarrow \forall V_\mu.\big((r\;W\;V) \longrightarrow \forall X_\iota.(\text{eiw}^\iota\;X\;W) \longrightarrow p\,X\,W\big)
\end{aligned}
$$

Note that $p_{o\mu\iota}$ represents the embedded variant of the HOML symbol $p_{o\iota}$ and has hence a more complex type that the original predicate symbol. It is, after embedding, also dependent on a possible world of type μ at which it is evaluated.
The Barcan Formula instance is part of the QMLTP library [RO12] as problem `SYM001+1.p`. The full proof of Leo-III is displayed in Appendix C.10.

E12 *Converse Barcan Formula*
The counterpart to the previous modal logic example is the so-called Converse Barcan Formula (CBF). For type ι, the instances are given by

$$\Box\forall X_\iota.pX \longrightarrow \forall X_\iota.\Box pX \tag{11}$$

where $p_{o\iota}$ is some predicate symbol. There exists an equally strong connection to modal model theory as for (BF): (CBF) is valid in a modal frame if and only if its domains are non-decreasing.

Similar to (BF), Leo-III can use its embedding mechanism to generate a proof for the above instance. Due to the inherent flexible nature of the embedding approach, the semantic context can easily be adapted for this example to use cumulative domain semantics (instead of decreasing domain semantics as in *E11*).[8]

Let $\mathbf{E}_{12}$ denote the above instance of the Converse Barcan Formula (11) in the context of cumulative domain semantics. Let again $\lceil . \rceil$ denote the embedding operator. The intermediate embedding steps are analogous to *E11* and therefore omitted. Leo-III finds a proof for this problem in under 3 s. The symbols sk^1, sk^2 and sk^3 are Skolem symbols of type μ, ι and μ, respectively.

$$\begin{array}{ll}
\mathsf{Embed}(\lceil \neg \mathbf{E}_{12} \rceil): & \mathcal{C}_0 : [\forall W_\mu.,V_\mu.,X_\iota.\, r\, W\, V \longrightarrow (\mathrm{eiw}^\iota\, X\, X \longrightarrow \mathrm{eiw}^\iota\, X\, V)]^{\mathtt{tt}}; \\
& \mathcal{C}_1 : [\forall W_\mu.\, \big(\forall V_\mu.\, \big((\mathrm{r}\, W\, V) \longrightarrow \forall X_\iota.\, (\mathrm{eiw}^\iota\, X\, V) \longrightarrow p\, X\, V\big) \\
& \qquad \longrightarrow \forall X_\iota.\, (\mathrm{eiw}^\iota\, X\, W) \longrightarrow \forall V_\mu.\, (r\, W\, V \longrightarrow p\, X\, V)\big)]^{\mathtt{ff}} \\
\mathsf{CNF}(\mathcal{C}_0): & \mathcal{C}_2 : [r\, \mathrm{X}^1\, \mathrm{X}^2]^{\mathtt{ff}} \vee [\mathrm{eiw}^\iota\, \mathrm{X}^3\, \mathrm{X}^1]^{\mathtt{ff}} \vee [\mathrm{eiw}^\iota\, \mathrm{X}^3\, \mathrm{X}^2]^{\mathtt{tt}} \\
\mathsf{CNF}(\mathcal{C}_1): & \mathcal{C}_3 : [p\, sk^2\, sk^3]^{\mathtt{ff}}; \\
& \mathcal{C}_4 : [\mathrm{eiw}^\iota\, sk^2\, sk^1]^{\mathtt{tt}}; \\
& \mathcal{C}_5 : [r\, sk^1\, sk^3]^{\mathtt{tt}}; \\
& \mathcal{C}_6 : [\mathrm{eiw}^\iota\, \mathrm{X}^5\, \mathrm{X}^4]^{\mathtt{ff}} \vee [r\, sk^1\, \mathrm{X}^4]^{\mathtt{ff}} \vee [p\, \mathrm{X}^5\, \mathrm{X}^4]^{\mathtt{tt}} \\
& \vdots \\
& \square
\end{array}$$

The proof is similar to the one of $\mathbf{E}_{11}$ with the difference that now $\mathcal{C}_0$ postulates cumulative domain semantics. After clausification, the refutation continues analogously to *E11*.

The Converse Barcan Formula instance is part of the QMLTP library as problem `SYM002+1.p`. The full proof of Leo-III is displayed at Appendix C.11.

5.4. Leo-III as a Meta-Prover

The architecture of Leo-III allows for an employment as a so-called *meta-prover*. Meta provers are systems that typically do little to none reasoning on their own, but instead invoke different theorem proving systems (possibly with different parameter settings) and collect their results. The use of meta-prover systems is practically justified because different ATP systems have different strengths and

[8] The Leo-III options used are: `--assume-modal-domains cumulative --assume-modal-system K -p`.

weaknesses (in particular, when being based on different proof calculi), and often perform particularly well on certain classes of problems while being outperformed by other provers on other classes of problems, and vice versa.

As described in §4.4, Leo-III is designed to integrate external reasoners into its own internal proof procedure. While this integration was primarily designed to incorporate first-order provers (and, possibly, further specialist reasoners), higher-order systems can also be included into the cooperation scheme. Hence, when including various (first- and higher-order) external reasoning systems Leo-III can be run purely as a meta system coordinating these provers and translating proof obligations for them, similar to Isabelle's Sledgehammer [BN10b]. Additionally, axiom selection methods and pre-processing techniques of Leo-III can be used prior to invocation of the external systems. Since Leo-III is not primarily designed as meta prover, the cooperation mechanisms for this kind of employment are not as optimized as in dedicated meta prover systems. As an example, sophisticated strategy scheduling or parameter selection schemes are missing in Leo-III.

6. Evaluation

In order to quantify the performance of Leo-III in the context of current state-of-the-art higher-order ATP systems, an extensive evaluation based on various benchmarks is presented.[1] Special attention is given to the identification of the impact of Leo-III's paramodulation calculus as opposed to the resolution-based approach used by its predecessor LEO-II.

For the evaluation, three benchmark data sets are used:

- *TPTP TH0* (2463 problems) is the set of all monomorphic higher-order (TH0) problems from the TPTP library [Sut17b] v7.0.0 that are annotated as theorems. The TPTP problem library is a de-facto standard for the evaluation of ATP systems and is also used as benchmark selection basis for the yearly CADE ATP System Competitions (CASC) [Sut16b]. The problems originate from various problem domains and sources. The TPTP THF problems in particular fostered the development of current higher-order ATPs, including LEO-II and Satallax.
- *TPTP TH1* (442 problems) is the subset of all rank-1 polymorphic higher-order (TH1) problems from the TPTP library v7.0.0 that are annotated as theorems and do not contain arithmetic. The problems consist mainly of HOL Light [Har09] core exports and Sledgehammer translations of various Isabelle theories [JM05].
- *QMLTP* (580 problems) is the subset of all mono-modal benchmarks from the QMLTP library [RO12] (version 1.1). Since each problem may have a different validity status for different semantics of modal logic, all mono-modal problems (and not only those marked Theorem) are selected. More precisely, for native modal logic reasoners, the original problem statements from the QMLTP are used. For Leo-III, the benchmark data results from embedding the QMLTP problems into modal THF as described in §4.7.3.

The evaluation measurements were taken on the StarExec cluster [SST14] in which currently each compute node is a 64 bit Red Hat Linux (kernel 3.10.0) machine featuring a 2.40 GHz Intel Xeon quad-core processors and a main memory of 128 GB.[2] For each problem, every prover was given a CPU time limit of

[1] All raw measurement data can be accessed at `http://alexandersteen.de/phd/`.

[2] See `https://www.starexec.org/starexec/public/about.jsp` for more information.

Table 1: Detailed result of the 2463 TPTP TH0 benchmark measurements

Systems	**Solutions**		**SZS Results**				**Avg. Time** [s]		**Σ Time** [s]	
	Abs.	**Rel.**	**THM**	**CAX**	**GUP**	**TMO**	**CPU**	**WC**	**CPU**	**WC**
Leo-III	2053	83.39	2045	8	15	394	15.39	5.61	31490	11508
Satallax 3.2	2140	86.89	2140	0	2	321	12.26	12.31	26238	26339
Satallax 3.0	1972	80.06	2028	0	2	433	17.83	17.89	36149	36289
LEO-II	1788	72.63	1789	0	43	603	5.84	5.96	10452	10661
Zipperpos.	1318	53.51	1318	0	360	785	2.60	2.73	3421	3592
Isabelle/HOL	0	0	2022	0	1	440	46.46	33.44	93933	67610

240 s (CASC default time limit) and unrestricted memory usage.

Depending on the benchmark data set, different ATP systems are used for comparison. This is simply due to the fact that not all ATP systems are applicable for all different data sets. For the *QMLTP* data set, native modal logic ATP systems are also used. The following theorem provers are employed in one or more measurements: Isabelle/HOL 2016 (TH0/TH1), Satallax 3.0 (TH0), Satallax 3.2 (TH0), LEO-II 1.7.0 (TH0), Leo-III 1.2 (TH0/TH1/QMLTP), Zipperposition 1.1 (TH0), MleanCoP 1.3 (QMLTP).

The evaluation results are presented separately for each benchmark data set in the following. The results for TPTP TH0, TPTP TH1 and QMLTP are presented in §6.1, §6.2 and §6.3, respectively. Additionally, in §6.4, the quality of Leo-III's proofs is assessed on the basis of the case study problems from §5.

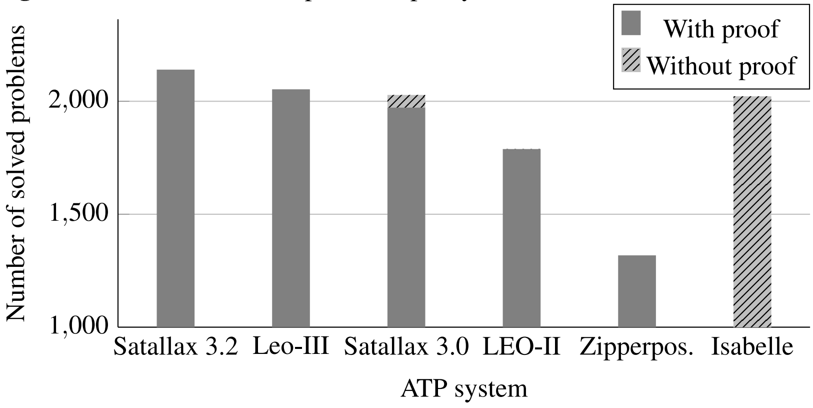
6.1. Higher-Order Reasoning

The *TPTP TH0* benchmark suite seems suitable as a primary indicator for the overall performance analysis of HOL ATP systems. The collection of THF problem is meanwhile quite diverse and contains problems from various application domains. The effectiveness of Leo-III on this problems set is compared against other state-of-the-art ATP systems in terms of problems solved and runtime. Also, the impact of external cooperation with first-order provers on Leo-III's reasoning capabilities is analyzed in more detail.

One problem (`SYN036^7.p`) has been excluded from the benchmark data set since it is flawed in the sense that its include directives are disorganized and hence cause errors in all tested ATP systems.

System Performance. Table 1 displays each system's performance on the TPTP TH0 data set. For each system the absolute number (Abs.) and relative share (Rel.) of solved problems is displayed. Solved here means that a system is

Figure 1: Number of solved problems per system for the TPTP TH0 data set



able to establish the SZS status Theorem[3] and also emits a proof certificate that substantiates this claim.[4] All results of the system, whether successful or not, are counted and categorized as THM (Theorem), CAX (ContradictoryAxioms), GUP (GaveUp) and TMO (TimeOut) for the respective SZS status of the returned result.[5] Additionally, the average and sum of all CPU times and wall clock (WC) times over all solved problems is presented.

Using external first-order cooperation Leo-III successfully solves 2053 of 2463 problems (roughly 83.39 %) from the TPTP TH0 data set. This is 735 (35.8 %) more than Zipperposition, 264 (12.86 %) more than LEO-II and 81 (3.95 %) more than Satallax 3.0. The only ATP system that solves more problems is the most recent version of Satallax (version 3.2) that solves 2140 problems, which is approximately 4.24 % more than Leo-III. Even if solutions without explicit proofs are counted, Leo-III would still have slightly higher number of solved problems than Satallax 3.0 and Isabelle/HOL with 25 (1.22 %) and 31

[3] More precisely, an ATP system can establish the SZS result Theorem or any of its sub-value according to the SZS ontology. In particular, the SZS result ContradictoryAxioms is a more specialized kind of Theorem and is hence counted towards solved problems.

[4] Similar restrictions are also implemented at the CADE ATP system competition (CASC).

[5] Note that GaveUp is used here as representative for similar SZS results that indicate that an ATP system terminated on its own accord before exceeding the resource limits (e.g. the timeout) without establishing a useful result. ATP system results of type Unknown and Inappropriate are therefore counted under GaveUp. Similarly, the SZS results indicating a termination due to resource constraints (ResourceOut, MemoryOut and TimeOut) are simply referred to as TimeOut throughout the evaluation.

(1.51 %) additional solutions, respectively.

The novel feature of Leo-III that allows to discover inconsistent problem statements (cf. §4.5) takes effect on the data set: Of the solutions found by Leo-III, eight problems are identified as SZS result CAX (ContradictoryAxioms) and are hence shown to contain inconsistent axioms. Without external cooperation, Leo-III can identify 14 inconsistent problems (cf. discussion of external cooperation below) in the data set.

The number of solved problems per ATP systems is graphically compared in Fig. 1. Evidently, Satallax and Leo-III are currently the most powerful ATP systems on the TPTP TH0 data set. Of course, if one does not require proof certificates, Isabelle/HOL is comparably powerful. Maybe somewhat surprisingly, Zipperposition is able to solve more than one half of all problems despite being essentially a first-order ATP system that has only recently been augmented with some higher-order reasoning mechanisms.

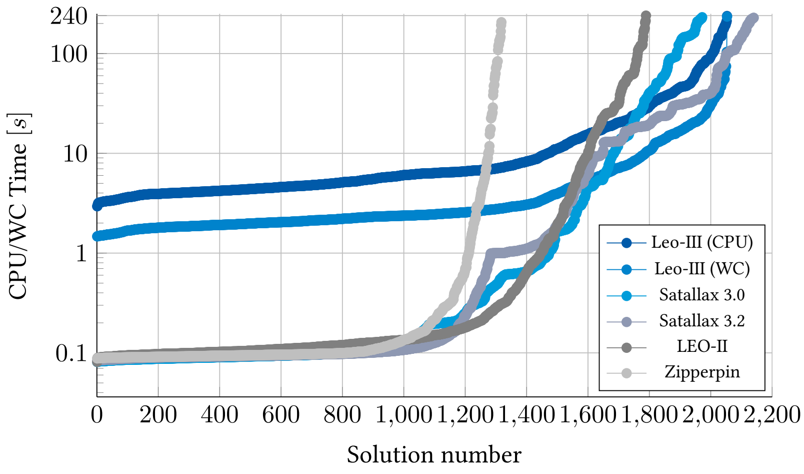
Unsurprisingly, each of the systems seems to have its particular strengths on certain types of problems and can, as a consequence, solve problems that no other system is able to solve (*unique solutions*). LEO-II, Leo-III, Satallax (3.2) and Zipperposition produce three, 18, 17 and 15 unique solutions, respectively. Evidently, Leo-III currently produces more unique solutions than any other ATP system in this setting. It is remarkable that Zipperposition, despite its quite recent enhancement to higher-order logic, produces as many as 15 unique solutions. An in-depth analysis of this situation seems beneficial for the remaining systems.

Additionally to the strong benchmark results discussed above, Leo-III solves nine unsolved problems and three problems that can currently not be solved by any other ATP system. Here, unsolved means that no ATP system has ever been able to establish an SZS success value at any TPTP version release (rating 1.0). This information is extracted from the TPTP problem rating that is attached to each problem. These problems cover a range of different applications, including natural language processing (via embedding to modal logic), reasoning in intuitionistic logic (likewise via embedding) and set theory.[6]

Among the ATP systems, Leo-III shows the largest discrepancy between CPU time and WC time on average and sum over all solved problems (cf. Table 1). This is due to the non-blocking asynchronous cooperation with first-order ATP systems employed during proof search. Here, the external provers are started as independent processes on the operating system level and can hence be assigned

[6] The exact, previously unsolved, problems are NLP004^7, SET013^7, SEU558^1, SEU683^1, SEV143^5, SYO037^1, SYO062^4.004, SYO065^4.001 and SYO066^4.004. The other three problems that can currently not be solved by any other ATP system are MSC007^1.003.004, SEU938^5 and SEV106^5.

Figure 2: Performance graphs of various ATP systems for data set TPTP TH0



to different CPU cores. Satallax, LEO-II and Zipperposition, in contrast, show only small differences between CPU and WC time on average and sum. A more precise measure for a system's utilization of multiple cores is the so-called *core usage*. It is given by the average of the ratios of CPU time to used wall clock time over all solved problems. The core usage of Leo-III for the TPTP TH0 data set is roughly 2.52. This means that, on average, two to three CPU cores are used during proof search by Leo-III. Satallax (3.2), LEO-II and Zipperposition show a quite opposite behavior with core usages of 0.64, 0.56 and 0.47, respectively.

An interesting result of Leo-III's high core usage can be seen at Fig. 2 that displays performance graphs for each ATP system. Here, it can be seen that the CPU time used by Leo-III degrades similar to the other provers whereas the real time spent, i.e. the wall clock time, gradually approaches those of the remaining systems and even outperforms them after approx. 1600 problems. For a scope of 400 problems, Leo-III is the fastest (with respect to WC time) prover, and then outperformed by Satallax 3.2. However, as can be seen in particular for the first 1200 problems, Leo-III suffers from the Java Virtual Machine back end as the execution platform imposes a constant start-up delay of approximately one second. Hence, even the simplest problem takes at least one second to be solved by Leo-III while easily being solved by Satallax, LEO-II and Zipperposition in under 10 ms. It is unclear if these drawbacks can be mitigated using recent ahead-of-time compilers for Java-based languages or similar techniques.

Table 2: Comparison of LEO-II and Leo-III on 1917 equational resp. 645 non-equational problems

	TH0 NoEq			**TH0 Eq**		
	LEO-II	Leo-III	Rel. Change [%]	LEO-II	Leo-III	Rel. Change [%]
No cooperation	285	410	43.86	537	1150	114.15
With cooperation	441	459	4.08	1348	1586	17.66

Equational Reasoning. As pointed out before, one primary goal of the Leo-III project is to augment the RUE-based reasoning approach of LEO-II with a dedicated and, hopefully, more effective way of handling (primitive) equality. In order to measure the advantages of Leo-III's underlying paramodulation calculus, the TPTP TH0 data set is divided into two disjoint sets: *TH0 Eq* (1917 problems) and *TH0 NoEq* (645 problems) that contain all problems from TPTP TH0 which contain equality resp. do not contain equality. Here, containing equality means that at least one syntactical occurrence of the interpreted TPTP symbol "=" is contained in the problem. Note that, in a higher-order setting, this classification is not as precise as in a first-order context. This is due to the fact that equality may occur between terms of every type including those of Boolean type (formulas). Hence, every simple non-equational problem can easily be syntactically expressed using equality symbols. Also, during proof search, equality predicates may get instantiated by primitive substitution even if the original problem did not contain any primitive equality. Additionally, there are infinitely many ways of defining equality relations in HOL. As a consequence, recognizing equality within HOL problems can only be an under-approximation. The simple classification proposed here is thus intended only as a coarse approximation and certainly leaves room for optimization and further discussion.

The employment of external cooperation again tampers with the evaluation precision of the effectiveness of a system's implemented calculus. In order to reduce this distortion, the measurements were additionally conducted using both LEO-II and Leo-III without any first-order prover cooperation.

Table 2 displays the number of solutions produced by LEO-II and Leo-III on the two separated data sets, each with external first-order cooperation enabled and without any cooperation. Using external cooperation, LEO-II solves 441 (68.37 %) non-equational problems and 1348 (70.32 %) equational problems. Leo-III on the other hand is able to solve 459 (71.16 %) and 1586 (82.73 %) problems, respectively. While this shows an increase in reasoning performance from LEO-II to Leo-III of approximately 4.1 % for non-equational problems, the increase of reasoning effectivity of Leo-III on equational problems is approx. 17.7 % and thereby more than four times as much as for non-equational problems.

With first-order cooperation disabled[7], LEO-II (Leo-III) solves 285 (410) of the non-equational problems and 537 (1150) of the equational ones. This corresponds to roughly 44.2 % (64.7 %) and 28.0 % (60.0 %) of the problems of the respective problem class. In this setting, Leo-III solves dramatically more problems than LEO-II with an increase of 43.9 % for non-equational problems and 115.2 % for equational problems. As can be seen, Leo-III is unconditionally stronger than LEO-II on any kind of input (cf. discussion on the influence of external cooperation further below). Apart from the general improved strength of Leo-III, the increase in reasoning effectivity for equational problems in particular is greater roughly by factor 2.5 and still considerably higher than for non-equational problems.

In both evaluation scenarios, that is, reasoning with and without external cooperation, it can be seen that Leo-III particularly improved its equational reasoning capabilities compared to LEO-II albeit having improved in general as well. Hence, there is strong evidence that the underlying paramodulation calculus of Leo-III seems fit for practical employment in ATP systems.

Impact of External Cooperation. Leo-III is designed as a cooperating ATP system. However, in principle, it does not require any external (first-order) reasoning system for functioning. In this section, the actual impact of external cooperation on Leo-III's reasoning strength is quantified. To that end, the above measurements on the TPTP TH0 data set are repeated for different combinations of external reasoners used in conjunction with Leo-III. More specifically, the first-order reasoners examined here are E 2.1, CVC4 1.5, iProver 2.6 and Vampire 4.1.

