Extensional Higher-Order Paramodulation in Leo-III*

Alexander Steen, Christoph Benzmüller

alexander.steen@uni.lu, c.benzmueller@fu-berlin.de

Abstract

Leo-III is an automated theorem prover for extensional type theory with Henkin semantics and choice. Reasoning with primitive equality is enabled by adapting paramodulation-based proof search to higher-order logic. The prover may cooperate with multiple external specialist reasoning systems such as first-order provers and SMT solvers. Leo-III is compatible with the TPTP/TSTP framework for input formats, reporting results and proofs, and standardized communication between reasoning systems, enabling e.g. proof reconstruction from within proof assistants such as Isabelle/HOL.

Leo-III supports reasoning in polymorphic first-order and higher-order logic, in all normal quantified modal logics, as well as in different deontic logics. Its development had initiated the ongoing extension of the TPTP infrastructure to reasoning within non-classical logics.

1 Introduction

Leo-III is an automated theorem prover (ATP) for classical higher-order logic (HOL) with Henkin semantics and choice. In contrast to its predecessors, LEO and LEO-II [BK98, BSPT15], that were based on resolution proof search, Leo-III implements a higher-order paramodulation calculus which aims at improved performance for equational reasoning [Ste18]. In the tradition of the Leo prover family, Leo-III collaborates with external reasoning systems, in particular, with first-order ATP systems such as E [Sch02], iProver [Kor08] and Vampire [RV02] as well as SMT solvers, e.g., with CVC4 [B$^+$11]. Cooperation is not restricted to first-order systems, and further specialized systems such as higher-order (counter-)model finders may be utilized by Leo-III.

Leo-III accepts all common TPTP dialects [Sut17] as well as the recent extensions to polymorphic types [BP13b, KSR16]. 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. The prover returns results according to the standardized TPTP SZS ontology and additionally produces verifiable TPTP-compatible proof certificates, for each proof that it finds.

*This work has been supported by the DFG under grant BE 2501/11-1 (Leo-III) and by the VolkswagenStiftung under grant CRAP (Rational Argumentation in Politics).

The predecessor systems LEO and LEO-II pioneered the area of cooperative resolution-based theorem proving for Henkin semantics. LEO (or LEO-I) was designed as an ATP component of the proof assistant and proof planner ΩMEGA [SBA06] and hard-wired to it. Its successor, LEO-II, is a standalone HOL ATP system based on Resolution by Unification and Extensionality (RUE) [Ben99b], and it supports reasoning with primitive equality.

The most recent incarnation of the Leo prover family, Leo-III, comes with improved reasoning performance in particular for equational problems, and with a more flexible and effective architecture for cooperation with external specialists. Reasoning in higher-order quantified non-classical logics, including all normal modal logics, and different versions of deontic logic is enabled by an integrated shallow semantical embedding approach [BP13a]. In contrast to other HOL ATP systems, including LEO-II, for which it was necessary for the user to manually conduct the tedious and error-prone encoding procedure before passing it to the prover, Leo-III is the first ATP system to include a rich library of these embeddings, transparent to the user [GSB17]. These broad logic competencies make Leo-III, up to the authors' knowledge, the most widely applicable ATP system for propositional and quantified, classical and non-classical logics available to date. This work has also stimulated the currently ongoing extension of the TPTP library to non-classical reasoning.[1]

Leo-III is implemented in Scala, its source code, and that of related projects presented in this article, is publicly available under BSD-3 license on GitHub.[2] Installing Leo-III does not require any special libraries, apart from reasonably current version of the JDK and Scala. Also, Leo-III is readily available via the SystemOnTPTP web interface [Sut17] and can be called, via Sledgehammer [BBP13], from the interactive proof assistant Isabelle/HOL [NPW02] for automatically discharging user's proof goals.

In a recent evaluation study of 19 different first-order and higher-order ATP systems, Leo-III was found the most versatile (in terms of supported logic formalisms) and best performing ATP system overall [BGK+19].