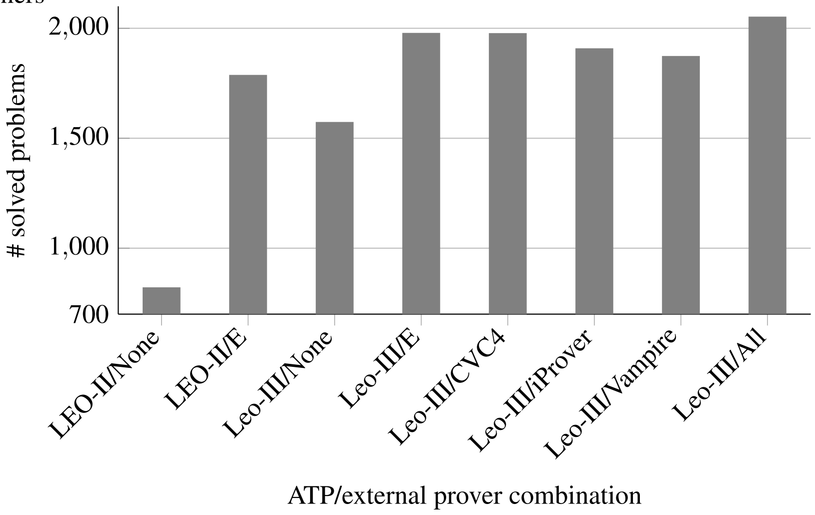
Also, for comparison, LEO-II is benchmarked with and without cooperation with E. The remaining higher-order ATP systems cannot be separated from their subsystems (e.g., Satallax using MiniSat and E).

Figure 3 shows the number of problems solved for each combination of external reasoner cooperation. As an example, the label "Leo-III/E" denotes a configuration in which Leo-III was used in conjunction with E. The remaining combinations are analogous. "Leo-III/none" denotes that Leo-III used no external cooperation at all. "Leo-III/all" represents the combination from previous experiments in which Leo-III cooperates with all the first-order reasoners.

LEO-II heavily depends on first-order cooperation; it solves only 822 problems without using E and 1789 problems when using E. This is due to the fact

[7] External cooperation needs to be enabled explicitly in Leo-III. On the other hand, LEO-II by default requires E to be available on the system. First-order cooperation can be disabled within LEO-II using the "`-f none`" option.

Figure 3: Performance of LEO-II/Leo-III with different external first-order reasoners



that LEO-II is designed to do only as much higher-order reasoning as necessary (e.g., relevant applications of primitive substitution or extensionality) and dispatch the remaining reasoning tasks to E. In contrast, the aim of Leo-III has been to strengthen the higher-order RUE calculus so that state-of-the-art techniques from first-order reasoning can be applied to higher-order reasoning as well. As a consequence, while external cooperation still has a great influence on Leo-III's reasoning strength, it is not as critical as for LEO-II: Internally, Leo-III is able to solve 1574 (63.93 %) problems which is 47.78 % more than LEO-II without using E, and still 16.26 % more than those of Zipperposition.

The most effective cooperating systems are E and CVC4 with which Leo-III solves 1979 and 1978 problems, respectively. When used in combination, Leo-III yields 2031 solutions. This suggests that both system have, in some aspects, orthogonal strengths and weaknesses; and can both convincingly contribute to Leo-III's reasoning process. The remaining two first-order provers, iProver and Vampire, perform slightly worse with 1909 and 1874 solved problems, respectively. Also, they do not add more solutions when used in addition to E and CVC4 (cf. ”Leo-III/all” with 2053 solutions), with iProver and Vampire contributing ten and twelve additional solutions, respectively.

Currently, the portion of CAX results strongly decreases when using external cooperation. This is because the externally generated (first-order) proofs are not analyzed by Leo-III which implies that it is not known which facts the first-order provers actually used within their proofs. As a consequence, the proof reconstruction over-approximates the derivation and assumes that all submitted facts were used in the proof, including the (negated) conjecture. This naive handling of external proofs impairs the effectivity of the inconsistency detection (cf. 4.5). Only refutations that are found faster by internal reasoning than by external reasoners can, at the moment, contribute to CAX solutions. This situation could be amended by inspecting the external proofs but remains future work.

6.2. Polymorphic Higher-Order Reasoning

For the second measurement, Leo-III was benchmarked on the THF TH1 data set that contains rank-1 polymorphic HOL problems (cf. §4.6). In contrast to the TPTP TH0 problems, the collection of polymorphic problems is quite young (started in 2017 with the release of TPTP v7.0.0), and is therefore not yet as large and diverse; there are 666 problems in total, of which 662 are theorems. The subset of these TPTP TH1 problems considered here is those 442 problems that are theorems and do not include arithmetic. The set consists of problems from the HOL Light core (205 problems) and of Sledgehammer-generated translations of various AFP Isabelle theories (237 problems) [KNP03]. The latter problems, in particular, regularly consist of a large number of axioms of which often only few are relevant to the conjecture, and thus pose yet another challenge to the ATP systems apart from reasoning in polymorphic HOL itself.

Currently, there exist only few other systems that are capable of reasoning within polymorphic HOL as specified by TPTP TH1. They are HOL(y)Hammer [KU15b] and the `tptp_isabelle` mode of Isabelle/HOL [NWP02], which schedules proof tactics within HOL Light [Har09] and Isabelle/HOL, respectively. Unfortunately, only Isabelle/HOL has been available for instrumentation in a reasonable recent and stable version for a comparison with Leo-III.[8]

Table 3 displays the measurement results for the TPTP TH1 data set. As before, for both ATP systems (and, in the case of Leo-III, each external cooperation system configuration) the absolute and relative number of solved problems, the detailed SZS return values and the average and overall time consumption is displayed. Again, the SZS result values GUP and TMO are representatives for similar return values (cf. footnote 5) and solutions are only counted towards solved

[8] A reasonably current version of HOL(y)Hammer is available at the SystemOnTPTP web interface. However, the system always returns the SZS result "GaveUp" regardless of the input problem.

Table 3: Detailed result of the 442 TPTP TH1 benchmark measurements

Systems	**Solutions**		**SZS Results**				**Avg. Time** [s]		**Σ Time** [s]	
	Abs.	**Rel.**	**THM**	**CAX**	**GUP**	**TMO**	**CPU**	**WC**	**CPU**	**WC**
Leo-III 1.2										
— internal	87	19.68	83	4	14	341	29.79	16.86	2592	1467
— E	162	36.65	160	2	10	270	33.27	16.31	5391	2642
— CVC4	161	36.43	158	3	9	272	41.08	22.22	6615	3577
— iProver	144	32.58	141	3	11	287	36.66	20.63	5279	2971
— Vampire	137	31.00	133	4	12	293	37.63	21.31	5155	2919
— all	185	41.86	183	2	8	249	49.18	24.93	9099	4613
Isabelle/HOL	0	0	237	0	23	182	93.53	81.44	22404	19300

problems if a proof certificate is provided. As a consequence, Leo-III is trivially the best (and only) ATP system for TPTP TH1 in this scenario. Nevertheless, as can be seen from the detailed breakdown of SZS result values, Isabelle/HOL finds 237 theorems (53.62 %) which is roughly 28.1 % more than the number of solutions founds by Leo-III. Even if the cooperation of Leo-III is restricted to only use E (CVC4), it still solves 162 (161) problems which is more than one third of all problems and approximately 31.6 % less than Isabelle/HOL. Also, Leo-III solves 35 unique problems whereas Isabelle/HOL solves 69 unique problems. With no external cooperation ("— internal" in Table 3), Leo-III only solves one fifth of all problems. Up to four problems are identified by Leo-III to contain inconsistent axioms and, hence, trivially validate their conjecture.

There are two components of Leo-III that considerably hinder its reasoning effectivity in the TPTP TH1 data set context: Firstly, the relevance filter (axiom selector) of Leo-III is quite naive and often only provides a weak filtering effect for large knowledge bases. As a consequence, Leo-III's reasoning strength degrades rather quickly if the number of axioms grows. Since the TPTP TH1 data set consists of approx. one half Sledgehammer exports that typically include numerous additional axioms, Leo-III's weak relevance filter has considerable influence on its success rate. In the TPTP TH0 data set, in contrast, the distribution of problem sizes is not as biased towards large problems as in the TH1 data set. Secondly, the monomorphization algorithm is not as optimized and mature as in Isabelle/HOL. Monomorphization is used in regular intervals for encoding proof obligations into monomorphic first-order logic during proof search. Its current implementation is rather inefficient and severely slows down the reasoning loop thus reducing the effective inference rate of Leo-III.

Isabelle/HOL prematurely aborts the reasoning process for 23 problems because of internal type errors, seemingly caused by faulty treatment of type quantifications. The exact cause of these errors has not yet been identified. Leo-III, on the other hand, fails for up to eight problems due to internal typing errors during proof search. The remaining six premature terminations are due to incompleteness of the used search mode. These observations suggest that TH1 reasoning within Isabelle/HOL and Leo-III is not yet as stable as for the common monomorphic input dialects and more effort needs to be invested in order to offer reliable and mature reasoning capabilities for polymorphic inputs.

6.3. Modal Logic Reasoning

The last measurement of this chapter quantifies the performance of Leo-III on first-order modal logic problems from the QMLTP library [RO12]. The QMLTP library is, up to the author's knowledge, the only uniform collection of first-order modal logic problems currently available. The current release (version 1.1) contains 600 problem statements, of which 421 contain first-order quantification, and 20 being multi-modal problems. The QMLTP problem set consists mainly of problems from textbooks on modal logic, from the TABLEAUX-2000 Non-Classical (Modal) System Comparisons (TANCS-2000) [MD00], and from Goedel translations of intuitionistic problems into modal logic [Göd69].

The relevant subset of QMLTP problems considered for the evaluation are those 580 problems that are mono-modal. In contrast to the previous measurements, not only problems marked as theorems are used. This is due to the fact that the measurements are conducted for multiple semantic settings and, for each of these settings a problem's validity status might be different. As a consequence, for every semantics a different set of problems would be used for the evaluation. In order to present an uniform evaluation framework for all semantic settings, counter-satisfiable and open problems are thus also included. Likewise, the multi-modal problems are excluded from the measurements since they are restricted to use cumulative domains and particular axiomatizations.

Leo-III is compared against MleanCoP version 1.3 [Ott14]. MleanCoP is a powerful representative of native first-order modal logics provers for modal logics **D**, **T**, **S4** and **S5** and multi-modal logics. Earlier experiments confirm that MleanCoP is currently the most effective ATP system for its supported modal logics [BOR12]. Note that, for comparability reasons, rigid constants and local consequence are used throughout this evaluation (cf. §4.7.2). This is because the QMLTP library assumes these settings and MleanCoP supports only such semantics. In contrast, Leo-III is by no means limited to these semantic variants. The evaluation is conducted using the original QMLTP problem statements in the case of MleanCoP. For Leo-III, the input problems are the result of translating the QMLTP problem statements to modal THF syntax.[9] As a result, technically, both systems are evaluated on different data sets. However, the transformation merely translates first-order-like application syntax to their higher-order equivalents (using explicit application operators) and adds type declarations for each symbol.

The measurement results for Leo-III and MleanCoP on the QMLTP data set are displayed in Tables 4a and 4b, respectively. As before, for both systems the absolute number of solved problems, a breakdown of all result values as well as both the average and sum of CPU and wallclock time over each solved problem is given. The measurements were taken for all semantic combinations produced by (assuming a local consequence relation):

- a modal axiomatization from $\{\mathbf{K},\mathbf{D},\mathbf{T},\mathbf{S4},\mathbf{S5}\}$,
- quantification semantics from $\{\mathit{vary},\mathit{decr},\mathit{cumul},\mathit{const}\}$, where *vary*, *decr*, *cumul*, *const* denote varying domains, decreasing domains, cumulative domains and constant domains, and
- rigid constant symbols.

The resulting twelve combinations are denoted by the respective combination of the first two components, e.g. **S5**/*vary* for a **S5** modal logic with varying domains, rigid constants and local consequence relation. Furthermore, the detailed breakdown of SZS results now also displays counter-satisfiability (CSA) results as not only theorems are included in the data set.

As can be seen from the measurements results, MleanCoP overall performs better in all semantic settings that it supports. Nevertheless, as can be seen from earlier evaluations [BOR12, GSB17], Leo-III's performance is still significantly better than that of other first-order modal logic provers or HOL reasoning backends used in combination with the embedding approach. Additionally, the average wallclock time used by Leo-III for solving the problems is, for most semantic settings, fairly close to the time used by MleanCoP.

Fig. 4 displays a comparison of both systems for all twelve semantics that are supported by MleanCoP. Leo-III is fairly competitive with MleanCoP (weaker by maximally 14.05 %, minimally 2.95 % and 8.90 % on average) for all **D** and **T** variants. For all **S4** variants, the gap between both systems increases (weaker by maximally 20.00 %, minimally 13.66 % and 16.18 % on average). For **S5** variants Leo-III is again very effective (stronger by 1.36 % on average) and it is ahead of MleanCoP for **S5**/const and **S5**/cumul. This is due to encoding of the

[9] The translated collection of QMLTP problems is available at `www.github.com/TobiasGleissner/QMLTP`.

Table 4: Detailed result of the 580 QMLTP benchmark measurements

(a) Leo-III

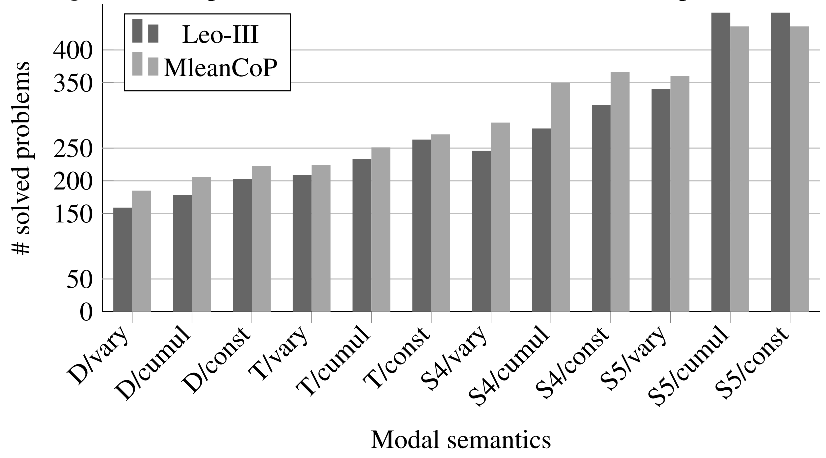
Semantics	**Solutions**	**SZS Results**				**Avg. Time** [s]		**Σ Time** [s]	
		THM	**CSA**	**GUP**	**TMO**	**CPU**	**WC**	**CPU**	**WC**
K/vary	150	150	0	94	336	20.30	8.97	3045	1345
K/decr	162	165	0	30	388	19.41	8.55	3144	1385
K/cumul	166	166	0	30	384	18.73	8.24	3109	1367
K/const	189	189	0	83	308	19.53	8.62	3692	1628
D/vary	159	159	0	76	345	19.80	8.62	3149	1371
D/decr	174	174	0	19	387	18.72	8.16	3257	1418
D/cumul	178	178	0	19	383	19.40	8.30	3453	1477
D/const	203	203	0	64	313	18.60	8.00	3776	1624
T/vary	209	209	0	73	298	19.52	8.22	4079	1718
T/decr	223	223	0	22	335	19.07	7.94	4254	1771
T/cumul	233	233	0	22	325	20.28	8.21	4724	1914
T/const	263	263	0	59	258	18.87	7.88	4964	2073
S4/vary	246	246	0	0	334	21.84	9.26	5372	2278
S4/decr	264	264	0	0	316	20.75	8.78	5477	2317
S4/cumul	280	280	0	0	300	23.07	9.34	6459	2615
S4/const	316	316	0	0	264	21.00	8.74	6635	2762
S5/vary	340	340	0	77	163	23.29	10.42	7920	3542
S5/decr	456	456	0	48	75	23.34	10.17	10644	4639
S5/cumul	456	456	0	48	75	23.34	10.17	10644	4639
S5/const	456	456	0	48	75	23.34	10.17	10644	4639

(b) MleanCoP

Semantics	**Solutions**	**SZS Results**				**Avg. Time** [s]		**Σ Time** [s]	
		THM	**CSA**	**GUP**	**TMO**	**CPU**	**WC**	**CPU**	**WC**
K/ —	†								
D/vary	185	185	271	20	104	7.72	8.45	1429	1564
D/decr	†	†				†		†	
D/cumul	206	206	245	21	108	6.36	7.12	1311	1467
D/const	223	223	219	19	119	6.24	6.82	1392	1520
T/vary	224	224	157	33	166	4.16	5.01	938	1122
T/decr	†	†				†		†	
T/cumul	251	251	131	26	172	2.80	3.65	703	916
T/const	271	271	113	24	172	3.10	3.95	840	1070
S4/vary	289	289	126	25	140	9.66	10.36	2791	2995
S4/decr	†	†				†		†	
S4/cumul	350	350	95	22	113	9.11	9.82	3188	3435
S4/const	366	366	82	21	111	9.37	10.07	3431	3687
S5/vary	360	360	93	16	111	5.01	5.79	1805	2084
S5/decr	†	†				†		†	
S5/cumul	436	436	40	17	87	4.46	5.25	1946	2289
S5/const	436	436	40	26	78	4.42	5.21	1927	2270

†: MleanCoP does not support the semantics

Figure 4: Comparison of Leo-III and MleanCoP on solved problems



S5 accessibility relation in Leo-III 1.2 as the universal relation between possible worlds as opposed to its prior encoding [BOR12, GSB17] as an equivalence relation.

One possible reason for this performance degradation in stronger modal logic systems such as **S4** is that during the embedding process more axioms (restrictions on the accessibility relation) are included to the translated problem. In particular, for modal logic **S4** a transitivity axiom is included for the accessibility relation. This axiom might increase the search space of the embedded problem, and in turn renders the proof search less effective. In modal logic **S5** such an effect would also appear if not mitigated by Leo-III by using an optimized translation that is specific to the nature of **S5** modal semantics. Here, the accessibility relation is usually regarded an equivalence relation [Gar16] (e.g., reflexive and euclidean). However, without affecting the proof-theoretic semantics of the system, the accessibility relation can also be postulated the universal relation [BvBW06] and can be eliminated completely from the embedding. This greatly simplifies the embedded modal logic formulas and makes Leo-III more effective on **S5** semantic variants.

In the QMLTP benchmark set, Leo-III solves 199 previously unsolved problems. This includes 49, 29, and 38 new solutions for the semantics **K**/*vary*, **K**/*const* and **K**/*cumul*, respectively, approximately 13 new solutions for every **S5** variant and roughly four to six new solutions for every remaining semantics

Table 5: Verification results for the nine case study problems from §5 that can be verified using GDV. For each problem, the time used by GDV for verifying the correctness of the proof is printed.

Problem	**Verification time** [s]
`SET557^1`	440
`SY0037^1`	804
`MSC007^1.003.004`	4099
`NUN025^1`	1286
`SYN997^1`	469
`SY0548^1`	576
`SY0519^1`	747
modified `NUN025^1` (Ex. *E8*)	1071
`SEV485^1`	470

supported by QMLTP. The extraordinarily high number of new solutions for variants of modal logic **K** is due to the fact that only few of the modal logic provers indexed by QMLTP support reasoning in these logics.

6.4. Proof Certificates

In this section, the quality of proof certificates produced by Leo-III is assessed. For that purpose, the proofs of Leo-III for the twelve problems from §5 are studied for verifiability by GDV [Sut06]. For the verification benchmarks GDV is used in conjunction with Isabelle/HOL and Nitpick as higher-order reasoning systems for proving and refuting the intermediate conclusions, respectively. The results of the verification attempts are presented in Table 5. GDV is able to verify eight out of the twelve proofs generated by Leo-III as correct. It can be seen that the necessary time for verifying each proofs is by magnitudes larger than the original time needed by Leo-III for solving the problem. This is due to the fact no semantic information contained in the proof is utilized by GDV for the verification. GDV merely invokes a reproving process each step of the produced derivation.

There are two reasons why three Leo-III proofs could not be verified in this way: Firstly, the proof of the polymorphic variant of the surjective Cantor Theorem from §5.2 contains a universal type quantification in the conjecture. It seems that the verification of such an conjecture is currently not supported by GDV using the Isabelle/HOL backend. Secondly, the two proofs of the modal logic reasoning examples from §5.3 contain non-trivial Skolemization. Note that this operation is satisfiability preserving but not an equivalence transformation. The verification of this kind of logical relationship between inference steps seems currently supported to only a limited degree by GDV. As a consequence, even comparably simple CNF transformations that include Skolemization impede GDV's verification capabilities.

Overall, the analysis presented here shows that the proofs of Leo-III are informative enough to enable proof verification, even in the case of the comparably simple GDV tool. This suggests that more sophisticated proof verification techniques which utilize semantical information about the individual proofs steps can be effectively applied to Leo-III proofs.

7. Conclusion and Outlook

The first part of this thesis has presented an extensional higher-order paramodulation calculus that treats equality as primitive rather than as defined notion. This calculus is shown to be sound and complete with respect to Henkin semantics. To that end, a model existence theorem that rests on a formulation of abstract consistency properties is presented, which slightly simplifies earlier variants prensented by Benzmüller et al. Moreover, the abstract consistency classes are not assumed to be saturated.

In the second part of this thesis an implementation of the paramodulation calculus is presented, forming the new higher-order ATP system Leo-III. Due to its wide range of natively supported classical and non-classical logics, including polymorphic HOL and numerous first- and higher-order modal logics, the system has many topical applications in computer science, AI, maths and philosophy. In particular, Leo-III features

- native reasoning in rank-1 polymorphic HOL,
- reasoning in almost every normal higher-order modal logic,
- support for all common TPTP dialects (CNF, FOF, TFF, THF) including their polymorphic variants (TF1, TH1),
- flexible cooperation with any TFF- or THF-compliant ATP,
- full compatibility with the standardized SZS result ontology, and
- detailed/verifiable TSTP-compatible proof certificates.

These hybrid logic capabilities make Leo-III, up to the author's knowledge, the most widely applicable ATP system available. Additionally, an extensive evaluation of Leo-III shows that it is also one of the most effective HO ATP systems to date. These results also answer, to some degree, the question whether it is generally feasible to employ a paramodulation-based calculus for higher-order logic.

Further Work. There is much room for further improvements and extensions of Leo-III and its underlying calculus. Extensive research in the context of first-order theorem proving systems indicates that both a solid theoretical foundation and also carefully developed implementation techniques are necessary in order to yield a practically effective reasoning tool.