This article presents a consolidated summary of previous conference and workshop contributions [BSW17, SB18, SWB16, SWB17, WSB15] as well as contributions from the first author's PhD thesis [Ste18]. It is structured as follows: §2 briefly introduces HOL and challenging automation aspects. In §3 the paramodulation calculus underlying Leo-III is sketched, and practically motivated extensions that are implemented in the prover are outlined. Subsequently, the implementation of Leo-III is described in more detail in §4, and §5 presents the technology that enables Leo-III to reason in various non-classical logics. An evaluation of Leo-III on a heterogeneous set of benchmarks, including problems in non-classical logics, is presented in §6. Finally, §7 concludes this article and sketches further work.

2 Higher-Order Theorem Proving

The term higher-order logic refers to expressive logical formalisms that allow for quantification over predicate and function variables; such a logic was first studied

[1] See `http://tptp.org/TPTP/Proposals/LogicSpecification.html`.

[2] See the individual projects related to the Leo prover family at `https://github.com/leoprover`. Further information are available at `http://inf.fu-berlin.de/~lex/leo3`.

by Frege in the 1870s [Fre79]. An alternative and more handy formulation was proposed by Church in the 1940s [Chu40]. He defined a higher-order logic on top of the simply typed λ-calculus. His particular formalism, referred to as Simple Type Theory (STT), was later further studied and refined by Henkin [Hen50], Andrews [And71, And72b, And72a] and others [BBK04, Mus07]. In the remainder, the term HOL is used synonymously to Henkin's Extensional Type Theory (ExTT) [BM14]; it constitutes the foundation of many contemporary higher-order automated reasoning systems. HOL, being a subsystem of SST, provides lambda-notation as an elegant and useful means to denote unnamed functions, predicates and sets (by their characteristic functions), and comes with built-in principles of Boolean and functional extensionality as well as type-restricted comprehension.

A more in-depth presentation of HOL, its historical development, metatheory and automation is provided by Benzmller and Miller [BM14].

Syntax and Semantics. HOL is a typed logic; every term of HOL is associated a fixed and unique type, written as subscript. The set $\mathcal{T}$ of simple types is freely generated from a non-empty set S of sort symbols (base types) and juxtaposition $\nu\tau$ of two types $\tau, \nu \in \mathcal{T}$, the latter denoting the type of functions from objects of type τ to objects of type ν. Function types are assumed to associate to the left and parentheses may be dropped if consistent with the intended reading. The base types are usually chosen to be $S := \{\iota, o\}$, where ι and o represent the type of individuals and the type of Boolean truth values, respectively.

Let Σ be a typed signature and let $\mathcal{V}$ denote a set of typed variable symbols such that there exist infinitely many variables for each type. Following Andrews [And02], it is assumed that the only primitive logical connectives are given by equality, denoted $=^{\tau}_{o\tau\tau} \in \Sigma$, for each type $\tau \in \mathcal{T}$ (called $\mathfrak{q}$ by Andrews). In the extensional setting of HOL, all remaining logical connectives such as disjunction $\vee_{ooo}$, conjunction $\wedge_{ooo}$, negation $\neg_{oo}$, etc., can be defined in terms of them. The terms of HOL are given by the following abstract syntax (where $\tau, \nu \in \mathcal{T}$ are types):

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

The terms are called constants, variables, abstractions and applications, respectively. The type of a term may be dropped for legibility reasons if obvious from the context. Application is assumed to associate to the left and parentheses may again be dropped whenever possible. Notions such as α-, β-, and η-conversion, denoted $\longrightarrow_\star$, for $\star \in \{\alpha, \beta, \eta\}$, free variables fv(.) of a term, etc., are defined as usual [BDS13]. Syntactical equality between HOL terms, denoted $\equiv_\star$, for $\star \in \{\beta, \eta, \beta\eta\}$, is defined with respect to the assumed underlying conversion rules (α-conversion is always assumed implicitly). Terms s_o of type o are formulas, and they are sentences if they are closed. By convention, infix notation for fully applied logical connectives is used, e.g. $s_o \vee t_o$ instead of $(\vee_{ooo}\ s_o)\ t_o$.