On the theoretical level, more work needs to be invested in augmenting the calculus EP with appropriate techniques for restricting its inferences in such a way that more sophisticated notions of redundancy can be employed. Obviously, adaptions of existing powerful mechanisms from first-order superposition would be desirable. Unfortunately, due to the inherently more complex nature of the term language of HOL there seems to be no straight-forward adaption of such techniques. A potentially fruitful path for further work in this direction is to investigate whether existing higher-order term orderings such as CPO (as employed by Leo-III) can be used to safely restrict the paramodulation inferences of EP. Since small improvements on the calculus level might greatly influence a system's performance, it seems worthwhile to study if weaker conditions than those of first-order superposition can be employed while retaining completeness, e.g. adaptions of ordered paramodulation or even weaker forms of ordering constraints. This is also motivated by the observation that considerable effort needs to be invested for constructing a complete superposition calculus for a logic that seems only marginally more expressive than standard first-order logic [BBCW18].

Another path is to formally prove completeness of a polymorphic variant of EP which actually forms the basis of Leo-III. Such an investigation seems to be merely technical but an important result for the overall correctness of Leo-III.

Practically relevant future work includes

- a formal study and more thorough evaluation of the higher-order adaption of feature vector indexing employed by Leo-III,
- the development of a more effective subsumption procedure using a subsumption index based on higher-order substitution trees as sketched in earlier work [LS16],
- an extensive study and evaluation of further clause selection heuristics,
- an improved generation of proof certificates for proofs that were found in conjunction with external first-order provers, and
- a more sophisticated and fine-grained control of external reasoning systems using per-system decisions on encoding details (e.g. the concrete choice of an elimination technique for λ-abstraction).

While the latter two points are rather aimed at improving the Leo-III system itself, the first three points are relevant for higher-order ATP systems in general. As pointed out in the context of first-order theorem proving, the use of good clause selection strategies is one of the most important factor for the practical effectiveness of an ATP system. One would expect that a corresponding study of selection heuristics (that in particular focus on certain higher-order features) could have an similar effect in the higher-order case. Additionally, currently there exist almost no general purpose indexing mechanisms for HOL ATP systems and the development of such indexing techniques puts more emphasis on the comparably neglected field of practical implementation techniques for higher-order reasoning systems. In particular, it seems that the development of a subsumption indexing structure might profit from most recent work in higher-order matching modulo associativity and commutativity [CK18].

A. Installation and Usage of Leo-III

A.1. Requirements

Leo-III requires the Java 1.8 Runtime (JRE) for execution. Leo-III works on any common operating system (including Windows, Mac OS and Linux derivatives). However, external cooperation so-far only works on Linux and Mac systems. If Windows is used, it might be possible to try running Leo-III using Cygwin or similar for exploiting external cooperation.

A.2. Installation

Using pre-built binaries

A current (pre-built) release of Leo-III 1.2 can be downloaded from GitHub under `https://github.com/leoprover/Leo-III/releases/`. Note that the binaries available at GitHub were built on a Debian-based system and might not work for all Linux derivatives. If the pre-built version does not work, consider building Leo-III from source. Its quite simple and only takes a minute or two.

Building from source

The following requirements (dependencies) are not managed by the SBT build tool and hence need to be present at the system prior to building Leo-III:

- Java JDK 1.8
- make (any reasonably current version)
- SBT, Scala Build Tool (version 1.x)
- gcc (any reasonably current version)

Note that gcc is only required if you want to build Leo-III with support for external cooperation (the way Leo-III is intended to be used and works best). If you want to build Leo-III without capabilities for external cooperation, gcc is not needed. Leo-III will still be a fully functional higher-order ATP system, you just cannot increase its reasoning effectivity using external reasoners.

Leo-III uses SBT for building the Scala sources.[1] SBT will download an appropriate version of Scala (and further dependencies) automatically. The ac-

[1]See `http://www.scala-sbt.org` for the SBT tool and further details.

tual build process in invoked by `make`. Proceed as follows to build Leo-III from source:

1. Download the source distribution of the latest stable version (here: 1.3) from GitHub and unpack the archive:

```
> wget https://github.com/leoprover/Leo-III/archive/v1.3.tar.gz
> tar -xvzf v1.3.tar.gz
```

2. Step into the newly created directory and run `make`:

```
> cd Leo-III-1.3/
> make
```

   The building process might take some time, depending on your computer system. You can also build a static version of Leo-III (if you want to move the executable around to other machines) using `make static`. Alternatively, if you do not have gcc installed (and you do not require external cooperation), you can run `make leo3`. However, you will not be able to make use of external reasoning systems to increase Leo-III's reasoning effectivity.
3. If no error occurred, you should find a `bin` directory at top-level:

```
> cd bin/
> ls
  leo3 leo3.jar
```

   where `leo3` is the executable of Leo-III. A jar file `leo3.jar` is also generated in case you want to include Leo-III as a library to some other application.
4. Optionally install the Leo-III binaries to a dedicated location using

```
> make install
```

   This will copy the `leo3` executable to the default install destination `$HOME/bin`. The install destination can be modified using the `DESTDIR` modifier.

A.3. Usage

It is assumed that the `leo3` executable can be found in `$PATH` in the following. Leo-III is invoked via command-line:

```
> leo3
Leo III -- A Higher-Order Theorem Prover.
C. Benzmuller, A. Steen, M. Wisniewski and others.

Usage: leo3 problem [option ...]
[...]
```

The release of Leo-III contains several test problems, including a polymorphic THF formulation of the surjective Cantor theorem (cf. example *E9* from §5.2), located at `./src/test/resources/th1/sur_cantor_th1.p`. The problem statement reads as follows:

```
thf(sur_cantor, conjecture, ![T:$tType]: (~(?[F: T>(T>$o)]: (
                                ! [Y:T>$o]: (
                                 ? [X:T]: (
                                   (F @ X) = Y
                                  )
                                 ) )))).
```

Leo-III can now be invoked for proving this conjecture. The `-p` option enables the output of a proof certificate (formatted using Sutcliffe's TPTP4X tool):

```
> ./leo3 ./src/test/resources/th1/sur_cantor_th1.p -p
% Time passed: 2490ms
% Effective reasoning time: 1619ms
% Axioms used in derivation (0):
% No. of inferences in proof: 14
% No. of processed clauses: 9
% No. of generated clauses: 60
[...]
% SZS status Theorem for ./src/test/resources/th1/sur_cantor_th1.p : 2490 ms resp. 1619
    ms w/o parsing
% SZS output start CNFRefutation for ./src/test/resources/th1/sur_cantor_th1.p
thf(skt1_type,type,( skt1: $tType )).

thf(sk1_type,type,( sk1: skt1 > skt1 > $o )).

thf(sk2_type,type,( sk2: ( skt1 > $o ) > skt1 )).

thf(1,conjecture,(
    ! [TA: $tType] :
      ~ ? [A: TA > TA > $o] :
        ! [B: TA > $o] :
```

```
        ? [C: TA] :
          ( ( A @ C )
          = B ) ),
    file('./src/test/resources/th1/sur_cantor_th1.p',sur_cantor)).

thf(2,negated_conjecture,(
    ~ ! [TA: $tType] :
        ~ ? [A: TA > TA > $o] :
          ! [B: TA > $o] :
          ? [C: TA] :
            ( ( A @ C )
            = B ) ),
    inference(neg_conjecture,[status(cth)],[1])).

thf(3,plain,(
    ~ ! [TA: $tType] :
        ~ ? [A: TA > TA > $o] :
          ! [B: TA > $o] :
          ? [C: TA] :
            ( ( A @ C )
            = B ) ),
    inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).

thf(4,plain,(
    ! [A: skt1 > $o] :
      ( ( sk1 @ ( sk2 @ A ) )
      = A ) ),
    inference(cnf,[status(esa)],[3])).

thf(5,plain,(
    ! [A: skt1 > $o] :
      ( ( sk1 @ ( sk2 @ A ) )
      = A ) ),
    inference(lifteq,[status(thm)],[4])).

thf(6,plain,(
    ! [B: skt1,A: skt1 > $o] :
      ( ( sk1 @ ( sk2 @ A ) @ B )
      = ( A @ B ) ) ),
    inference(func_ext,[status(esa)],[5])).

thf(8,plain,(
    ! [B: skt1,A: skt1 > $o] :
      ( ( sk1 @ ( sk2 @ A ) @ B )
      | ~ ( A @ B ) ) ),
    inference(bool_ext,[status(thm)],[6])).

thf(198,plain,(
    ! [B: skt1,A: skt1 > $o] :
      ( ( sk1 @ ( sk2 @ A ) @ B )
      | ( ( A @ B )
       != ( ~ ( sk1 @ ( sk2 @ A ) @ B ) ) )
      | ~ $true ) ),
    inference(eqfactor_ordered,[status(thm)],[8])).

thf(217,plain,
    ( sk1
    @ ( sk2
      @ ^ [A: skt1] :
          ~ ( sk1 @ A @ A ) )
    @ ( sk2
      @ ^ [A: skt1] :
          ~ ( sk1 @ A @ A ) ) ),
    inference(pre_uni,[status(thm)],[198:[bind(A,$thf(^ [C: skt1] : ~ ( sk1 @ C @ C ))),
         bind(B,$thf(sk2 @ ^ [C: skt1] : ~ ( sk1 @ C @ C )))]])).

thf(7,plain,(
    ! [B: skt1,A: skt1 > $o] :
      ( ~ ( sk1 @ ( sk2 @ A ) @ B )
      | ( A @ B ) ) ),
    inference(bool_ext,[status(thm)],[6])).

thf(17,plain,(
    ! [B: skt1,A: skt1 > $o] :
      ( ~ ( sk1 @ ( sk2 @ A ) @ B )
      | ( ( A @ B )
       != ( ~ ( sk1 @ ( sk2 @ A ) @ B ) ) )
      | ~ $true ) ),
    inference(eqfactor_ordered,[status(thm)],[7])).

thf(29,plain,(
    ~ ( sk1
      @ ( sk2
        @ ^ [A: skt1] :
            ~ ( sk1 @ A @ A ) )
      @ ( sk2
        @ ^ [A: skt1] :
            ~ ( sk1 @ A @ A ) ) ) ),
    inference(pre_uni,[status(thm)],[17:[bind(A,$thf(^ [C: skt1] : ~ ( sk1 @ C @ C ))),
         bind(B,$thf(sk2 @ ^ [C: skt1] : ~ ( sk1 @ C @ C )))]])).

thf(225,plain,($false),
    inference(rewrite,[status(thm)],[217,29])).
% SZS output end CNFRefutation for ./src/test/resources/th1/sur_cantor_th1.p
```

The line starting with ”`% SZS status Theorem`” confirms that the conjecture is indeed a theorem and the contents between ”`% SZS output start`” and ”`% SZS output end`” are the proof certificate for this claim.

B. List of Contributions

This thesis project contributes both to theoretical and practical aspects of higher-order logic automation. An overview over the individual contributions is displayed below:

Theoretical contributions

Model Existence. Following earlier work by Benzmüller, Kohlhase and Brown, a model existence theorem is presented for an extensional formulation of higher-order logic that is based on primitive equality as sole logical connective. The choice of primitive equality as only connective allows dropping the requirement $\mathfrak{q}$ of Benzmüller et al. (who had to explicitly assume it) as it is implied by the definition of HOL models. For the model existence result, a set of abstract consistency properties is presented that simplifies and combines earlier work of Benzmüller and Brown in the setting of equational higher-order logic with Henkin semantics. As a consequence, the model existence theorem moreover does not assume saturation which makes it applicable for completeness proofs of machine-oriented calculi.

Higher-Order Paramodulation. A paramodulation calculus that is sound and complete with respect to Henkin semantics (without choice) and that treats equality as a primitive notion rather than being defined, is presented. In contrast to earlier work of Benzmüller, the calculus EP does not come with a dedicated resolution rule and does not expand occurrences of equality predicates to their corresponding definition due to Leibniz.

Practical contributions

System Implementation. One of the main practical contributions of this thesis project is the design, development, evaluation and dissemination of the novel, stand-alone HO ATP system Leo-III. Leo-III has been implemented from scratch (it is not an extension/variant of LEO-II implementation code) in the modern Scala language, accumulating roughly 40000 lines of code in total, thereof approx. 30000 lines of code directly related to this dissertation project. Leo-III is one of the most powerful HO ATP systems today and is, up to the author's knowledge, the most widely applicable ATP available in terms of supported logics.

Heuristic Equational Simplifications. Using a higher-order term ordering, Leo-III pioneers the use of equational simplification procedures from first-order theorem proving, in particular from superposition systems. Additionally, more sophisticated clausification routines have been developed that make use of formula renaming and subterm extraction.

Function synthesis. Leo-III seems to be the first HO ATP system that comes with dedicated rules for function synthesis, including a special treatment of injective functions. The underlying calculus rules have been introduced in this thesis.

Proof guidance. Sophisticated clause selection schemes for saturation-based ATPs have been presented in this thesis. These selection mechanism have been adapted from approaches that have significantly improved the performance of ATP systems in the context of first-order reasoning.

Flexible Cooperation Schemes. Flexible means for cooperating with different kinds of external reasoning systems have been developed and presented in this thesis project. The implementation is asynchronous (non-blocking), independent of the concrete external prover at hand and adjusts its translation procedure to the capabilities of the goal system.

Polymorphic HOL Reasoning. A conservative extension of the EP calculus is contributed that generalizes paramodulation to rank-1 polymorphic HOL. This generalization has been implemented within Leo-III, yielding a reasoning system capable of reasoning in TF1 and TH1 languages.

Reasoning in Non-Classical Logics. An automation procedure for reasoning in numerous higher-order modal logics is presented in this thesis. In particular, a new problem syntax is proposed that will be further developed into a new TPTP language standard.

Data structures. The work presents a sophisticated data structure for higher-order terms that combines a locally nameless spine term representation with explicit substitutions for efficient substitution and traversal operations.

Indexing methods. Leo-III makes use of a higher-order adaption of feature vector indexing that reduces the number of subsumption checks during proof search. Up to the author's knowledge, Leo-III is the first HO ATP system that uses non-trivial term indexing methods for relevant high-level proof search procedures.

Inconsistent Problems. Using the novel feature of Leo-III to detect inconsistencies in successfully proven problems, this thesis project has contributed 15 inconsistent problems from the TPTP library that should hence be amended.

Integration into Isabelle/HOL. As an additional contribution of this thesis project, the Leo-III prover has been included into the Sledgehammer system of Isabelle/HOL. Consequently, Leo-III can now be used for discharging higher-order proof obligations from within the Isabelle/HOL proof assistant (as of release Isabelle 2018, August 2018).

C. Complete Leo-III Proofs

The complete Leo-III proofs of the example problems from §5 are displayed in this section.[1] All of the proofs given below are produced by Leo-III as-is (except for editorial changes).

C.1. Proof of `SET557^1`

```
% SZS status Theorem for SET557^1.p
% SZS output start CNFRefutation for SET557^1.p
thf(sk1_type, type, sk1: ($i > ($i > $o))).
thf(sk2_type, type, sk2: (($i > $o) > $i)).
thf(1,conjecture,((~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = B))
      )),file('/home/lex/TPTP/Problems/SET/SET557^1.p',surjectiveCantorThm)).
thf(2,negated_conjecture,((~ (~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A
       @ C) = B))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (~ (? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = (B)))
      ))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,((? [A:($i > ($i > $o))]: ! [B:($i > $o)]: ? [C:$i]: ((A @ C) = (B)))),
      inference(polarity_switch,[status(thm)],[3])).
thf(5,plain,(! [A:($i > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(cnf,[status(esa)
      ],[4])).
thf(6,plain,(! [A:($i > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(lifteq,[status(
      thm)],[5])).
thf(7,plain,(! [B:$i,A:($i > $o)] : (((sk1 @ (sk2 @ (A)) @ B) = (A @ B)))),inference(
      func_ext,[status(esa)],[6])).
thf(9,plain,(! [B:$i,A:($i > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | (~ (A @ B)))),inference(
      bool_ext,[status(thm)],[7])).
thf(199,plain,(! [B:$i,A:($i > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | ((A @ B) != (~ (sk1 @
      (sk2 @ (A)) @ B))) | ~ ($true))),inference(eqfactor_ordered,[status(thm)],[9])).
thf(216,plain,((sk1 @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:$i]: ~ (sk1 @
      A @ A))))),inference(pre_uni,[status(thm)],[199:[bind(A, $thf(^ [C:$i]: ~ (sk1 @ C
       @ C))),bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(8,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | (A @ B))),inference(
      bool_ext,[status(thm)],[7])).
thf(18,plain,(! [B:$i,A:($i > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | ((A @ B) != (~ (sk1
       @ (sk2 @ (A)) @ B))) | ~ ($true))),inference(eqfactor_ordered,[status(thm)],[8]))
      .
thf(28,plain,((~ (sk1 @ (sk2 @ (^ [A:$i]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:$i]: ~ (sk1
      @ A @ A)))))),inference(pre_uni,[status(thm)],[18:[bind(A, $thf(^ [C:$i]: ~ (sk1 @
       C @ C))),bind(B, $thf(sk2 @ (^ [C:$i]: ~ (sk1 @ C @ C))))]])).
thf(230,plain,($false),inference(rewrite,[status(thm)],[216,28])).
% SZS output end CNFRefutation for SET557^1.p
```

[1] The proof of `MSC007^1.003.004` from example *E3* is not included as it consumes more than 10 pages.

C.2. Proof of SYO037^1

```
% SZS status Theorem for SYO037^1.p : 9613 ms resp. 7245 ms w/o parsing
% SZS output start CNFRefutation for SYO037^1.p
thf(sk1_type, type, sk1: (($i > $o) > $i)). thf(sk2_type, type, sk2: ($i > ($i > $o))).
thf(7,axiom,(! [A:($i > $o)] : (((sk2 @ (sk1 @ A)) = A))),introduced(tautology,[
      new_symbols(inverse(sk1),[sk2])])).
thf(8,plain,(! [B:$i,A:($i > $o)] : (((sk2 @ (sk1 @ A) @ B) = (A @ B)))),inference(
      func_ext,[status(esa)],[7])).
thf(1,conjecture,((~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o),C:($i > $o)]: (((A @ B) =
      (A @ C)) => (B = C))))),file('/home/lex/TPTP/Problems/SYO/SYO037^1.p',conj)).
thf(2,negated_conjecture,((~ (~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o),C:($i > $o)]:
      (((A @ B) = (A @ C)) => (B = C)))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,(~ (~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o),C:($i > $o)]: (((A @ B) = (A
      @ C)) => (B = C))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,((? [A:(($i > $o) > $i)]: ! [B:($i > $o),C:($i > $o)]: (((A @ (B)) = (A @ (
      C))) => ((B) = (C))))),inference(polarity_switch,[status(thm)],[3])).
thf(5,plain,(! [B:($i > $o),A:($i > $o)] : ((~ ((sk1 @ (A)) = (sk1 @ (B)))) | ((A) = (B
      )))),inference(cnf,[status(esa)],[4])).
thf(6,plain,(! [B:($i > $o),A:($i > $o)] : (((sk1 @ (A)) != (sk1 @ (B))) | ((A) = (B)))
      ),inference(lifteq,[status(thm)],[5])).
thf(13,plain,(! [C:$i,B:($i > $o),A:($i > $o)] : (((A @ C) = (B @ C)) | ((sk1 @ (A)) !=
       (sk1 @ (B))))),inference(func_ext,[status(esa)],[6])).
thf(263,plain,(! [E:$i,D:($i > $o),C:($i > $o),B:$i,A:($i > $o)] : (((A @ B) = (C @ E))
       | ((sk1 @ (C)) != (sk1 @ (D))) | ((sk2 @ (sk1 @ (A)) @ B) != (D @ E)))),inference
      (paramod_ordered,[status(thm)],[8,13])).
thf(332,plain,(! [D:($i > ($i > $o)),C:($i > $i),B:$i,A:($i > $o)] : (((D @ B @ (C @ B)
      ) = (A @ B)) | ((sk1 @ (A)) != (sk1 @ (^ [E:$i]: (sk2 @ (sk1 @ (D @ E)) @ (C @ E))
      ))))),inference(pre_uni,[status(thm)],[263:[bind(A, $thf(H @ E)),bind(B, $thf(G @
      E)),bind(C, $thf(C)),bind(D, $thf(^ [H:$i]: (sk2 @ (sk1 @ (H @ H)) @ (G @ H)))),
      bind(E, $thf(E))]])).
thf(333,plain,(! [C:($i > ($i > $o)),B:($i > $i),A:$i] : (((C @ A @ (B @ A)) = (sk2 @ (
      sk1 @ (C @ A)) @ (B @ A))))),inference(pattern_uni,[status(thm)],[332:[bind(A,
      $thf(^ [G:$i]: (sk2 @ (sk1 @ (G @ G)) @ (F @ G)))),bind(B, $thf(A)),bind(C, $thf(F
      )),bind(D, $thf(G))]])).
thf(356,plain,(! [C:($i > ($i > $o)),B:($i > $i),A:$i] : (((C @ A @ (B @ A)) = (sk2 @ (
      sk1 @ (C @ A)) @ (B @ A))))),inference(simp,[status(thm)],[333])).
thf(9,plain,(! [B:$i,A:($i > $o)] : ((~ (sk2 @ (sk1 @ A) @ B)) | (A @ B))),inference(
      bool_ext,[status(thm)],[8])).
thf(14,plain,(! [B:($i > $o),A:$i] : ((~ (sk2 @ (sk1 @ (^ [C:$i]: ~ (B @ C))) @ A)) |
      (~ (B @ A)))),inference(prim_subst,[status(thm)],[9:[bind(A, $thf(^ [D:$i]: ~ (C @
       D)))]])).
thf(18,plain,(! [B:($i > $o),A:$i] : ((~ (B @ A)) | (~ (sk2 @ (sk1 @ (^ [C:$i]: ~ (B @
      C))) @ A)))),inference(cnf,[status(esa)],[14])).
thf(20,plain,(! [B:($i > $o),A:$i] : ((~ (B @ A)) | (~ (sk2 @ (sk1 @ (^ [C:$i]: ~ (B @
      C))) @ A)))),inference(simp,[status(thm)],[18])).
thf(963,plain,(! [B:($i > $o),A:$i]: ((~ (B @ A)) | ((sk2 @ (sk1 @ (^ [C:$i]: ~ (B @ C)
      )) @ A) != (B @ A)) | ~ ($true))),inference(eqfactor_ordered,[status(thm)],[20])).
thf(987,plain,((~ (sk2 @ (sk1 @ (^ [A:$i]: ~ (sk2 @ A @ A))) @ (sk1 @ (^ [A:$i]: ~ (sk2
       @ A @ A)))))),inference(pre_uni,[status(thm)],[963:[bind(A, $thf(sk1 @ (^ [C:$i]:
       ~ (sk2 @ C @ C)))),bind(B, $thf(^ [C:$i]: (sk2 @ C @ C)))]])).
thf(1030,plain,(! [C:($i > ($i > $o)),B:($i > $i),A:$i] : ((~ (C @ A @ (B @ A))) | ((
      sk2 @ (sk1 @ (C @ A)) @ (B @ A)) != (sk2 @ (sk1 @ (^ [D:$i]: ~ (sk2 @ D @ D))) @ (
      sk1 @ (^ [D:$i]: ~ (sk2 @ D @ D))))))),inference(paramod_ordered,[status(thm)
      ],[356,987])).
```

```
thf(1042,plain,((~ (~ (sk2 @ (sk1 @ (^ [A:$i]: ~ (sk2 @ A @ A))) @ (sk1 @ (^ [A:$i]: ~
      (sk2 @ A @ A))))))),inference(pre_uni,[status(thm)],[1030:[bind(A, $thf(sk1 @ (^ [
      D:$i]: ~ (sk2 @ D @ D)))),bind(B, $thf(^ [D:$i]: (D))),bind(C, $thf(^ [D:$i,E:$i]:
       ~ (sk2 @ E @ E)))]])).
thf(1044,plain,((sk2 @ (sk1 @ (^ [A:$i]: ~ (sk2 @ A @ A))) @ (sk1 @ (^ [A:$i]: ~ (sk2 @
       A @ A))))),inference(cnf,[status(esa)],[1042])).
thf(1051,plain,($false),inference(rewrite,[status(thm)],[1044,987])).
% SZS output end CNFRefutation for  SYO037^1.p
```

C.3. Proof of `NUN025^1`

```
% SZS status Theorem for NUN025^1.p : 6417 ms resp. 5077 ms w/o parsing
% SZS output start CNFRefutation for NUN025^1.p
thf(zero_type, type, zero: $i). thf(s_type, type, s: ($i > $i)).
thf(ite_type, type, ite: ($o > ($i > ($i > $i)))).
thf(1,conjecture,(((! [A:$o,B:$i,C:$i]: ((A) => ((ite @ A @ B @ C) = B)) & ! [A:$o,B:$i
     ,C:$i]: ((~ (A)) => ((ite @ A @ B @ C) = C)) & ! [A:$i]: ((s @ A) != A)) => (? [A
     :($i > $i)]: (((A @ zero) = (s @ zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (
     s @ zero))) = (s @ zero)))))),file('/home/lex/TPTP/Problems/NUN/NUN025^1.p',n9)).
thf(2,negated_conjecture,~ ((! [A:$o,B:$i,C:$i]: ((A) => ((ite @ A @ B @ C) = B)) & ! [
     A:$o,B:$i,C:$i]: ((~ (A)) => ((ite @ A @ B @ C) = C)) & ! [A:$i]: ((s @ A) != A))
     => (? [A:($i > $i)]: (((A @ zero) = (s @ zero)) & ((A @ (s @ zero)) = zero) & ((A
     @ (s @ (s @ zero))) = (s @ zero))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,~((! [A:$o,B:$i,C:$i]: (A => ((ite @ A @ B @ C) = B)) & ! [A:$o,B:$i,C:$i]:
      ((~A) => ((ite @ A @ B @ C) = C)) & ! [A:$i]: ~ ((s @ A) = A)) => (? [A:($i > $i)
     ]: (((A @ zero) = (s @ zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (s @ zero))
     ) = (s @ zero))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,~ ((! [A:$o]: ((A) => (! [B:$i,C:$i]: ((ite @ A @ B @ C) = B))) & ! [A:$o]:
      ((~ (A)) => (! [B:$i,C:$i]: ((ite @ A @ B @ C) = C))) & ~ (? [A:$i]: ((s @ A) = A
     ))) => (? [A:($i > $i)]: (((A @ zero) = (s @ zero)) & ((A @ (s @ zero)) = zero) &
     ((A @ (s @ (s @ zero))) = (s @ zero))))),inference(miniscope,[status(thm)],[3])).
thf(7,plain,(! [C:$i,B:$i,A:$o] : (A | ((ite @ A @ B @ C) = C))),inference(cnf,[status(
     esa)],[4])).
thf(13,plain,(! [C:$i,B:$i,A:$o] : (((ite @ A @ B @ C) = C) | A)),inference(lifteq,[
     status(thm)],[7])).
thf(14,plain,(! [C:$i,B:$i,A:$o] : (((ite @ A @ B @ C) = C) | A)),inference(simp,[
     status(thm)],[13])).
thf(16,plain,(! [C:$o,B:$i,A:$i] : (((ite @ (~ (C)) @ A @ B) = B) | (~ (C)))),inference
     (prim_subst,[status(thm)],[14:[bind(A, $thf(~ (D)))]])).
thf(20,plain,(! [C:$o,B:$i,A:$i] : (~ (C) | ((ite @ (~ (C)) @ A @ B) = B))),inference(
     cnf,[status(esa)],[16])).
thf(22,plain,(! [C:$o,B:$i,A:$i] : (~ (C) | ((ite @ (~ (C)) @ A @ B) = B))),inference(
     simp,[status(thm)],[20])).
thf(8,plain,(! [C:$i,B:$i,A:$o] : (~ (A) | ((ite @ A @ B @ C) = B))),inference(cnf,[
     status(esa)],[4])).
thf(15,plain,(! [C:$i,B:$i,A:$o] : (((ite @ A @ B @ C) = B) | ~ (A))),inference(lifteq
     ,[status(thm)],[8])).
thf(5,plain,(! [A:($i > $i)] : ((~ ((A @ zero) = (s @ zero))) | (~ ((A @ (s @ zero)) =
     zero)) | (~ ((A @ (s @ (s @ zero))) = (s @ zero))))),inference(cnf,[status(esa)
     ],[4])).
thf(9,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero)
      | ((A @ (s @ (s @ zero))) != (s @ zero)))),inference(lifteq,[status(thm)],[5])).
thf(10,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero
     ) | ((A @ (s @ (s @ zero))) != (s @ zero)))),inference(simp,[status(thm)],[9])).
thf(56,plain,(! [D:($i > $i),C:$i,B:$i,A:$o] : (~ (A) | ((D @ zero) != (s @ zero)) | ((
     D @ (s @ zero)) != zero) | (B != (s @ zero)) | ((ite @ A @ B @ C) != (D @ (s @ (s
     @ zero)))))),inference(paramod_ordered,[status(thm)],[15,10])).
thf(60,plain,(! [C:($i > $i),B:($i > $i),A:($i > $o)] : ((~ (A @ (s @ (s @ zero)))) |
     ((ite @ (A @ zero) @ (B @ zero) @ (C @ zero)) != (s @ zero)) | ((ite @ (A @ (s @
     zero)) @ (B @ (s @ zero)) @ (C @ (s @ zero))) != zero) | ((B @ (s @ (s @ zero)))
     != (s @ zero)))),inference(pre_uni,[status(thm)],[56:[bind(A, $thf(E @ (s @ (s @
     zero)))),bind(B, $thf(F @ (s @ (s @ zero)))),bind(C, $thf(G @ (s @ (s @ zero)))),
     bind(D, $thf(^ [H:$i]: (ite @ (E @ H) @ (F @ H) @ (G @ H))))]])).
thf(74,plain,(! [C:($i > $i),B:($i > $i),A:($i > $o)] : ((~ (A @ (s @ (s @ zero)))) |
     ((ite @ (A @ zero) @ (B @ zero) @ (C @ zero)) != (s @ zero)) | ((ite @ (A @ (s @
     zero)) @ (B @ (s @ zero)) @ (C @ (s @ zero))) != zero) | ((B @ (s @ (s @ zero)))
     != (s @ zero)))),inference(simp,[status(thm)],[60])).
thf(455,plain,(! [F:($i > $i),E:($i > $i),D:($i > $o),C:$i,B:$i,A:$o] : (~ (A) | (~ (D
     @ (s @ (s @ zero)))) | (B != (s @ zero)) | ((ite @ (D @ (s @ zero)) @ (E @ (s @
     zero)) @ (F @ (s @ zero))) != zero) | ((E @ (s @ (s @ zero))) != (s @ zero)) | ((
     ite @ A @ B @ C) != (ite @ (D @ zero) @ (E @ zero) @ (F @ zero))))),inference(
     paramod_ordered,[status(thm)],[15,74])).
thf(507,plain,(! [C:($i > $i),B:($i > $i),A:($i > $o)] : ((~ (A @ zero)) | (~ (A @ (s @
      (s @ zero)))) | ((B @ zero) != (s @ zero)) | ((ite @ (A @ (s @ zero)) @ (B @ (s @
      zero)) @ (C @ (s @ zero))) != zero) | ((B @ (s @ (s @ zero))) != (s @ zero)))),
     inference(pre_uni,[status(thm)],[455:[bind(A, $thf(D @ zero)),bind(B, $thf(E @
     zero)),bind(C, $thf(F @ zero))]])).
thf(557,plain,(! [C:($i > $i),B:($i > $i),A:($i > $o)] : ((~ (A @ zero)) | (~ (A @ (s @
      (s @ zero)))) | ((B @ zero) != (s @ zero)) | ((ite @ (A @ (s @ zero)) @ (B @ (s @
      zero)) @ (C @ (s @ zero))) != zero) | ((B @ (s @ (s @ zero))) != (s @ zero)))),
     inference(simp,[status(thm)],[507])).
thf(642,plain,(! [F:($i > $i),E:($i > $i),D:($i > $o),C:$o,B:$i,A:$i] : (~ (C) | (~ (D
     @ zero)) | (~ (D @ (s @ (s @ zero)))) | ((E @ zero) != (s @ zero)) | (B != zero) |
      ((E @ (s @ (s @ zero))) != (s @ zero)) | ((ite @ (~ (C)) @ A @ B) != (ite @ (D @
     (s @ zero)) @ (E @ (s @ zero)) @ (F @ (s @ zero)))))),inference(paramod_ordered,[
     status(thm)],[22,557])).
thf(687,plain,(! [C:($i > $o),B:($i > $i),A:($i > $i)] : ((~ (C @ (s @ zero))) | (~ (~
     (C @ zero))) | (~ (~ (C @ (s @ (s @ zero))))) | ((A @ zero) != (s @ zero)) | ((B @
      (s @ zero)) != zero) | ((A @ (s @ (s @ zero))) != (s @ zero)))),inference(pre_uni
     ,[status(thm)],[642:[bind(A, $thf(E @ (s @ zero))),bind(B, $thf(F @ (s @ zero))),
     bind(C, $thf(G @ (s @ zero))),bind(D, $thf(^ [H:$i]: ~ (G @ H)))]])).
thf(688,plain,(! [A:($i > $o)] : ((~ (~ (A @ (s @ (s @ zero))))) | (~ (~ (A @ zero))) |
      (~ (A @ (s @ zero))))),inference(pre_uni,[status(thm)],[687:[bind(A, $thf(^ [D:$i
     ]: (s @ zero))),bind(B, $thf(^ [D:$i]: (zero))),bind(C, $thf(C))]])).
thf(731,plain,(! [A:($i > $o)] : ((~ (A @ (s @ zero))) | (A @ zero) | (A @ (s @ (s @
     zero))))),inference(cnf,[status(esa)],[688])).
thf(782,plain,(! [A:($i > $o)] : ((~ (A @ (s @ zero))) | (A @ zero) | (A @ (s @ (s @
     zero))))),inference(simp,[status(thm)],[731])).
thf(1422,plain,(! [A:($i > $o)] : ((~ (A @ (s @ zero))) | (A @ zero) | ((A @ (s @ (s @
     zero))) != (~ (A @ (s @ zero)))) | ~ ($true))),inference(eqfactor_ordered,[status(
     thm)],[782])).
thf(1432,plain,((~ ((s @ zero) = (s @ zero))) | ((s @ zero) = zero) | (((s @ zero) = (s
      @ (s @ zero))) != (~ ((s @ zero) = (s @ zero)))) | ~ ($true)),inference(
     replace_leibeq,[status(thm)],[1422:[bind(A, $thf(= @ $i @ (s @ zero)))]])).
thf(1476,plain,(((s @ zero) != (s @ zero)) | ((s @ zero) = zero) | (((s @ zero) = (s @
     (s @ zero))) != (~ ((s @ zero) = (s @ zero)))) | ~ ($true)),inference(lifteq,[
     status(thm)],[1432])).
thf(1618,plain,(((s @ zero) = zero) | ((s @ zero) = (s @ (s @ zero)))),inference(simp,[
     status(thm)],[1476])).
thf(1634,plain,(((s @ (s @ zero)) = (s @ zero)) | ((s @ zero) = zero)),inference(lifteq
     ,[status(thm)],[1618])).
thf(6,plain,(! [A:$i] : ((~ ((s @ A) = A)))),inference(cnf,[status(esa)],[4])).
thf(11,plain,(! [A:$i] : (((s @ A) != A))),inference(lifteq,[status(thm)],[6])).
thf(12,plain,(! [A:$i] : (((s @ A) != A))),inference(simp,[status(thm)],[11])).
thf(58,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero
     ) | ((A @ (s @ (s @ zero))) != (s @ zero)) | ((s @ zero) != (A @ zero)))),
     inference(eqfactor_ordered,[status(thm)],[10])).
thf(69,plain,(((s @ zero) != (s @ zero)) | ((s @ zero) != zero)),inference(pre_uni,[
     status(thm)],[58:[bind(A, $thf(^ [B:$i]: (s @ zero)))]])).
thf(71,plain,(((s @ zero) != zero)),inference(simp,[status(thm)],[69])).
thf(1834,plain,($false),inference(simplifyReflect,[status(thm)],[1634,12,71])).
% SZS output end CNFRefutation for NUN025^1.p
```

C.4. Proof of `SYN997^1`

```
% SZS status Theorem for SYN997^1.p : 3332 ms resp. 1618 ms w/o parsing
% SZS output start CNFRefutation for SYN997^1.p
thf(sk1_type, type, sk1: ((($i > $o) > $i) > ($i > $o))).
thf(sk2_type, type, sk2: ((($i > $o) > $i) > $i)).
thf(1,conjecture,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B
      @ (A @ B))))),file('/home/lex/TPTP/Problems/SYN/SYN997^1.p',conj)).
thf(2,negated_conjecture,((~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @
       C)) => (B @ (A @ B)))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B @
      (A @ (B))))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(5,plain,(! [A:(($i > $o) > $i)] : ((sk1 @ (A) @ (sk2 @ (A))))),inference(cnf,[
      status(esa)],[3])).
thf(7,axiom,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B @ (A
      @ B))))),introduced(axiom_of_choice)).
thf(8,plain,(! [B:$i,A:(($i > $o) > $i)] : ((~ (sk1 @ (A) @ B)) | (sk1 @ (A) @ (@+ [C:
      $i]: (sk1 @ (A) @ C))))),inference(choice,[status(esa)],[7])).
thf(4,plain,(! [A:(($i > $o) > $i)] : ((~ (sk1 @ (A) @ (A @ (sk1 @ (A))))))),inference(
      cnf,[status(esa)],[3])).
thf(13,plain,(! [C:(($i > $o) > $i),B:$i,A:(($i > $o) > $i)] : ((~ (sk1 @ (A) @ B)) |
      ((sk1 @ (A) @ (@+ [D:$i]: (sk1 @ (A) @ D))) != (sk1 @ (C) @ (C @ (sk1 @ (C))))))),
      inference(paramod_ordered,[status(thm)],[8,4])).
thf(16,plain,(! [A:$i] : ((~ (sk1 @ (^ [P: $i > $o]: (@+ [X: $i]: (P @ X))) @ A)))),
      inference(pre_uni,[status(thm)],[13:[bind(A, $thf((^ [P: $i > $o]: (@+ [X: $i]: (P
       @ X))))),bind(B, $thf(B)),bind(C, $thf((^ [P: $i > $o]: (@+ [X: $i]: (P @ X)))))
      ]]])).
thf(18,plain,(! [A:$i] : ((~ (sk1 @ ((^ [P: $i > $o]: (@+ [X: $i]: (P @ X)))) @ A)))),
      inference(simp,[status(thm)],[16])).
thf(19,plain,(! [B:$i,A:(($i > $o) > $i)] : (((sk1 @ (A) @ (sk2 @ (A))) != (sk1 @ ((^ [
      P: $i > $o]: (@+ [X: $i]: (P @ X)))) @ B)))),inference(paramod_ordered,[status(thm
      )],[5,18])).
thf(20,plain,($false),inference(pattern_uni,[status(thm)],[19:[bind(A, $thf((^ [P: $i >
       $o]: (@+ [X: $i]: (P @ X))))),bind(B, $thf(sk2 @ ((^ [P: $i > $o]: (@+ [X: $i]: (
      P @ X))))))]])).
% SZS output end CNFRefutation for SYN997^1.p
```

C.5. Proof of `SYO548^1`

```
% SZS status Theorem for SYO548^1.p : 1352 ms resp. 696 ms w/o parsing
% SZS output start CNFRefutation for SYO548^1.p
thf(sk3_type, type, sk3: ($i > $o)).
thf(sk4_type, type, sk4: $i).
thf(1,conjecture,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: ~ (B @ C)) =>
      (~ (B @ (A @ B)))))),file('/home/lex/TPTP/Problems/SYO/SYO548^1.p',choicecomp)).
thf(2,negated_conjecture,((~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: ~ (B
       @ C)) => (~ (B @ (A @ B))))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: ~ (B @ C)) => (~
      (B @ (A @ (B)))))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(5,plain,((~ (! [A:($i > $o)]: ((? [B:$i]: ~ (A @ B)) => (~ (A @ (@+ [B:$i]: ~ (A @
      B)))))))),inference(instance,[status(thm)],[3])).
thf(9,plain,((~ (! [A:($i > $o)]: ((~ (! [B:$i]: (A @ B))) => (~ (A @ (@+ [B:$i]: ~ (A
      @ B)))))))),inference(miniscope,[status(thm)],[5])).
thf(11,plain,((~ (sk3 @ sk4))),inference(cnf,[status(esa)],[9])).
thf(16,axiom,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B @ (A
       @ B))))),introduced(axiom_of_choice)).
thf(21,plain,(! [A:$i] : ((~ (~ (sk3 @ A))) | (~ (sk3 @ (@+ [B:$i]: ~ (sk3 @ B)))))),
      inference(choice,[status(esa)],[16])).
thf(23,plain,(! [A:$i] : ((~ (sk3 @ (@+ [B:$i]: ~ (sk3 @ B)))) | (sk3 @ A))),inference(
      cnf,[status(esa)],[21])).
thf(10,plain,((sk3 @ (@+ [A:$i]: ~ (sk3 @ A)))),inference(cnf,[status(esa)],[9])).
thf(24,plain,(! [A:$i] : (~ ($true) | (sk3 @ A))),inference(rewrite,[status(thm)
      ],[23,10])).
thf(25,plain,(! [A:$i] : ((sk3 @ A))),inference(simp,[status(thm)],[24])).
thf(27,plain,(~ ($true)),inference(rewrite,[status(thm)],[11,25])).
thf(28,plain,($false),inference(simp,[status(thm)],[27])).
% SZS output end CNFRefutation for SYO548^1.p
```

C.6. Proof of SYO519^1

```
% SZS status Theorem for SYO519^1.p : 2780 ms resp. 1407 ms w/o parsing
% SZS output start CNFRefutation for SYO519^1.p
thf(sk1_type, type, sk1: $i).
thf(sk2_type, type, sk2: $i).
thf(11,axiom,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B @ (A
       @ B))))),introduced(axiom_of_choice)).
thf(13,plain,(! [A:$i] : ((~ (((sk2 = sk1) => (A = sk2)) & (A = sk1))) | (((sk2 = sk1)
      => ((@+ [B:$i]: (((sk2 = sk1) => (B = sk2)) & (B = sk1))) = sk2)) & ((@+ [B:$i]:
      (((sk2 = sk1) => (B = sk2)) & (B = sk1))) = sk1)))),inference(choice,[status(esa)
      ],[11])).
thf(19,plain,(! [A:$i] : (((@+ [B:$i]: (((sk2 = sk1) => (B = sk2)) & (B = sk1))) = sk1)
       | (sk2 = sk1) | (~ (A = sk1)))),inference(cnf,[status(esa)],[13])).
thf(34,plain,(! [A:$i] : (((@+ [B:$i]: (((sk2 = sk1) => (B = sk2)) & (B = sk1))) = sk1)
       | (sk2 = sk1) | (A != sk1))),inference(lifteq,[status(thm)],[19])).
thf(35,plain,(((@+ [A:$i]: (((sk2 = sk1) => (A = sk2)) & (A = sk1))) = sk1) | (sk2 =
      sk1)),inference(simp,[status(thm)],[34])).
thf(1,conjecture,((! [A:$i,B:$i]: ? [C:($i > $i)]: (((C @ A) = B) & ((C @ B) = A)))),
      file('/home/lex/TPTP/Problems/SYO/SYO519^1.p',ifi)).
thf(2,negated_conjecture,((~ (! [A:$i,B:$i]: ? [C:($i > $i)]: (((C @ A) = B) & ((C @ B)
       = A))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (! [A:$i,B:$i]: ? [C:($i > $i)]: (((C @ A) = B) & ((C @ B) = A))))),
      inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,(! [A:($i > $i)] : ((~ ((A @ sk1) = sk2)) | (~ ((A @ sk2) = sk1)))),
      inference(cnf,[status(esa)],[3])).
thf(5,plain,(! [A:($i > $i)] : (((A @ sk1) != sk2) | ((A @ sk2) != sk1))),inference(
      lifteq,[status(thm)],[4])).
thf(7,plain,(! [A:($i > $i)] : (((A @ sk1) != sk2) | ((A @ sk2) != sk2) | ((A @ sk1) !=
       sk1))),inference(eqfactor_ordered,[status(thm)],[5])).
thf(9,plain,((sk2 != sk1)),inference(pre_uni,[status(thm)],[7:[bind(A, $thf(^ [B:$i]: (
      B)))]])).
thf(8,plain,(((@+ [A:$i]: (((sk1 = sk1) => (A = sk2)) & ((sk1 = sk2) => (A = sk1)))) !=
       sk2) | ((@+ [A:$i]: (((sk2 = sk1) => (A = sk2)) & ((sk2 = sk2) => (A = sk1)))) !=
       sk1)),introduced(choice_instance)).
thf(10,plain,(((@+ [A:$i]: ((A = sk2) & ((sk1 = sk2) => (A = sk1)))) != sk2) | ((@+ [A:
      $i]: (((sk2 = sk1) => (A = sk2)) & (A = sk1))) != sk1)),inference(simp,[status(thm
      )],[8])).
thf(12,plain,(! [A:$i] : ((~ ((A = sk2) & ((sk1 = sk2) => (A = sk1)))) | (((@+ [B:$i]:
      ((B = sk2) & ((sk1 = sk2) => (B = sk1)))) = sk2) & ((sk1 = sk2) => ((@+ [B:$i]: ((
      B = sk2) & ((sk1 = sk2) => (B = sk1)))) = sk1))))),inference(choice,[status(esa)
      ],[11])).
thf(17,plain,(! [A:$i] : (((@+ [B:$i]: ((B = sk2) & ((sk1 = sk2) => (B = sk1)))) = sk2)
       | (~ (A = sk2)) | (sk1 = sk2))),inference(cnf,[status(esa)],[12])).
thf(28,plain,(! [A:$i] : (((@+ [B:$i]: ((B = sk2) & ((sk1 = sk2) => (B = sk1)))) = sk2)
       | (A != sk2) | (sk2 = sk1))),inference(lifteq,[status(thm)],[17])).
thf(29,plain,(((@+ [A:$i]: ((A = sk2) & ((sk1 = sk2) => (A = sk1)))) = sk2) | (sk2 =
      sk1)),inference(simp,[status(thm)],[28])).
thf(38,plain,(((@+ [A:$i]: ((A = sk2) & ((sk1 = sk2) => (A = sk1)))) = sk2)),inference(
      simplifyReflect,[status(thm)],[29,9])).
thf(39,plain,((sk2 != sk2) | ((@+ [A:$i]: (((sk2 = sk1) => (A = sk2)) & (A = sk1))) !=
      sk1)),inference(rewrite,[status(thm)],[10,38])).
thf(40,plain,(((@+ [A:$i]: (((sk2 = sk1) => (A = sk2)) & (A = sk1))) != sk1)),inference
      (simp,[status(thm)],[39])).
thf(48,plain,($false),inference(simplifyReflect,[status(thm)],[35,9,40])).
% SZS output end CNFRefutation for SYO519^1.p
```

C.7. Proof of modified NUN025^1 (Ex. E_8)

```
% SZS status Theorem for - : 2992 ms resp. 2247 ms w/o parsing
% SZS output start CNFRefutation for -
thf(zero_type, type, zero: $i).
thf(s_type, type, s: ($i > $i)).
thf(33,axiom,((? [A:(($i > $o) > $i)]: ! [B:($i > $o)]: ((? [C:$i]: (B @ C)) => (B @ (A
       @ B))))),introduced(axiom_of_choice)).
thf(35,plain,(! [A:$i] : ((~ ((((s @ zero) = zero) => (A = (s @ zero))) & (A = zero) &
      (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) | ((((s @ zero) = zero) =>
       ((@+ [B:$i]: ((((s @ zero) = zero) => (B = (s @ zero))) & (B = zero) & (((s @
      zero) = (s @ (s @ zero))) => (B = (s @ zero))))) = (s @ zero))) & ((@+ [B:$i]:
      ((((s @ zero) = zero) => (B = (s @ zero))) & (B = zero) & (((s @ zero) = (s @ (s @
       zero))) => (B = (s @ zero))))) = zero) & (((s @ zero) = (s @ (s @ zero))) => ((@+
       [B:$i]: ((((s @ zero) = zero) => (B = (s @ zero))) & (B = zero) & (((s @ zero) =
      (s @ (s @ zero))) => (B = (s @ zero))))) = (s @ zero)))))),inference(choice,[
      status(esa)],[33])).
thf(62,plain,(! [A:$i] : (((@+ [B:$i]: ((((s @ zero) = zero) => (B = (s @ zero))) & (B
      = zero) & (((s @ zero) = (s @ (s @ zero))) => (B = (s @ zero))))) = zero) | ((s @
      zero) = zero) | (~ (A = zero)) | ((s @ zero) = (s @ (s @ zero))))),inference(cnf,[
      status(esa)],[35])).
thf(149,plain,(! [A:$i] : (((@+ [B:$i]: ((((s @ zero) = zero) => (B = (s @ zero))) & (B
       = zero) & (((s @ zero) = (s @ (s @ zero))) => (B = (s @ zero))))) = zero) | ((s @
       zero) = zero) | (A != zero) | ((s @ (s @ zero)) = (s @ zero)))),inference(lifteq
      ,[status(thm)],[62])).
thf(150,plain,(((@+ [A:$i]: ((((s @ zero) = zero) => (A = (s @ zero))) & (A = zero) &
      (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) = zero) | ((s @ zero) =
      zero) | ((s @ (s @ zero)) = (s @ zero))),inference(simp,[status(thm)],[149])).
thf(13,plain,(((@+ [A:$i]: (((zero = zero) => (A = (s @ zero))) & ((zero = (s @ zero))
      => (A = zero)) & ((zero = (s @ (s @ zero))) => (A = (s @ zero))))) != (s @ zero))
      | ((@+ [A:$i]: ((((s @ zero) = zero) => (A = (s @ zero))) & (((s @ zero) = (s @
      zero)) => (A = zero)) & (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) !=
       zero) | ((@+ [A:$i]: ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s @ (s
       @ zero)) = (s @ zero)) => (A = zero)) & (((s @ (s @ zero)) = (s @ (s @ zero))) =>
       (A = (s @ zero))))) != (s @ zero))),introduced(choice_instance)).
thf(17,plain,(((@+ [A:$i]: ((A = (s @ zero)) & ((zero = (s @ zero)) => (A = zero)) & ((
      zero = (s @ (s @ zero))) => (A = (s @ zero))))) != (s @ zero)) | ((@+ [A:$i]: ((((
      s @ zero) = zero) => (A = (s @ zero))) & (A = zero) & (((s @ zero) = (s @ (s @
      zero))) => (A = (s @ zero))))) != zero) | ((@+ [A:$i]: ((((s @ (s @ zero)) = zero)
       => (A = (s @ zero))) & (((s @ (s @ zero)) = (s @ zero)) => (A = zero)) & (A = (s
      @ zero)))) != (s @ zero))),inference(simp,[status(thm)],[13])).
thf(34,plain,(! [A:$i] : ((~ ((A = (s @ zero)) & ((zero = (s @ zero)) => (A = zero)) &
      ((zero = (s @ (s @ zero))) => (A = (s @ zero))))) | (((@+ [B:$i]: ((B = (s @ zero)
      ) & ((zero = (s @ zero)) => (B = zero)) & ((zero = (s @ (s @ zero))) => (B = (s @
      zero))))) = (s @ zero)) & ((zero = (s @ zero)) => ((@+ [B:$i]: ((B = (s @ zero)) &
       ((zero = (s @ zero)) => (B = zero)) & ((zero = (s @ (s @ zero))) => (B = (s @
      zero))))) = zero)) & ((zero = (s @ (s @ zero))) => ((@+ [B:$i]: ((B = (s @ zero))
      & ((zero = (s @ zero)) => (B = zero)) & ((zero = (s @ (s @ zero))) => (B = (s @
      zero))))) = (s @ zero)))))),inference(choice,[status(esa)],[33])).
thf(43,plain,(! [A:$i] : (((@+ [B:$i]: ((B = (s @ zero)) & ((zero = (s @ zero)) => (B =
       zero)) & ((zero = (s @ (s @ zero))) => (B = (s @ zero))))) = (s @ zero)) | (~ (A
      = (s @ zero))) | (zero = (s @ zero)) | (~ (A = (s @ zero))))),inference(cnf,[
      status(esa)],[34])).
thf(133,plain,(! [A:$i] : (((@+ [B:$i]: ((B = (s @ zero)) & ((zero = (s @ zero)) => (B
      = zero)) & ((zero = (s @ (s @ zero))) => (B = (s @ zero))))) = (s @ zero)) | (A !=
       (s @ zero)) | ((s @ zero) = zero) | (A != (s @ zero)))),inference(lifteq,[status(
      thm)],[43])).
thf(134,plain,(((@+ [A:$i]: ((A = (s @ zero)) & ((zero = (s @ zero)) => (A = zero)) &
      ((zero = (s @ (s @ zero))) => (A = (s @ zero))))) = (s @ zero)) | ((s @ zero) =
      zero)),inference(simp,[status(thm)],[133])).
thf(1,conjecture,(((! [A:$i]: ((s @ A) != A)) => (? [A:($i > $i)]: (((A @ zero) = (s @
      zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (s @ zero))) = (s @ zero)))))),
      file('-',n9)).
thf(2,negated_conjecture,((~ ((! [A:$i]: ((s @ A) != A)) => (? [A:($i > $i)]: (((A @
      zero) = (s @ zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (s @ zero))) = (s @
      zero))))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ ((! [A:$i]: ~ ((s @ A) = A)) => (? [A:($i > $i)]: (((A @ zero) = (s @
      zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (s @ zero))) = (s @ zero))))))),
      inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,((~ ((~ (? [A:$i]: ((s @ A) = A))) => (? [A:($i > $i)]: (((A @ zero) = (s @
       zero)) & ((A @ (s @ zero)) = zero) & ((A @ (s @ (s @ zero))) = (s @ zero))))))),
      inference(miniscope,[status(thm)],[3])).
thf(5,plain,(! [A:($i > $i)] : ((~ ((A @ zero) = (s @ zero))) | (~ ((A @ (s @ zero)) =
      zero)) | (~ ((A @ (s @ (s @ zero))) = (s @ zero))))),inference(cnf,[status(esa)
      ],[4])).
thf(7,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero)
       | ((A @ (s @ (s @ zero))) != (s @ zero)))),inference(lifteq,[status(thm)],[5])).
thf(8,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero)
       | ((A @ (s @ (s @ zero))) != (s @ zero)))),inference(simp,[status(thm)],[7])).
thf(11,plain,(! [A:($i > $i)] : (((A @ zero) != (s @ zero)) | ((A @ (s @ zero)) != zero
      ) | ((A @ (s @ (s @ zero))) != (s @ zero)) | ((s @ zero) != (A @ zero)))),
      inference(eqfactor_ordered,[status(thm)],[8])).
thf(15,plain,(((s @ zero) != (s @ zero)) | ((s @ zero) != zero)),inference(pre_uni,[
      status(thm)],[11:[bind(A, $thf(^ [B:$i]: (s @ zero)))]])).
thf(16,plain,(((s @ zero) != zero)),inference(simp,[status(thm)],[15])).
thf(153,plain,(((@+ [A:$i]: ((A = (s @ zero)) & ((zero = (s @ zero)) => (A = zero)) &
      ((zero = (s @ (s @ zero))) => (A = (s @ zero))))) = (s @ zero))),inference(
      simplifyReflect,[status(thm)],[134,16])).
thf(154,plain,(((s @ zero) != (s @ zero)) | ((@+ [A:$i]: ((((s @ zero) = zero) => (A =
      (s @ zero))) & (A = zero) & (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))
      ) != zero) | ((@+ [A:$i]: ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s
      @ (s @ zero)) = (s @ zero)) => (A = zero)) & (A = (s @ zero)))) != (s @ zero))),
      inference(rewrite,[status(thm)],[17,153])).
thf(155,plain,(((@+ [A:$i]: ((((s @ zero) = zero) => (A = (s @ zero))) & (A = zero) &
      (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) != zero) | ((@+ [A:$i]:
      ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s @ (s @ zero)) = (s @ zero)
      ) => (A = zero)) & (A = (s @ zero)))) != (s @ zero))),inference(simp,[status(thm)
      ],[154])).
thf(36,plain,(! [A:$i] : ((~ ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s @
      (s @ zero)) = (s @ zero)) => (A = zero)) & (A = (s @ zero)))) | ((((s @ (s @ zero)
      ) = zero) => ((@+ [B:$i]: ((((s @ (s @ zero)) = zero) => (B = (s @ zero))) & (((s
      @ (s @ zero)) = (s @ zero)) => (B = zero)) & (B = (s @ zero)))) = (s @ zero))) &
      (((s @ (s @ zero)) = (s @ zero)) => ((@+ [B:$i]: ((((s @ (s @ zero)) = zero) => (B
       = (s @ zero))) & (((s @ (s @ zero)) = (s @ zero)) => (B = zero)) & (B = (s @ zero
      )))) = zero)) & ((@+ [B:$i]: ((((s @ (s @ zero)) = zero) => (B = (s @ zero))) &
      (((s @ (s @ zero)) = (s @ zero)) => (B = zero)) & (B = (s @ zero)))) = (s @ zero))
      ))),inference(choice,[status(esa)],[33])).
thf(74,plain,(! [A:$i] : (((@+ [B:$i]: ((((s @ (s @ zero)) = zero) => (B = (s @ zero)))
       & (((s @ (s @ zero)) = (s @ zero)) => (B = zero)) & (B = (s @ zero)))) = (s @
      zero)) | (~ (A = (s @ zero))) | ((s @ (s @ zero)) = (s @ zero)) | (~ (A = (s @
      zero))))),inference(cnf,[status(esa)],[36])).
thf(91,plain,(! [A:$i] : (((@+ [B:$i]: ((((s @ (s @ zero)) = zero) => (B = (s @ zero)))
       & (((s @ (s @ zero)) = (s @ zero)) => (B = zero)) & (B = (s @ zero)))) = (s @
      zero)) | (A != (s @ zero)) | ((s @ (s @ zero)) = (s @ zero)) | (A != (s @ zero))))
      ,inference(lifteq,[status(thm)],[74])).
thf(92,plain,(((@+ [A:$i]: ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s @ (s
       @ zero)) = (s @ zero)) => (A = zero)) & (A = (s @ zero)))) = (s @ zero)) | ((s @
      (s @ zero)) = (s @ zero))),inference(simp,[status(thm)],[91])).
thf(6,plain,(! [A:$i] : ((~ ((s @ A) = A)))),inference(cnf,[status(esa)],[4])).
thf(9,plain,(! [A:$i] : (((s @ A) != A))),inference(lifteq,[status(thm)],[6])).
thf(372,plain,(((@+ [A:$i]: ((((s @ (s @ zero)) = zero) => (A = (s @ zero))) & (((s @ (
      s @ zero)) = (s @ zero)) => (A = zero)) & (A = (s @ zero)))) = (s @ zero))),
      inference(simplifyReflect,[status(thm)],[92,9])).
thf(373,plain,(((@+ [A:$i]: ((((s @ zero) = zero) => (A = (s @ zero))) & (A = zero) &
      (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) != zero) | ((s @ zero) !=
      (s @ zero))),inference(rewrite,[status(thm)],[155,372])).
thf(374,plain,(((@+ [A:$i]: ((((s @ zero) = zero) => (A = (s @ zero))) & (A = zero) &
      (((s @ zero) = (s @ (s @ zero))) => (A = (s @ zero))))) != zero)),inference(simp,[
      status(thm)],[373])).
thf(446,plain,($false),inference(simplifyReflect,[status(thm)],[150,374,9,16])).
% SZS output end CNFRefutation for -
```

C.8. Proof of Polymorphic Cantor

```
% SZS status Theorem for sur_cantor_th1.p : 1948 ms resp. 1293 ms w/o parsing
% SZS output start CNFRefutation for sur_cantor_th1.p
thf(skt1_type, type, skt1: $tType).
thf(sk1_type, type, sk1: (skt1 > (skt1 > $o))).
thf(sk2_type, type, sk2: ((skt1 > $o) > skt1)).
thf(1,conjecture,((! [TA: $tType]: (~ (? [A:(TA > (TA > $o))]: ! [B:(TA > $o)]: ? [C:TA
      ]: ((A @ C) = B))))),file('/home/lex/dev/Leo-III/src/test/resources/th1/
      sur_cantor_th1.p',sur_cantor)).
thf(2,negated_conjecture,((~ (! [TA: $tType]: (~ (? [A:(TA > (TA > $o))]: ! [B:(TA > $o
      )]: ? [C:TA]: ((A @ C) = B)))))),inference(neg_conjecture,[status(cth)],[1])).
thf(3,plain,((~ (! [TA: $tType]: (~ (? [A:(TA > (TA > $o))]: ! [B:(TA > $o)]: ? [C:TA]:
       ((A @ C) = (B))))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(4,plain,(! [A:(skt1 > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(cnf,[status(esa
      )],[3])).
thf(5,plain,(! [A:(skt1 > $o)] : (((sk1 @ (sk2 @ A)) = (A)))),inference(lifteq,[status(
      thm)],[4])).
thf(6,plain,(! [B:skt1,A:(skt1 > $o)] : (((sk1 @ (sk2 @ (A)) @ B) = (A @ B)))),
      inference(func_ext,[status(esa)],[5])).
thf(8,plain,(! [B:skt1,A:(skt1 > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | (~ (A @ B)))),
      inference(bool_ext,[status(thm)],[6])).
thf(198,plain,(! [B:skt1,A:(skt1 > $o)] : ((sk1 @ (sk2 @ (A)) @ B) | ((A @ B) != (~ (
      sk1 @ (sk2 @ (A)) @ B))) | ~ ($true))),inference(eqfactor_ordered,[status(thm)
      ],[8])).
thf(217,plain,((sk1 @ (sk2 @ (^ [A:skt1]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:skt1]: ~ (
      sk1 @ A @ A))))),inference(pre_uni,[status(thm)],[198:[bind(A, $thf(^ [C:skt1]: ~
      (sk1 @ C @ C))),bind(B, $thf(sk2 @ (^ [C:skt1]: ~ (sk1 @ C @ C))))]])).
thf(7,plain,(! [B:skt1,A:(skt1 > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | (A @ B))),
      inference(bool_ext,[status(thm)],[6])).
thf(17,plain,(! [B:skt1,A:(skt1 > $o)] : ((~ (sk1 @ (sk2 @ (A)) @ B)) | ((A @ B) != (~
      (sk1 @ (sk2 @ (A)) @ B))) | ~ ($true))),inference(eqfactor_ordered,[status(thm)
      ],[7])).
thf(29,plain,((~ (sk1 @ (sk2 @ (^ [A:skt1]: ~ (sk1 @ A @ A))) @ (sk2 @ (^ [A:skt1]: ~ (
      sk1 @ A @ A)))))),inference(pre_uni,[status(thm)],[17:[bind(A, $thf(^ [C:skt1]: ~
      (sk1 @ C @ C))),bind(B, $thf(sk2 @ (^ [C:skt1]: ~ (sk1 @ C @ C))))]])).
thf(225,plain,($false),inference(rewrite,[status(thm)],[217,29])).
thf(226,plain,($false),inference(simp,[status(thm)],[225])).
% SZS output end CNFRefutation for sur_cantor_th1.p
```

C.9. Proof of `SEV485^1`

```
% SZS status Theorem for SEV485^1.p : 2804 ms resp. 1152 ms w/o parsing
% SZS output start CNFRefutation for SEV485^1.p
thf('type/nums/num_type', type, 'type/nums/num': $tType).
thf('const/sets/UNIV_type', type, 'const/sets/UNIV': !>[TA: $tType]: (TA > $o)).
thf('const/sets/HAS_SIZE_type', type, 'const/sets/HAS_SIZE': !>[TA: $tType]: ((TA > $o)
       > ('type/nums/num' > $o))).
thf('const/sets/FINITE_type', type, 'const/sets/FINITE': !>[TA: $tType]: ((TA > $o) >
      $o)).
thf('const/sets/CARD_type', type, 'const/sets/CARD': !>[TA: $tType]: ((TA > $o) > 'type
      /nums/num')).
thf('const/nums/NUMERAL_type', type, 'const/nums/NUMERAL': ('type/nums/num' > 'type/
      nums/num')).
thf('const/nums/BIT1_type', type, 'const/nums/BIT1': 'type/nums/num'>'type/nums/num').
thf('const/nums/BIT0_type', type, 'const/nums/BIT0': 'type/nums/num'>'type/nums/num').
thf('const/nums/_0_type', type, 'const/nums/_0': 'type/nums/num').
thf(3,axiom,(('const/sets/HAS_SIZE' @ $o @ ('const/sets/UNIV' @ $o) @ ('const/nums/
      NUMERAL' @ ('const/nums/BIT0' @ ('const/nums/BIT1' @ 'const/nums/_0'))))),file('/
      home/lex/TPTP/TH1/SEV/SEV485^1.p','thm/sets/HAS_SIZE_BOOL_')).
thf(6,plain,(('const/sets/HAS_SIZE' @ $o @ ('const/sets/UNIV' @ $o) @ ('const/nums/
      NUMERAL' @ ('const/nums/BIT0' @ ('const/nums/BIT1' @ 'const/nums/_0'))))),
      inference(defexp_and_simp_and_etaexpand,[status(thm)],[3])).
thf(4,axiom,((![TA: $tType]: (! [A:(TA > $o),B:'type/nums/num']: (('const/sets/HAS_SIZE
      ' @ TA @ A @ B) = (('const/sets/FINITE' @ TA @ A) & (('const/sets/CARD' @ TA @ A)
      = B)))))),file('/home/lex/TPTP/TH1/SEV/SEV485^1.p','thm/sets/HAS_SIZE_')).
thf(7,plain,((![TA: $tType]: (! [A:(TA > $o),B:'type/nums/num']: (('const/sets/HAS_SIZE
      ' @ TA @ (A) @ B) = (('const/sets/FINITE' @ TA @ A) & (('const/sets/CARD' @ TA @ A
      ) = B)))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[4])).
thf(8,plain,(![TA:$tType,B:'type/nums/num',A:(TA > $o)] : ((('const/sets/HAS_SIZE' @ TA
       @ (A) @ B) = (('const/sets/FINITE' @ TA @ (A)) & (('const/sets/CARD' @ TA @ (A))
      = B))))),inference(cnf,[status(esa)],[7])).
thf(9,plain,(![TA:$tType,B:'type/nums/num',A:(TA > $o)] : (((('const/sets/FINITE' @ TA
      @ (A)) & (('const/sets/CARD' @ TA @ (A)) = B)) = ('const/sets/HAS_SIZE' @ TA @ (A)
       @ B)))),inference(lifteq,[status(thm)],[8])).
thf(11,plain,(![TA:$tType,B:'type/nums/num',A:(TA > $o)] : ((('const/sets/FINITE' @ TA
      @ (A)) & (('const/sets/CARD' @ TA @ (A)) = B)) | (~ ('const/sets/HAS_SIZE' @ TA @
      (A) @ B)))),inference(bool_ext,[status(thm)],[9])).
thf(14,plain,![TA:$tType,B:'type/nums/num',A:(TA > $o)]: ((~ ('const/sets/HAS_SIZE' @
      TA @ A @ B)) | ('const/sets/FINITE' @ TA @ A)),inference(cnf,[status(esa)],[11])).
thf(23,plain,(![TA:$tType,B:'type/nums/num',A:(TA > $o)]: (('const/sets/FINITE' @ TA @
      A) | (('const/sets/HAS_SIZE' @ $o @ ('const/sets/UNIV' @ $o) @ ('const/nums/
      NUMERAL' @ ('const/nums/BIT0' @ ('const/nums/BIT1' @ 'const/nums/_0')))) != ('
      const/sets/HAS_SIZE' @ TA @ A @ B)))),inference(paramod_ordered,[status(thm)
      ],[6,14])).
thf(24,plain,'const/sets/FINITE' @ $o @ ('const/sets/UNIV' @ $o),inference(pattern_uni
      ,[status(thm)],[23:[bind(A, $thf('const/sets/UNIV' @ $o)),bind(B, $thf('const/nums
      /NUMERAL' @ ('const/nums/BIT0' @ ('const/nums/BIT1' @ 'const/nums/_0'))))]])).
thf(1,conjecture,(('const/sets/FINITE' @ $o @ ('const/sets/UNIV' @ $o))),file('/home/
      lex/TPTP/TH1/SEV/SEV485^1.p','thm/sets/FINITE_BOOL_')).
thf(2,negated_conjecture,((~ ('const/sets/FINITE' @ $o @ ('const/sets/UNIV' @ $o)))),
      inference(neg_conjecture,[status(cth)],[1])).
thf(5,plain,((~ ('const/sets/FINITE' @ $o @ ('const/sets/UNIV' @ $o)))),inference(
      defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(26,plain,($false),inference(simplifyReflect,[status(thm)],[24,5])).
% SZS output end CNFRefutation for SEV485^1.p
```

C.10. Proof of the Barcan Formula

```
% SZS status Theorem for bf.p : 2722 ms resp. 1054 ms w/o parsing
% SZS output start CNFRefutation for bf.p
thf(mworld_type, type, mworld: $tType).
thf(mrel_type, type, mrel: (mworld > (mworld > $o))).
thf(mvalid_type, type, mvalid: ((mworld > $o) > $o)).
thf(mvalid_def, definition, (mvalid = (^[A: mworld > $o]: (![B: mworld]: (A @ B))))).
thf(mimplies_type, type, mimplies: ((mworld > $o) > ((mworld > $o) > (mworld > $o)))).
thf(mimplies_def, definition, (mimplies = (^ [A:(mworld > $o),B:(mworld > $o),C:mworld
      ]: ((A @ C) => (B @ C))))).
thf(mbox_type, type, mbox: ((mworld > $o) > (mworld > $o))).
thf(mbox_def, definition, (mbox = (^ [A:(mworld > $o),B:mworld]: ! [C:mworld]: ((mrel @
       B @ C) => (A @ C))))).
thf(eiw__d_i_type, type, eiw__d_i: ($i > (mworld > $o))).
thf(mforall_vary__d_i_type, type, mforall_vary__d_i: (($i > (mworld > $o)) > (mworld >
      $o))).
thf(mforall_vary__d_i_def, definition, (mforall_vary__d_i = (^ [A:($i > (mworld > $o)),
      B:mworld]: ! [C:$i]: ((eiw__d_i @ C @ B) => (A @ C @ B))))).
thf(p_type, type, p: ($i > (mworld > $o))).
thf(sk1_type, type, sk1: mworld).
thf(sk2_type, type, sk2: mworld).
thf(sk3_type, type, sk3: $i).
thf(1,conjecture,((mvalid @ (mimplies @ (mforall_vary__d_i @ (^ [A:$i]: (mbox @ (p @ A)
      ))) @ (mbox @ (mforall_vary__d_i @ (p)))))),file('/home/lex/dev/temp/bf.p',1)).
thf(2,negated_conjecture,((~ (mvalid @ (mimplies @ (mforall_vary__d_i @ (^ [A:$i]: (
      mbox @ (p @ A)))) @ (mbox @ (mforall_vary__d_i @ (p))))))),inference(
      neg_conjecture,[status(cth)],[1])).
thf(7,plain,((~ (! [A:mworld]: ((! [B:$i]: ((eiw__d_i @ B @ A) => (! [C:mworld]: ((mrel
       @ A @ C) => (p @ B @ C))))) => (! [B:mworld]: ((mrel @ A @ B) => (! [C:$i]: ((
      eiw__d_i @ C @ B) => (p @ C @ B))))))))),inference(defexp_and_simp_and_etaexpand,[
      status(thm)],[2,mbox_def, mforall_vary__d_i_def, mvalid_def, mimplies_def])).
thf(9,plain,((eiw__d_i @ sk3 @ sk2)),inference(cnf,[status(esa)],[7])).
thf(5,axiom,((! [A:mworld,B:mworld,C:$i]: ((mrel @ A @ B) => ((eiw__d_i @ C @ B) => (
      eiw__d_i @ C @ A))))),file('/home/lex/dev/temp/bf.p',eiw_decre__d_i_r)).
thf(16,plain,((! [A:mworld,B:mworld,C:$i]: ((mrel @ A @ B) => ((eiw__d_i @ C @ B) => (
      eiw__d_i @ C @ A))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[5])).
thf(17,plain,((! [A:mworld,B:mworld]: ((mrel @ A @ B) => (! [C:$i]: ((eiw__d_i @ C @ B)
       => (eiw__d_i @ C @ A)))))),inference(miniscope,[status(thm)],[16])).
thf(18,plain,(! [C:$i,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (~ (eiw__d_i @ C @ B))
       | (eiw__d_i @ C @ A))),inference(cnf,[status(esa)],[17])).
thf(45,plain,(! [C:$i,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (eiw__d_i @ C @ A) |
      ((eiw__d_i @ sk3 @ sk2) != (eiw__d_i @ C @ B)))),inference(paramod_ordered,[status
      (thm)],[9,18])).
thf(46,plain,(! [A:mworld] : ((~ (mrel @ A @ sk2)) | (eiw__d_i @ sk3 @ A))),inference(
      pattern_uni,[status(thm)],[45:[bind(A, $thf(A)),bind(B, $thf(sk2)),bind(C, $thf(
      sk3))]])).
thf(10,plain,((mrel @ sk1 @ sk2)),inference(cnf,[status(esa)],[7])).
thf(11,plain,(! [B:mworld,A:$i] : ((~ (eiw__d_i @ A @ sk1)) | (~ (mrel @ sk1 @ B)) | (p
       @ A @ B))),inference(cnf,[status(esa)],[7])).
thf(21,plain,(! [B:mworld,A:$i] : ((~ (eiw__d_i @ A @ sk1)) | (p @ A @ B) | ((mrel @
      sk1 @ sk2) != (mrel @ sk1 @ B)))),inference(paramod_ordered,[status(thm)],[10,11])
      ).
thf(22,plain,(! [A:$i] : ((~ (eiw__d_i @ A @ sk1)) | (p @ A @ sk2))),inference(
      pattern_uni,[status(thm)],[21:[bind(A, $thf(A)),bind(B, $thf(sk2))]])).
```

```
thf(75,plain,(! [B:$i,A:mworld] : ((~ (mrel @ A @ sk2)) | (p @ B @ sk2) | ((eiw__d_i @
      sk3 @ A) != (eiw__d_i @ B @ sk1)))),inference(paramod_ordered,[status(thm)
      ],[46,22])).
thf(76,plain,((~ (mrel @ sk1 @ sk2)) | (p @ sk3 @ sk2)),inference(pattern_uni,[status(
      thm)],[75:[bind(A, $thf(sk1)),bind(B, $thf(sk3))]])).
thf(8,plain,((~ (p @ sk3 @ sk2))),inference(cnf,[status(esa)],[7])).
thf(83,plain,(~ ($true) | $false),inference(rewrite,[status(thm)],[76,10,8])).
thf(84,plain,($false),inference(simp,[status(thm)],[83])).
% SZS output end CNFRefutation for bf.p
```

C.11. Proof of the Converse Barcan Formula

```
% SZS status Theorem for cbf.p : 2824 ms resp. 999 ms w/o parsing
% SZS output start CNFRefutation for cbf.p
thf(mworld_type, type, mworld: $tType).
thf(mrel_type, type, mrel: mworld > mworld > $o).
thf(mvalid_type, type, mvalid: (mworld > $o) > $o).
thf(mvalid_def, definition, mvalid = (^[A: mworld > $o]: (![B: mworld]: (A @ B)))).
thf(mimplies_type, type, mimplies: (mworld > $o) > (mworld > $o) > mworld > $o).
thf(mimplies_def, definition, mimplies = (^ [A:(mworld > $o),B:(mworld > $o),C:mworld]:
       ((A @ C) => (B @ C)))).
thf(mbox_type, type, mbox: (mworld > $o) > mworld > $o).
thf(mbox_def, definition, mbox = (^ [A:(mworld > $o),B:mworld]: ! [C:mworld]: ((mrel @
      B @ C) => (A @ C)))).
thf(eiw__d_i_type, type, eiw__d_i: ($i > (mworld > $o))).
thf(mforall_vary__d_i_type, type, mforall_vary__d_i: ($i > mworld > $o) > mworld > $o).
thf(mforall_vary__d_i_def, definition, (mforall_vary__d_i = (^ [A:($i > (mworld > $o)),
      B:mworld]: ! [C:$i]: ((eiw__d_i @ C @ B) => (A @ C @ B))))).
thf(p_type, type, p: ($i > (mworld > $o))).
thf(sk1_type, type, sk1: mworld). thf(sk2_type, type, sk2: $i).
thf(sk3_type, type, sk3: mworld).
thf(1,conjecture,((mvalid @ (mimplies @ (mbox @ (mforall_vary__d_i @ (p))) @ (
      mforall_vary__d_i @ (^ [A:$i]: (mbox @ (p @ A))))))),file('cbf.p',1)).
thf(2,negated_conjecture,(~(mvalid @ (mimplies @ (mbox @ (mforall_vary__d_i @ p)) @ (
      mforall_vary__d_i @ (^ [A:$i]: (mbox @ (p @ A))))))),inference(neg_conjecture,[
      status(cth)],[1])).
thf(7,plain,((~ (! [A:mworld]: ((! [B:mworld]: ((mrel @ A @ B) => (! [C:$i]: ((eiw__d_i
       @ C @ B) => (p @ C @ B))))) => (! [B:$i]: ((eiw__d_i @ B @ A) => (! [C:mworld]:
      ((mrel @ A @ C) => (p @ B @ C))))))))),inference(defexp_and_simp_and_etaexpand,[
      status(thm)],[2,mbox_def, mforall_vary__d_i_def, mvalid_def, mimplies_def])).
thf(10,plain,((eiw__d_i @ sk2 @ sk1)),inference(cnf,[status(esa)],[7])).
thf(5,axiom,((! [A:mworld,B:mworld,C:$i]: ((mrel @ A @ B) => ((eiw__d_i @ C @ A) => (
      eiw__d_i @ C @ B))))),file('cbf.p',eiw_cumul__d_i_r)).
thf(16,plain,((! [A:mworld,B:mworld,C:$i]: ((mrel @ A @ B) => ((eiw__d_i @ C @ A) => (
      eiw__d_i @ C @ B))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[5])).
thf(17,plain,((! [A:mworld,B:mworld]: ((mrel @ A @ B) => (! [C:$i]: ((eiw__d_i @ C @ A)
       => (eiw__d_i @ C @ B)))))),inference(miniscope,[status(thm)],[16])).
thf(18,plain,(! [C:$i,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (~ (eiw__d_i @ C @ A))
       | (eiw__d_i @ C @ B))),inference(cnf,[status(esa)],[17])).
thf(11,plain,(! [B:$i,A:mworld] : ((~ (mrel @ sk1 @ A)) | (~ (eiw__d_i @ B @ A)) | (p @
       B @ A))),inference(cnf,[status(esa)],[7])).
thf(8,plain,((~ (p @ sk2 @ sk3))),inference(cnf,[status(esa)],[7])).
thf(25,plain,(! [B:$i,A:mworld] : ((~ (mrel @ sk1 @ A)) | (~ (eiw__d_i @ B @ A)) | ((p
      @ B @ A) != (p @ sk2 @ sk3)))),inference(paramod_ordered,[status(thm)],[11,8])).
thf(26,plain,((~ (mrel @ sk1 @ sk3)) | (~ (eiw__d_i @ sk2 @ sk3))),inference(
      pattern_uni,[status(thm)],[25:[bind(A, $thf(sk3)),bind(B, $thf(sk2))]])).
thf(9,plain,((mrel @ sk1 @ sk3)),inference(cnf,[status(esa)],[7])).
thf(28,plain,~$true | ~(eiw__d_i @ sk2 @ sk3),inference(rewrite,[status(thm)],[26,9])).
thf(29,plain,((~ (eiw__d_i @ sk2 @ sk3))),inference(simp,[status(thm)],[28])).
thf(41,plain,(! [C:$i,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (~ (eiw__d_i @ C @ A))
       | ((eiw__d_i @ C @ B) != (eiw__d_i @ sk2 @ sk3)))),inference(paramod_ordered,[
      status(thm)],[18,29])).
thf(42,plain,(![A:mworld]: ((~ (mrel @ A @ sk3)) | (~ (eiw__d_i @ sk2 @ A)))),inference
      (pattern_uni,[status(thm)],[41:[bind(B, $thf(sk3)),bind(C, $thf(sk2))]])).
thf(61,plain,(![A:mworld]: ((~ (mrel @ A @ sk3)) | ((eiw__d_i @ sk2 @ sk1) != (eiw__d_i
       @ sk2 @ A)))),inference(paramod_ordered,[status(thm)],[10,42])).
```

```
thf(62,plain,((~ (mrel @ sk1 @ sk3))),inference(pattern_uni,[status(thm)],[61:[bind(A,
      $thf(sk1))]])).
thf(70,plain,($false),inference(simp,[status(thm)],inference(rewrite,[status(thm)
      ],[62,9]))).
% SZS output end CNFRefutation for cbf.p
```

D. Deutsche Zusammenfassung

In der vorliegenden Dissertation werden sowohl die theoretischen Grundlagen als auch Implementierungstechniken für die Entwicklung eines effektiven automatischen Theorembeweisers für Prädikatenlogik höherer Stufe präsentiert. Ein Hauptaugenmerk der Arbeit liegt dabei auf der Demonstration der Machbarkeit, ein performantes Theorembeweisersystem für das automatische Schließen in gleichheitsbasierter extensionaler Typentheorie (hier gleichbedeutend mit Prädikatenlogik höherer Stufe) mit Hilfe eines Paramodulationskalküls zu implementieren. Zu diesem Zweck wird ein korrekter und vollständiger Paramodulationskalkül für extensionale Typentheorie unter Henkinsemantik erarbeitet. Für den Vollständigkeitsbeweis wurden bereits existierende Beweistechniken der abstrakten Konsistenz vereinheitlicht und, für den hier präsentierten gleichheitsbasierten Ansatz, vereinfacht.

Der praktisch motivierte Hauptteil der Arbeit diskutiert die Softwarearchitektur und Implementierung des neuen, auf dem zuvor entwickelten Paramodulationskalkül basierten, Theorembeweiser Leo-III. Dabei implementiert Leo-III eine um weitgehend praktische Aspekte erweiterte Version des Kalküls und umfasst z.B. gleichheitsbasierte Simplifikationtechniken, heuristische Termersetzung und Unterstützung für das Schließen mit Auswahlfunktionen. Leo-III umfasst als Deduktionsplattform zudem ein flexibles, asynchrones Kommunikationssystem für die Kooperation mit externen Systemen, insbesondere mit Theorembeweisern für Prädikatenlogik erster Stufe. Das System implementiert fortschrittliche Beweissuchemethoden und basiert diese auf effizienten Datenstrukturen. Außerdem werden weitere, praktisch relevante Anwendungen und Fähigkeiten von Leo-III skizziert, darunter die Unterstützung von polymorpher Typentheorie und das Schließen in allen normalen quantifizierten Modallogiken. Die Effektivität von Leo-III wird durch eine breite Evaluation auf verschiedenen Datensätzen untersucht. Die Ergebnisse dieser Evaluation bestätigen dass Leo-III zu den aktuell stärksten Theorembeweisern für höherstufige Prädikatenlogik zählt und zudem für ein breites Spektrum von Anwendungen nutzbar ist.

E. About the Author

Alexander Steen was born in 1990 in Cuxhaven where he also grew up and finished his A-levels at Lichtenberg-Gymnasium. In 2014, Alexander graduated from Freie Universität Berlin with a Bachelor's and Master's degree in Computer Science and a Bachelor's degree in Mathematics. Since his late undergraduate studies, he served as tutor for computer science and was awarded a scholarship of the German Academic Scholarship Foundation (*Studienstiftung des Deutschen Volkes*). Alexander joined the Dahlem Center for Robotics and Machine Learning at the Institute of Computer Science of Freie Universität Berlin as a research assistant for the Leo-III project which was funded by the German Research Foundation (*Deutsche Forschungsgemeinschaft*). Since February 2018, Alexander works as a research assistant for the project "Consistent Rational Argumentation in Politics" funded by the Volkswagenstiftung.

During his time at Freie Universität Berlin, Alexander served in several committees of academic administration at the Department of Mathematics and Computer Science and the Institute of Computer Science. He was strongly involved in university teaching, successfully participated in advanced training for university teaching (*Hochschuldidaktisches Lehrzertifikat*) and was awarded the central teaching award (*Zentraler Lehrpreis*) of Freie Universität in 2015 for the conception of a novel, interdisciplinary lecture on Computational Metaphysics. The automated theorem prover Leo-III, which Alexander implemented as one of the main developers during his PhD studies, won the 2nd place in the 2017 international CADE ATP System Competition (CASC), THF category.

List of Selected Publications

Core publications related to the dissertation project

10. Alexander Steen, Christoph Benzmüller, *The Higher-Order Prover Leo-III*. In Didier Galmiche, Stephan Schulz, Roberto Sebastiani (Eds.), Automated Reasoning — 9th International Joint Conference, IJCAR 2018, Oxford, UK, July 14-17, 2018, Proceedings , Springer, LNCS, 2018. (To appear)
9. Tobias Gleißner, Alexander Steen, Christoph Benzmüller, *Theorem Provers for Every Normal Modal Logic*. In Thomas Eiter, David Sands (Eds.), LPAR-21. 21st International Conference on Logic for Programming,

Artificial Intelligence and Reasoning, EasyChair, EPiC Series in Computing, Volume 46, pp. 14-30, 2017.

8. Alexander Steen, Max Wisniewski, Christoph Benzmüller, *Going Polymorphic - TH1 Reasoning for Leo-III*. In Thomas Eiter, David Sands, Geoff Sutcliffe and Andrei Voronkov (Eds.), IWIL Workshop and LPAR Short Presentations, EasyChair, Kalpa Publications in Computing, Volume 1, pp. 100-112, 2017.
7. Christoph Benzmüller, Alexander Steen, Max Wisniewski, *Leo-III Version 1.1 (System description)*. In Thomas Eiter, David Sands, Geoff Sutcliffe and Andrei Voronkov (Eds.), IWIL Workshop and LPAR Short Presentations, EasyChair, Kalpa Publications in Computing, Volume 1, pp. 11-26, 2017.
6. Tomer Libal, Alexander Steen, *Towards a Substitution Tree Based Index for Higher-order Resolution Theorem Provers*. In 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016), Coimbra, Portugal, July 2016, Proceedings. CEUR Workshop Proceedings, Volume 1653, CEUR-WS.org, 2016.
5. Max Wisniewski, Alexander Steen, Christoph Benzmüller, *TPTP and Beyond: Representation of Quantified Non-Classical Logics*. In C. Benzmüller, J. Otten (Eds.), 2nd International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARNQL 2016), Coimbra, Portugal, July 2016, Proceedings, CEUR Workshop Proceedings, Volume 1770, CEUR-WS.org, 2016.
4. Alexander Steen, Max Wisniewski, Christoph Benzmüller, *Agent-Based HOL Reasoning*. In 5th International Congress on Mathematical Software, ICMS 2016, Berlin, Germany, July 2016, Proceedings, Springer, LNCS, volume 9725, 2016.
3. Max Wisniewski, Alexander Steen, Kim Kern, Christoph Benzmüller, *Effective Normalization Techniques for HOL*. In Nicola Olivetti, Ashish Tiwari (Eds.), 8th International Joint Conference on Automated Reasoning, IJCAR 2016, Coimbra, Portugal, 27 June - 2 July, 2016, Proceedings. Springer, LNAI, volume 9706, 2016.
2. Alexander Steen, Christoph Benzmüller, *There Is No Best Beta-Normalization Strategy for Higher-Order Reasoners*. In Martin Davis, Ansgar Fehnker, Annabelle CcIver, Andrei Voronkov (Eds.), 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Suva, Fiji, November 2015, Proceedings, Springer, LNCS/ARCoSS, volume 9450, 2015.
1. Max Wisniewski, Alexader Steen, Christoph Benzmüller, LEOPARD –

A Generic Platform for the Implementation of Higher-Order Reasoners (Project Paper). In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, Volker Sorge (Eds.), Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings , Springer, LNCS, volume 9150, pp. 325–330, 2015.

Further related publications

6. Alexander Steen, Christoph Benzmüller, *System Demonstration: The Higher-Order Prover Leo-III.* In Christoph Benzmüller, Jens Otten (Eds.), ARQNL 2018. Automated Reasoning in Quantified Non-Classical Logics, Proceedings, CEUR Workshop Proceedings, http://ceur-ws.org, volume 2095. (To appear)
5. Alexander Steen, Max Wisniewski, Hans-Jörg Schurr, Christoph Benzmüller, *Capability Discovery for Automated Reasoning Systems.* In Thomas Eiter, David Sands, Geoff Sutcliffe and Andrei Voronkov (Eds.), IWIL Workshop and LPAR Short Presentations, EasyChair, Kalpa Publications in Computing, Volume 1, pp. 113-118, 2017.
4. Alexander Steen, Max Wisniewski, Christoph Benzmüller, *Tutorial on Reasoning in Expressive Non-Classical Logics with Isabelle/HOL.* In Christoph Benzüller, Raul Rojas, Geoff Sutcliffe (Eds.), Second Global Conference on Artificial Intelligence (GCAI), Proceedings, EasyChair, EPiC Series in Computing, Volume 41, pp. 1-10, 2016.
3. Alexander Steen, Christoph Benzmüller, *Sweet SIXTEEN: Automation via Embedding into Classical Higher-Order Logic.* In Logic and Logical Philosophy, Volume 25, No 4, Nicolaus Copernicus University, 2016.
2. Alexander Steen, Max Wisniewski, Christoph Benzmüller, *Einsatz von Theorembeweisern in der Lehre.* In Andreas Schwill, Ulrike Lucke (Eds.), Hochschuldidaktik der Informatik, HDI2016 – 7. Fachtagung des GI-Fachbereichs Informatik und Ausbildung / Didaktik der Informatik ; 13.-14. September 2016 an der Universität Potsdam, Commentarii Informaticae Didacticae, Volume 10, pages 81 - 92, Universitätsverlag Potsdam, 2016.
1. Max Wisniewski, Alexander Steen, *Embedding of Quantified Higher-Order Nominal Modal Logic into Classical Higher-Order Logic.* In Christoph Benzmüller, Jens Otten (Eds.), 1st International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2014), Vienna, Austria, Proceedings, EasyChair, EasyChair Proceedings in Computing, volume 33, pp. 59–64, 2014.

List of Figures

List of Tables

Index

Bibliography

[AB06] Peter. B. Andrews and Chad E. Brown. Tps: A hybrid automatic-interactive system for developing proofs. *Journal of Applied Logic*, 4(4):367 – 395, 2006. Towards Computer Aided Mathematics.

[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. *J. Autom. Reasoning*, 16(3):321–353, 1996.

[ACCL90] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 31–46, New York, NY, USA, 1990. ACM.

[AG09] Pietro Abate and Rajeev Goré. The Tableau Workbench. *Electronic Notes in Theoretical Computer Science*, 231:55–67, 2009.

[AH76] K. Appel and W. Haken. Every planar map is four colorable. *Bulletin of the American Mathematical Society*, 82(5):711–712, 09 1976.

[And63] Peter B. Andrews. A reduction of the axioms for the theory of propositional types. *Fundamenta Mathematicae*, 52:345–350, 1963.

[And65] Peter B. Andrews. *Transfinite Type Theory with Type Variables*. Studies in logic and the foundations of mathematics. North-Holland Pub. Co., 1965.

[And71] Peter B. Andrews. Resolution in type theory. *J. Symb. Log.*, 36(3):414–432, 1971.

[And72a] Peter B. Andrews. General models and extensionality. *J. Symb. Log.*, 37(2):395–397, 1972.

[And72b] Peter B. Andrews. General models, descriptions, and choice in type theory. *J. Symb. Log.*, 37(2):385–394, 1972.

[And74] Peter B. Andrews. Provability in elementary type theory. *Mathematical Logic Quarterly*, 20(25-27):411–418, 1974.

[And89] Peter B. Andrews. On connections and higher-order logic. *J. Autom. Reasoning*, 5(3):257–291, 1989.

[And02a] Peter B Andrews. *An introduction to mathematical logic and type theory*, volume 27. Springer Science & Business Media, 2002.

[And02b] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory*. Applied Logic Series. Springer, 2002.

[Ans78] St. Anselm. *St Anselm's Proslogion*. Oxford:OUP, 1078. Republished in 1965.

[B$^+$08] François Bobot et al. The Alt-Ergo automated theorem prover, 2008.

[B$^+$11] Clark Barrett et al. CVC4. In *Computer aided verification*, pages 171–177. Springer, 2011.

[Bac10] Julian Backes. Tableaux for Higher-Order Logic with If-Then-Else, Description and Choice. Master's thesis, Saarland University, Saarbruecken, Germany, 2010.

[Bar81] Henk P. Barendregt. The lambda calculus: Its syntax and semantics, volume 103 of studies in logic and the foundations of computer science, 1981.

[BB07] Christoph Benzmüller and Chad Brown. The curious inference of Boolos in MIZAR and OMEGA. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof – Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*, pages 299–388. The University of Bialystok, Polen, 2007.

[BBCW18] Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. `http://matryoshka.gforge.inria.fr/pubs/lfhosup_paper.pdf`, 2018.

[BBK04a] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.

[BBK04b] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Semantic techniques for cut-elimination in higher order logic. Technical report, Saarland University, Saarbrücken, Germany and International University Bremen, Germany, 2004. SEKI Report SR-2004-07.

[BBK09] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1:6):1–21, 2009.

[BBP13] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C Paulson. Extending sledgehammer with smt solvers. *Journal of automated reasoning*, 51(1):109–128, 2013.

[BBPS16] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. *Logical Methods in Computer Science*, 12(4), 2016.

[BBWW17] Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A transfinite knuth-bendix order for lambda-free higher-order terms. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 432–453. Springer, 2017.

[BCH15] Guillaume Bury, Raphaël Cauderlier, and Pierre Halmagrand. Implementing polymorphism in zenon. In Boris Konev, Stephan Schulz, and Laurent Simon, editors, *IWIL@LPAR 2015, 11th International Workshop on the Implementation of Logics, Suva, Fiji, November 23, 2015*, volume 40 of *EPiC Series in Computing*, pages 15–20. EasyChair, 2015.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BDN09] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of agda– a functional language with dependent types. In *International Conference on Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[BDS13] Henk P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

[Bec30] Oskar Becker. *Zur Logik der Modalitäten*. Max Niemeyer Verlag, 1930.

[Ben99a] Christoph Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving*. PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Saarland University, Saarbrücken, Germany, 1999.

[Ben99b] Christoph Benzmüller. Extensional higher-order paramodulation and RUE-resolution. In Harald Ganzinger, editor, *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, number 1632 in LNCS, pages 399–413. Springer, 1999.

[Ben09] Christoph Benzmüller. Automating access control logic in simple type theory with LEO-II. In Dimitris Gritzalis and Javier López, editors, *Emerging Challenges for Security, Privacy and Trust, 24th IFIP TC 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18-20, 2009. Proceedings*, volume 297 of *IFIP*, pages 387–398. Springer, 2009.

[Ben11] Christoph Benzmüller. Combining and automating classical and non-classical logics in classical higher-order logic. *Annals of Mathematics and Artificial Intelligence (Special issue Computational logics in Multi-agent Systems (CLIMA XI))*, 62(1-2):103–128, 2011.

[Ben15a] Christoph Benzmüller. Higher-order automated theorem provers. In David Delahaye and Bruno Woltzenlogel Paleo, editors, *All about Proofs, Proof for All*, Mathematical Logic and Foundations, pages 171–214. College Publications, London, UK, 2015.

[Ben15b] Christoph Benzmüller. HOL provers for first-order modal logics — experiments. In Christoph Benzmüller and Jens Otten, editors, *ARQNL 2014. Automated Reasoning in Quantified Non-Classical Logics*, volume 33 of *EPiC Series in Computing*, pages 37–41. EasyChair, 2015.

[Ben15c] Christoph Benzmüller. Invited talk: On a (quite) universal theorem proving approach and its application in metaphysics. In Hans De Nivelle, editor, *TABLEAUX 2015*, volume 9323 of *LNAI*, pages 213–220, Wroclaw, Poland, 2015. Springer. (Invited paper).

[Ben17a] Christoph Benzmüller. Cut-elimination for quantified conditional logic. *Journal of Philosophical Logic*, 46(3):333–353, 2017.

[Ben17b] Christoph Benzmüller. Recent successes with a meta-logical approach to universal logical reasoning (extended abstract). In Simone André da Costa Cavalheiro and José Luiz Fiadeiro, editors, *Formal Methods: Foundations and Applications - 20th Brazilian Symposium, SBMF 2017, Recife, Brazil, November 29 - December 1, 2017, Proceedings*, volume 10623 of *Lecture Notes in Computer Science*, pages 7–11. Springer, 2017.

[Ben17c] Christoph Benzmüller. Universal reasoning, rational argumentation and human-machine interaction. Technical report, CoRR, 2017. `http://arxiv.org/abs/1703.09620`.

[BFM$^+$06] Christoph Benzmüller, Armin Fiedler, Andreas Meier, Martin Pollet, and Jörg Siekmann. Omega. In Freek Wiedijk, editor, *The Sev-*

enteen Provers of the World, number 3600 in LNCS, pages 127–141. Springer, 2006.

[BFP18] Christoph Benzmüller, Ali Farjami, and Xavier Parent. A dyadic deontic logic in hol. In Cleo Condoravdi, Shyam Nair, and Jan Broersen, editors, *Deon 2018 — 14th International Conference on Deontic Logic and Normative Systems 3-6 July 2018, Utrecht, the Netherlands, 2018, Proceedings*. Springer, 2018. To appear.

[BFSW17] Jasmin Christian Blanchette, Pascal Fontaine, Stephan Schulz, and Uwe Waldmann. Towards strong higher-order automation for fast interactive verification. In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements, Gothenburg, Sweden, 6th August 2017*, volume 51 of *EPiC Series in Computing*, pages 16–23. EasyChair, 2017.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[BG90] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 1990.

[BG94] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press, 2001.

[BGK$^+$16] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for isabelle/hol. *J. Autom. Reasoning*, 57(3):219–244, 2016.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Bib87] Wolfgang Bibel. *Automated theorem proving, 2nd Edition*. Artificial intelligence. Vieweg, 1987.

[BJR15] Frédéric Blanqui, Jean-Pierre Jouannaud, and Albert Rubio. The computability path ordering. *Logical Methods in Computer Science*, 11(4), 2015.

[BK98a] Christoph Benzmüller and Michael Kohlhase. Extensional higher-order resolution. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, number 1421 in LNAI, pages 56–71. Springer, 1998.

[BK98b] Christoph Benzmüller and Michael Kohlhase. System description: LEO - A higher-order theorem prover. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, volume 1421 of *Lecture Notes in Computer Science*, pages 139–144. Springer, 1998.

[BM14] Christoph Benzmüller and Dale Miller. Automation of higher-order logic. In Dov M. Gabbay, Jörg H. Siekmann, and John Woods, editors, *Handbook of the History of Logic, Volume 9 — Computational Logic*, pages 215–254. North Holland, Elsevier, 2014.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[BN10a] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.

[BN10b] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement day. In *IJCAR*, volume 6173, pages 107–121. Springer, 2010.

[Böh12] Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

[Boo87] George Boolos. A curious inference. *Journal of Philosophical Logic*, 16(1):1–12, 1987.

[BOR12] Christoph Benzmüller, Jens Otten, and Thomas Raths. Implementing and evaluating provers for first-order modal logics. In Luc De Raedt, Christian Bessiere, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter Lucas, editors, *ECAI 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 163–168, Montpellier, France, 2012. IOS Press.

[BP11] François Bobot and Andrei Paskevich. Expressing polymorphic types in a many-sorted language. In *International Symposium on Frontiers of Combining Systems*, pages 87–102. Springer, 2011.

[BP13a] Christoph Benzmüller and Lawrence Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.

[BP13b] Jasmin C. Blanchette and A. Paskevich. TFF1: the TPTP typed first-order form with rank-1 polymorphism. In M. P. Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *LNCS*, pages 414–420. Springer, 2013.

[BPST15] Christoph Benzmüller, Lawrence C. Paulson, Nik Sultana, and Frank Theiß. The Higher-Order Prover LEO-II. *Journal of Automated Reasoning*, 55(4):389–404, 2015.

[BR12] Christoph Benzmüller and Thomas Raths. FMLtoHOL (version 1.0): Automating first-order modal logics with LEO-II and friends. Technical report, Freie Universität Berlin, Germany, 2012. arXiv:1207.6685.

[BR13] Christoph Benzmüller and Thomas Raths. HOL based first-order modal logic provers. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 127–136, Stellenbosch, South Africa, 2013. Springer.

[Bro12] Chad E. Brown. Satallax: An automatic higher-order prover. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*, IJCAR'12, pages 111–117, Berlin, Heidelberg, 2012. Springer-Verlag.

[Bro13] Chad E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. *J. Autom. Reasoning*, 51(1):57–77, 2013.

[Bru72] N. G. De Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *INDAG. MATH*, 34:381–392, 1972.

[BS13] Christoph Benzmüller and Nik Sultana. LEO-II version 1.5. In Jasmin Christian Blanchette and Josef Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series in Computing*, pages 2–10. EasyChair, 2013.

[BS16] Christoph Benzmüller and Dana Scott. Automating free logic in Isabelle/HOL. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016, 5th International Congress, Proceedings*, volume 9725 of *LNCS*, pages 43–50, Berlin, Germany, 2016. Springer.

[BSJK08] Christoph Benzmüller, Volker Sorge, Mateja Jamnik, and Manfred Kerber. Combined reasoning by automated cooperation. *Journal of Applied Logic*, 6(3):318–342, 2008.

[BVB07] Patrick Blackburn and Johan Van Benthem. 1 modal logic: a semantic perspective. In *Studies in Logic and Practical Reasoning*, volume 3, pages 1–84. Elsevier, 2007.

[BvBW06] Patrick Blackburn, Johan FAK van Benthem, and Frank Wolter. *Handbook of modal logic*, volume 3. Elsevier, 2006.

[BWB$^+$11] Jasmin Christian Blanchette, Tjark Weber, Mark Batty, Scott Owens, and Susmit Sarkar. Nitpicking c++ concurrency. In *Proceedings of the 13th international ACM SIGPLAN symposium on Principles and practices of declarative programming*, pages 113–124. ACM, 2011.

[BWP14] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Automating Gödel's ontological proof of God's existence with higher-order automated theorem provers. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 93 – 98. IOS Press, 2014.

[BWP15] Christoph Benzmüller and Bruno Woltzenlogel Paleo. On logic embeddings and Gödel's God. In Mihai Codescu, Razvan Diaconescu, and Ionut Tutu, editors, *Recent Trends in Algebraic Development Techniques: 22nd International Workshop, WADT 2014, Sinaia, Romania, September 4-7, 2014, Revised Selected Papers*, number 9563 in LNCS, pages 3–6, Sinaia, Romania, 2015. Springer. (Invited paper).

[BWP16] Christoph Benzmüller and Bruno Woltzenlogel Paleo. The inconsistency in Gödel's ontological argument: A success story for AI in metaphysics. In Subbarao Kambhampati, editor, *IJCAI 2016*, volume 1-3, pages 936–942. AAAI Press, 2016.

[BWP17] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Experiments in Computational Metaphysics: Gödel's proof of God's existence. *Savijnanam: scientific exploration for a spiritual paradigm. Journal of the Bhaktivedanta Institute*, 9:43–57, 2017.

[BWW17] Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A lambda-free higher-order recursive path order. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences* *on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 461–479, 2017.

[BWWP17] Christoph Benzmüller, Leon Weber, and Bruno Woltzenlogel Paleo. Computer-assisted analysis of the Anderson-Hájek controversy. *Logica Universalis*, 11(1):139–151, 2017.

[Chu32] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.

[Chu41] Alonzo Church. *The calculi of lambda-conversion*, volume 6. Princeton University Press, 1941.

[CK18] David M Cerna and Temur Kutsia. Higher-order equational pattern anti-unification [preprint]. *arXiv preprint arXiv:1801.07438*, 2018.

[CL07] Jean-François Couchot and Stéphane Lescuyer. Handling polymorphism in automated deduction. In *International Conference on Automated Deduction*, pages 263–278. Springer, 2007.

[CLS11] Koen Claessen, Ann Lillieström, and Nicholas Smallbone. Sort it out with monotonicity. In *International Conference on Automated Deduction*, pages 207–221. Springer, 2011.

[Coo76] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, October 1976.

[Coq94] Thierry Coquand. A new paradox in type theory. In *Proceedings of the Ninth International Congress of Logic, Methodology, and Philosophy of Science*, pages 7–14. Elsevier, 1994.

[CP03] I. Cervesato and F. Pfenning. A linear spine calculus. *J. Logic and Computation*, 13(5):639–688, 2003.

[CRSS95] David Cyrluk, Sreeranga Rajan, Natarajan Shankar, and Mandayam K Srivas. Effective theorem proving for hardware verification. In *Theorem Provers in Circuit Design*, pages 203–222. Springer, 1995.

[Cru15] Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. (Extensions de la Superposition pour l'Arithmétique Linéaire Entière, l'Induction Structurelle, et bien plus encore).* PhD thesis, École Polytechnique, Palaiseau, France, 2015.

[Cur30] Haskell B. Curry. Grundlagen der kombinatorischen logik. *American Journal of Mathematics*, 52(3):509–536, 1930.

[Dav83] Martin Davis. The prehistory and early history of automated deduction. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, pages 1–28. Springer, 1983.

[Dav01] Martin Davis. The early history of automated deduction. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 3–15. Elsevier and MIT Press, 2001.

[DB70] Nicolaas Govert De Bruijn. The mathematical language automath, its usage, and some of its extensions. In *Symposium on automatic demonstration*, pages 29–61. Springer, 1970.

[dC$^+$01] Luis Fariñas del Cerro et al. Lotrec : The Generic Tableau Prover for Modal and Description Logics. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First Int. Joint Conf., IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *LNCS*, pages 453–458. Springer, 2001.

[DH86] Vincent J. Digricoli and Malcolm C. Harrison. Equality-based binary resolution. *J. ACM*, 33(2):253–289, April 1986.

[DKS97] Jörg Denzinger, Martin Kronenburg, and Stephan Schulz. Discount - a distributed and learning equational prover. *Journal of Automated Reasoning*, 18(2):189–198, Apr 1997.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[Dru09] T. Drucker. *Perspectives on the History of Mathematical Logic*. Modern Birkhäuser Classics. Birkhäuser Boston, 2009.

[End15] Herbert B. Enderton. Second-order and higher-order logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2015 edition, 2015.

[Far08] William M. Farmer. The seven virtues of simple type theory. *J. Applied Logic*, 6(3):267–286, 2008.

[FB16] Michael Färber and Chad Brown. Internal guidance for satallax. In *International Joint Conference on Automated Reasoning*, pages 349–361. Springer, 2016.

[FB17] David Fuenmayor and Christoph Benzmüller. Types, Tableaus and Gödel's God in Isabelle/HOL. *Archive of Formal Proofs*, 2017. This publication is machine verified with Isabelle/HOL, but only mildly human reviewed.

[Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996.

[FM98] Melvin Fitting and R.L. Mendelsohn. *First-Order Modal Logic*. Synthese Library Studies in Epistemology Logic, Methodology, and Philosophy of Science Volume 277. Springer Netherlands, 1998.

[FR98] Michael J Fischer and Michael O Rabin. Super-exponential complexity of presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 122–135. Springer, 1998.

[Fre79] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Verlag von Louis Nebert, Halle, 1879.

[Fre93] Gottlob Frege. *Grundgesetze der Arithmetik*, volume 1. H. Pohle, 1893.

[FZ07] Branden Fitelson and EdwardN. Zalta. Steps toward a computational metaphysics. *Journal of Philosophical Logic*, 36(2):227–247, 2007.

[Gac08] Andrew Gacek. The Abella interactive theorem prover (system description). In *Automated Reasoning, IJCAR*, volume 5195 of *LNCS*, pages 154–161. Springer, 2008.

[Gar16] James Garson. Modal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2016 edition, 2016.

[Gil60] Paul C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4(1):28–35, 1960.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[GKU17] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun,* *Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 125–143. EasyChair, 2017.

[GM93] Mike Gordon and Tom Melham, editors. *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.

[GMR$^+$15] Jürgen Giesl, Frédéric Mesnard, Albert Rubio, René Thiemann, and Johannes Waldmann. Termination competition (termcomp 2015). In *International Conference on Automated Deduction*, pages 105–108. Springer, 2015.

[GMW79] Michael J. C. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[Göd69] Kurt Gödel. An interpretation of the intuitionistic sentential logic. In J. Hintakka, editor, *The Philosophy of Mathematics*, pages 128 – 129. Oxford University Press, 1969.

[Göd70] Kurt Gödel. Appx. a: Notes in kurt gödel's hand. pages 144 – 145. 1970. In J.H. Sobel. Logic and Theism: Arguments for and Against Beliefs in God. Cambridge U. Press, 2004.

[Gol81] Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[Gol11] R. Goldblatt. *Quantifiers, Propositions and Identity: Admissible Semantics for Quantified Modal and Substructural Logics*. Lecture Notes in Logic. Cambridge University Press, 2011.

[GP94] M. J. C. Gordon and A. M. Pitts. The HOL logic and system. In J. Bowen, editor, *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*, chapter 3, pages 49–70. Elsevier Science B.V., 1994.

[Gre81] Cordell Green. Application of theorem proving to problem solving. In *Readings in Artificial Intelligence*, pages 202–222. Elsevier, 1981.

[GSB17] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. Theorem provers for every normal modal logic. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 14–30, Maun, Botswana, 2017. EasyChair.

[Gup92] Aarti Gupta. Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1(2-3):151–238, 1992.

[HAB$^+$17] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

[Häh01] Reiner Hähnle. Tableaux and related methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 100–178. Elsevier and MIT Press, 2001.

[Har09] John Harrison. Hol light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, pages 60–66, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Hen50] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, 06 1950.

[Hin55] Jaakko Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica*, 8(7):55, 1955.

[HS00] Ulrich Hustadt and Renate A Schmidt. Mspass: Modal reasoning by translation and first-order resolution. In *TABLEAUX*, volume 1847, pages 67–71. Springer, 2000.

[Hue72] Gerard Pierre Huet. *Constrained Resolution: A Complete Method for Higher-order Logic.* PhD thesis, Case Western Reserve University, Cleveland, OH, USA, 1972. AAI7306307.

[Hue73a] Gerard P. Huet. A mechanization of type theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 139–146, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[Hue73b] Gerard P. Huet. The undecidability of unification in third order logic. *Information and control*, 22(3):257–267, 1973.

[Hue75] Gerard P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[Hue81] Gérard P. Huet. A complete proof of correctness of the knuth-bendix completion algorithm. *J. Comput. Syst. Sci.*, 23(1):11–21, 1981.

[Hur03] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68, 2003.

[HV11] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In *International Conference on Automated Deduction*, pages 299–314. Springer, 2011.

[JK11] Swen Jacobs and Viktor Kuncak. Towards complete reasoning about axiomatic specifications. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 278–293. Springer, 2011.

[JM05] Mauro Jaskelioff and Stephan Merz. Proving the correctness of disk paxos. *Archive of Formal Proofs*, June 2005. `http://isa-afp.org/entries/DiskPaxos.html`, Formal proof development.

[Joh85] Thomas Johnsson. Lambda lifting: Treansforming programs to recursive equations. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings*, volume 201 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 1985.

[JP09] Paul B Jackson and Grant Olney Passmore. Proving spark verification conditions with smt solvers, 2009.

[JU18] Jan Jakubuv and Josef Urban. Hierarchical invention of theorem proving strategies. *AI Commun.*, 31(3):237–250, 2018.

[KB70] Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. In *Computational problems in abstract algebra*, pages 263–297. Elsevier, 1970.

[Ker94] Manfred Kerber. On the translation of higher-order problems into first-order logic. In *ECAI*, pages 145–149, 1994.

[KKKS13] Etienne Kneuss, Ivan Kuraj, Viktor Kuncak, and Philippe Suter. Synthesis modulo recursive functions. *Acm Sigplan Notices*, 48(10):407–426, 2013.

[KMPS10] Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, and Philippe Suter. Complete functional synthesis. *ACM Sigplan Notices*, 45(6):316–329, 2010.

[KNP03] Gerwin Klein, Tobias Nipkow, and Lawrence Paulson. The archive of formal proofs, 2003.

[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 *International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 294–309. Springer, 1995.

[Kop12] Cynthia Kop. *Higher order termination*. PhD thesis, Faculty of Sciences, Department of Computer Science, VUA, 2012.

[Kor08] Konstantin Korovin. iprover–an instantiation-based theorem prover for first-order logic (system description). In *International Joint Conference on Automated Reasoning*, pages 292–298. Springer, 2008.

[Kro09] Daniel Kroening. Software verification. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 505–532. IOS Press, 2009.

[KRTU99] A. J. Kfoury, Simona Ronchi Della Rocca, Jerzy Tiuryn, and Pawel Urzyczyn. Alpha-conversion and typability. *Inf. Comput.*, 150(1):1–21, 1999.

[KSR16] Cezary Kaliszyk, Geoff Sutcliffe, and Florian Rabe. TH1: the TPTP typed higher-order form with rank-1 polymorphism. In P. Fontaine, S. Schulz, and J. Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*, volume 1635 of *CEUR Workshop Proceedings*, pages 41–55. CEUR-WS.org, 2016.

[KU15a] Cezary Kaliszyk and Josef Urban. Femalecop: Fairly efficient machine learning connection prover. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 88–96. Springer, 2015.

[KU15b] Cezary Kaliszyk and Josef Urban. Hol(y)hammer: Online ATP service for HOL light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer Berlin Heidelberg, 2013.

[Lag11] Jeffrey C. Lagarias. *The Kepler Conjecture: The Hales-Ferguson Proof*. Springer New York, 2011.

[Lei89] Gottfried Wilhelm Leibniz. Discourse on metaphysics. In Leroy E. Loemker, editor, *Philosophical Papers and Letters*, pages 303–330. Springer Netherlands, Dordrecht, 1989.

[Lib15] Tomer Libal. Regular patterns in second-order unification. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2015.

[Lin08] Fredrik Lindblad. Higher-order proof construction based on first-order narrowing. *Electr. Notes Theor. Comput. Sci.*, 196:69–84, 2008.

[LISK17] Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *CoRR*, abs/1701.06972, 2017.

[LS16] Tomer Libal and Alexander Steen. Towards a substitution tree based index for higher-order resolution theorem provers. In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *5th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, number 1635 in CEUR Workshop Proceedings, pages 82–94, Aachen, 2016.

[Mac95] D. MacKenzie. The automation of proof: A historical and sociological exploration. *IEEE Ann. Hist. Comput.*, 17(3):7–29, September 1995.

[McC94] William W. McCune. Otter 3.0 reference manual and guide. Technical report, Argonne National Laboratory, Argonne, IL, 1994.

[McC97a] William McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. *Automated reasoning and its applications: Essays in honor of Larry Wos*, pages 71–114, 1997.

[McC97b] William McCune. Solution of the robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[MD00] Fabio Massacci and Francesco M. Donini. Design and results of tancs-2000 non-classical (modal) systems comparison. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 52–56, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Mel93] T. F. Melham. The HOL logic extended with quantification over type variables. *Formal Methods in System Design*, 3(1-2):7–24, 1993.

[Men16] Christopher Menzel. Actualism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[Mil83] Dale A Miller. *Proofs in higher-order logic*. PhD thesis, Carnegie-Mellon University, 1983.

[Mil91a] Dale Miller. Unification of simply typed lamda-terms as logic programming. In Koichi Furukawa, editor, *Logic Programming, Proceedings of the Eigth International Conference, Paris, France, June 24-28, 1991*, pages 255–269. MIT Press, 1991.

[Mil91b] Dale A. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[MM76] Alberto Martelli and Ugo Montanari. *Unification in linear time and space: A structured presentation*. Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, 1976.

[MP08] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008.

[MP09] Jia Meng and Lawrence C Paulson. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

[Mus07] Reinhard Muskens. Intensional models for the theory of types. *J. Symb. Log.*, 72(1):98–118, 2007.

[MW80] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, January 1980.

[MW97] William McCune and Larry Wos. Otter-the cade-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

[Nad99] Gopalan Nadathur. A fine-grained notation for lambda terms and its use in intensional operations. *Journal of Functional and Logic Programming*, 1999(2), 1999.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Logic in Computer Science, 1991. LICS'91., Proceedings of Sixth Annual IEEE Symposium on*, pages 342–349. IEEE, 1991.

[Nip93] Tobias Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science*, pages 64–74, 1993.

[NM99] Gopalan Nadathur and DustinJ. Mitchell. System Description: Teyjus - A Compiler and Abstract Machine Based Implementation of λProlog. In *Automated Deduction, CADE*, volume 1632 of *LNAI*, pages 287–291. Springer, 1999.

[NR92] Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering constrained clauses. In Deepak Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*, volume 607 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 1992.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 371–443. Elsevier and MIT Press, 2001.

[NS56] Allen Newell and Herbert Simon. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79, 1956.

[NW01] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 335–367. Elsevier and MIT Press, 2001.

[NWP02] Tobias Nipkow, Makarius Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[OB17] Jens Otten and Wolfgang Bibel. Advances in connection-based automated theorem proving. In Michael G. Hinchey, Jonathan P. Bowen, and Ernst-Rüdiger Olderog, editors, *Provably Correct Systems*, NASA Monographs in Systems and Software Engineering, pages 211–241. Springer, 2017.

[ORR$^+$96] Sam Owre, S. Rajan, John M. Rushby, Natarajan Shankar, and Mandayam K. Srivas. PVS: combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer, 1996.

[Ott14] Jens Otten. Mleancop: A connection prover for first-order modal logic. In *International Joint Conference on Automated Reasoning*, pages 269–276. Springer, 2014.

[Pau88] Lawrence C. Paulson. A formulation of the simple theory of types (for isabelle). In Per Martin-Löf and Grigori Mints, editors, *COLOG-88, International Conference on Computer Logic, Tallinn,* *USSR, December 1988, Proceedings*, volume 417 of *Lecture Notes in Computer Science*, pages 246–274. Springer, 1988.

[Pau11] C. Paulin-Mohring. Introduction to the coq proof-assistant for practical software verification. In *Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pages 45–95, 2011.

[PB10] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In *PAAR@ IJCAR*, pages 1–10, 2010.

[PD10] Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Automated Reasoning, IJCAR*, volume 6173 of *LNCS*, pages 15–21. Springer, 2010.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[Pie09] Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Logic*, 11(1):6:1–6:40, November 2009.

[PJ72] T. Pietrzykowski and D. C. Jensen. A complete mechanization of (Ω)-order type theory. In *Proceedings of the ACM Annual Conference - Volume 1*, ACM '72, pages 82–92, New York, NY, USA, 1972. ACM.

[PJ87] Simon L Peyton Jones. *The implementation of functional programming languages (prentice-hall international series in computer science)*. Prentice-Hall, Inc., 1987.

[PP03] Brigitte Pientka and Frank Pfenning. Optimizing higher-order pattern unification. In *19th International Conference on Automated Deduction*, pages 473–487. Springer-Verlag, 2003.

[Pra60] Dag Prawitz. An improved proof procedure. *Theoria*, 26(2):102–139, 1960.

[Pre29] M. Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In *Automated Deduction, CADE*, volume 1632 of *LNAI*, pages 202–206. Springer, 1999.

[PU18] Bartosz Piotrowski and Josef Urban. Atpboost: Learning premise selection in binary setting with ATP feedback. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018.

[PW12] Álvaro Pelayo and Michael A. Warren. Homotopy type theory and voevodsky's univalent foundations. *CoRR*, abs/1210.5658, 2012.

[Qia96] Zhenyu Qian. Unification of higher-order patterns in linear time and space. *Journal of Logic and Computation*, 6(3):315–341, 1996.

[RDK$^+$15] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark Barrett. Counterexample-guided quantifier instantiation for synthesis in smt. In *International Conference on Computer Aided Verification*, pages 198–216. Springer, 2015.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Symposium on Programming*, pages 408–423, 1974.

[Rey98a] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.

[Rey98b] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998. Cambridge Books Online.

[Rin09] Jussi Rintanen. Planning and SAT. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 483–504. IOS Press, 2009.

[RO12] Thomas Raths and Jens Otten. The QMLTP Problem Library for First-Order Modal Logics. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 454–461. Springer, 2012.

[Rob65] John A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.

[Rob69] John A. Robinson. Mechanizing higher-order logic. *Machine Intelligence*, 4(150-170):24, 1969.

[Rus03] Bertrand Russell. *The Principles of Mathematics*. Number v. 1 in The Principles of Mathematics. University Press, 1903.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30(3):222–262, 1908.

[Rus96] Bertrand Russell. *The principles of mathematics*. WW Norton & Company, 1996.

[RV01] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI communications*, 15(2, 3):91–110, 2002.

[RV03] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1):101 – 115, 2003. First Order Theorem Proving.

[RW69] George Robinson and Larry Wos. Paramodulation and theorem-proving in first-order theories with equality. *Machine intelligence*, 4:135–150, 1969.

[Sah75] Henrik Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. *Studies in Logic and the Foundations of Mathematics*, 82:110–143, 1975.

[SB10] Geoff Sutcliffe and Christoph Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. *Journal of Formalized Reasoning*, 3(1):1–27, 2010.

[SB15] Alexander Steen and Christoph Benzmüller. There Is No Best Beta-Normalization Strategy for Higher-Order Reasoners. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 9450 of *LNAI*, pages 329–339, Suva, Fiji, 2015. Springer.

[SB16] Alexander Steen and Christoph Benzmüller. Sweet SIXTEEN: Automation via embedding into classical higher-order logic. *Logic and Logical Philosophy*, 25:535–554, 2016.

[SB18] Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning — 9th International Joint Conference, IJCAR 2018, Oxford, UK, July 14-17, 2018, Proceedings*, LNCS. Springer, 2018. To Appear.

[SBA06] Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. Computer supported mathematics with OMEGA. *Journal of Applied Logic*, 4(4):533–559, 2006.

[SBP13] Nik Sultana, Jasmin Christian Blanchette, and Lawrence C Paulson. Leo-ii and satallax on the sledgehammer test bench. *Journal of Applied Logic*, 11(1):91–102, 2013.

[Sch24] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Mathematische annalen*, 92(3-4):305–316, 1924.

[Sch02] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Communications*, 15(2,3):111–126, August 2002.

[Sch13] Stephan Schulz. Simple and efficient clause subsumption with feature vector indexing. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 45–67. Springer, 2013.

[Sch15] Lenhart Schubert. Computational linguistics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2015 edition, 2015.

[Sch17] Stephan Schulz. We know (nearly) nothing! But can we learn? In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements*, volume 51 of *EPiC Series in Computing*, pages 29–32. EasyChair, 2017.

[Sco72] Dana Scott. Appx. b: Notes in dana scott's hand. pages 145 – 146. 1972. In J.H. Sobel. Logic and Theism: Arguments for and Against Beliefs in God. Cambridge U. Press, 2004.

[SG89] Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.

[SM16] Stephan Schulz and Martin Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2016.

[Smu63] Raymond M Smullyan. A unifying principal in quantification theory. *Proceedings of the National Academy of Sciences*, 49(6):828–832, 1963.

[Smu95] Raymond M Smullyan. *First-order logic*. Courier Corporation, 1995.

[Sny91] Wayne Snyder. *Higher Order Unification*, pages 123–153. Birkhäuser Boston, Boston, MA, 1991.

[Sob87] Jordan Howard Sobel. Gödel's ontological proof. *On Being and Saying. Essays for Richard Cartwright*, pages 241–261, 1987.

[SRV01] R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. Term indexing. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier, Amsterdam, The Netherlands, 2001.

[SS15] Simon Schäfer and Stephan Schulz. Breeding theorem proving heuristics with genetic algorithms. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19,* *2015*, volume 36 of *EPiC Series in Computing*, pages 263–274. EasyChair, 2015.

[SSCB12] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The tptp typed first-order form with arithmetic. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 406–419. Springer, 2012.

[SST14] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 367–373. Springer, 2014.

[SSUP17] Stephan Schulz, Geoff Sutcliffe, Josef Urban, and Adam Pease. Detecting inconsistencies in large first-order knowledge bases. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 310–325. Springer, 2017.

[Sta77] Richard Statman. The typed λ-calculus is not elementary recursive. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 90–94. IEEE, 1977.

[Sta12] Robert Stalnaker. *Mere Possibilities: Metaphysical Foundations of Modal Semantics*. Carl G. Hempel Lecture Series. Princeton University Press, 2012.

[Ste14] Alexander Steen. Efficient Data Structures for Automated Theorem Proving in Expressive Higher-Order Logics. Master's thesis, Freie Universität Berlin, Berlin, Germany, 2014.

[Sti09] Colin Stirling. Decidability of higher-order matching. *arXiv preprint arXiv:0907.3804*, 2009.

[Sut06] Geoff Sutcliffe. Semantic derivation verification: Techniques and implementation. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[Sut07] Geoff Sutcliffe. TPTP, TSTP, CASC, etc. In V. Diekert, M. Volkov, and A. Voronkov, editors, *Proceedings of the 2nd International Computer Science Symposium in Russia*, number 4649 in Lecture Notes in Computer Science, pages 7–23. Springer-Verlag, 2007.

[Sut08] Geoff Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In *LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics (Doha, Qattar)*, volume 418, pages 38–49. CEUR Workshop Proceedings, 2008.

[Sut09] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[Sut10] Geoff Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In E. Clarke and A. Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.

[Sut16a] Geoff Sutcliffe. The 8th IJCAR automated theorem proving system competition - CASC-J8. *AI Commun.*, 29(5):607–619, 2016.

[Sut16b] Geoff Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[Sut17a] Geoff Sutcliffe. The CADE-26 automated theorem proving system competition - CASC-26. *AI Commun.*, 30(6):419–432, 2017.

[Sut17b] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[SWB16] Alexander Steen, Max Wisniewski, and Christoph Benzmüller. Agent-based HOL reasoning. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016, 5th International Congress, Proceedings*, volume 9725 of *LNCS*, pages 75–81, Berlin, Germany, 2016. Springer.

[SWSB17] Alexander Steen, Max Wisniewski, Hans-Jörg Schurr, and Christoph Benzmüller. Capability discovery for automated reasoning systems. In Thomas Eiter, David Sands, Geoff Sutcliffe, and Andrei Voronkov, editors, *IWIL@LPAR 2017 Workshop and LPAR-21 Short Presentations, Maun, Botswana, May 7-12, 2017*, volume 1 of *Kalpa Publications in Computing*, Maun, Botswana, 2017. EasyChair.

[Tam96] Tanel Tammet. A resolution theorem prover for intuitonistic logic. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings*, volume 1104 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 1996.

[TB06] Frank Theiss and Christoph Benzmüller. Term indexing for the LEO-II prover. In *IWIL-6 workshop at LPAR 2006: The 6th International Workshop on the Implementation of Logics*, Pnom Penh, Cambodia, 2006.

[TCM02] Virginie Thion, Serenella Cerrito, and Marta Mayer. A general theorem prover for quantified modal logics. *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 193–258, 2002.

[Tho97] Simon Thompson. Higher-order + Polymorphic = Reusable. http://www.cs.kent.ac.uk/pubs/1997/224, May 1997.

[TO98] Andrew P. Tolmach and Dino Oliva. From ML to ada: Strongly-typed language interoperability via source translation. *J. Funct. Program.*, 8(4):367–412, 1998.

[TSK12] Dmitry Tishkovsky, Renate A Schmidt, and Mohammad Khodadadi. The tableau prover generator MetTeL2. In *European Workshop on Logics in Artificial Intelligence*, pages 492–495. Springer, 2012.

[Tur79] David A Turner. Another algorithm for bracket abstraction. *The Journal of Symbolic Logic*, 44(2):267–270, 1979.

[USPV08] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. Malarea sg1-machine learner for automated reasoning with semantic guidance. In *International Joint Conference on Automated Reasoning*, pages 441–456. Springer, 2008.

[VBD83] Johan Van Benthem and Kees Doets. Higher-order logic. In *Handbook of philosophical logic*, pages 275–329. Springer, 1983.

[VH67] Jean Van Heijenoort. *From Frege to Gödel: A Source Book in Mathematics 1879-1931.* Harvard University Press, Cambridge, MA, 1967.

[VN25] John Von Neumann. Eine axiomatisierung der mengenlehre. *Journal für die reine und angewandte Mathematik*, 154:219–240, 1925.

[Völ07] Norbert Völker. HOL2P – a system of classical higher order logic with second order polymorphism. In *Theorem Proving in Higher Order Logics*, pages 334–351. Springer, 2007.

[Wan83] Hao Wang. Towards mechanical mathematics. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers on Computational Logic*, pages 244–264. Springer, 1983.

[Wan14] Daniel Wand. Polymorphic+typeclass superposition. In Stephan Schulz, Leonardo de Moura, and Boris Konev, editors, *4th Workshop on Practical Aspects of Automated Reasoning, PAAR@IJCAR*

2014, Vienna, Austria, 2014, volume 31 of *EPiC Series in Computing*, pages 105–119. EasyChair, 2014.

[WB16] Max Wisniewski and Christoph Benzmüller. Is it reasonable to employ agents in theorem proving? In Jan van den Heerik and Joaquim Filipe, editors, *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) – Volume 1, pages 281-286*, volume 1, pages 281–286, Rome, Italy, 2016. SCITEPRESS – Science and Technology Publications, Lda.

[WCR64] Lawrence Wos, Daniel Carson, and George Robinson. The unit preference strategy in theorem proving. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pages 615–621. ACM, 1964.

[WDF$^+$09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In *International Conference on Automated Deduction*, pages 140–145. Springer, 2009.

[Wei99] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Intelligent Robotics and Autonomous Agents Series. CogNet, 1999.

[Wei17] Christoph Weidenbach. Do portfolio solvers harm? In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements*, volume 51 of *EPiC Series in Computing*, pages 76–81. EasyChair, 2017.

[Wie99] Tomasz Wierzbicki. Complexity of the higher order matching. In *International Conference on Automated Deduction*, pages 82–96. Springer, 1999.

[Wil13] Timothy Williamson. *Modal Logic as Metaphysics*. Oxford University Press, 2013.

[WM78] Mark N. Wegman and Paterson Mike. Linear unification. *J. Comput. Syst. Sci*, 16:158–167, 1978.

[Wol98] Andreas Wolf. P-setheo: Strategy parallelism in automated theorem proving. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 320–324. Springer, 1998.

[Wol09] David A Wolfram. *The clausal theory of types*, volume 21. Cambridge University Press, 2009.

[WRC65] Lawrence Wos, George A Robinson, and Daniel F Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM (JACM)*, 12(4):536–541, 1965.

[WSB15] Max Wisniewski, Alexander Steen, and Christoph Benzmüller. LEOPARD - A Generic Platform for the Implementation of Higher-Order Reasoners. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *LNCS*, pages 325–330. Springer, 2015.

[WSB16] Max Wisniewski, Alexander Steen, and Christoph Benzmüller. TPTP and beyond: Representation of quantified non-classical logics. In Christoph Benzmüller and Jens Otten, editors, *ARQNL 2016. Automated Reasoning in Quantified Non-Classical Logics*, volume 1770, pages 51–65. CEUR Workshop Proceedings, http://ceur-ws.org, 2016.

[WSKB16] Max Wisniewski, Alexander Steen, Kim Kern, and Christoph Benzmüller. Effective normalization techniques for HOL. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning — 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, volume 9706 of *LNCS*, pages 362–370. Springer, 2016.

[WTWD17] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2783–2793. Curran Associates, Inc., 2017.

[Zal17] Edward N. Zalta. Frege's theorem and foundations for arithmetic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.

[Zer08] Ernst Zermelo. Untersuchungen über die grundlagen der mengenlehre. i. *Mathematische Annalen*, 65(2):261–281, 1908.

[Zie18] Marco Ziener. Mixing automated theorem proving and machine learning. Master's thesis, Freie Universität Berlin, Berlin, Germany, 2018.