As a consequence of Gödel's Incompleteness Theorem, HOL with standard semantics is necessarily incomplete. In contrast, theorem proving in HOL is usually considered with respect to so-called general semantics (or Henkin semantics) in which a meaningful notion of completeness can be achieved [Hen50]. The usual notions of general model structures, validity in these structures and

related notions are assumed in the following. Intensional models have been described by Muskens [Mus07] and studies of further general notions of semantics are presented by Andrews [And71] and Benzmüller et al. [BBK04].

Challenges to HOL Automation. HOL validates functional and Boolean extensionality principles, referred to as $\mathrm{EXT}^{\nu\tau}$ respective EXT^{o} below that can be formulated within its term language as

$$\mathrm{EXT}^{\nu\tau} := \forall F_{\nu\tau}.\, \forall G_{\nu\tau}.\, (\forall X_{\tau}.\, F\ X =^{\nu} G\ X) \Rightarrow F =^{\nu\tau} G$$
$$\mathrm{EXT}^{o} := \forall P_{o}.\, \forall Q_{o}.\, (P \Leftrightarrow Q) \Rightarrow P =^{o} Q$$

These principles state that two functions are equal if they correspond on every argument, and that two formulas are equal if they are equivalent (where $\Leftrightarrow_{ooo}$ denotes equivalence), respectively. 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 (where $\top_o$ denotes syntactical truth), and that $(\lambda P_o.\, \lambda Q_o.\, P \vee Q) = (\lambda P_o.\, \lambda Q_o.\, Q \vee P)$ is a theorem. Boolean Extensionality, in particular, poses a considerable challenge for HOL automation: Two terms may be equal, and thus subject to generating inferences, if the equivalence of all Boolean typed subterms can be inferred. As a consequence, the implementation of non-ground proof calculi that make use of higher-order unification procedures cannot use syntactical unification for locally deciding which inferences are to be generated. In contrast to first-order theorem proving, it hence seems useful to interleave syntactical unification and semantical proof search, which is more difficult to control in practice.

As a further complication, higher-order unification is only semi-decidable and not unitary [Hue73, Gol81]. It is not clear, how many and which unifiers produced by a higher-order unification routine should be chosen during proof search, and the unification procedure may never terminate on non-unifiable terms.

In the context of first-order logic with equality, superposition based calculi have proven an effective basis for reasoning systems and provide a powerful notion of redundancy [BG90, NR92, BG94]. Reasoning with equality can also be addressed, e.g., by an RUE resolution approach [DH86] and, in the higher-order case, by reducing equality to equivalent formulas not containing the equality predicate [Ben99b], as done in LEO. The latter approaches however lack effectivity in practical applications of large scale equality reasoning.

Also, from an implementation point of view, there are only few approaches available to highly efficient data structures and indexing techniques for implementing HOL ATP system. This additionally hampers the practical effectivity of HOL reasoning systems and their application.

HOL ATP Systems. Next to the LEO prover family [BK98, BSPT15, SB18], there are further HOL ATP systems available: This includes TPS [AB06] as one of the earliest systems, as well as Satallax [Bro12], cocATP [BC04], agsyHOL [Lin14] and the higher-order (counter)model finder Nitpick [BN10]. Additionally, there is ongoing work on extending the first-order theorem prover Vampire to full higher-order reasoning [BR18, BR19], and some interactive proof assistants such as Isabelle/HOL [NPW02] can also be used for automated reasoning in HOL. Further related systems include higher-order extensions of SMT solvers [BREO+19], and there is ongoing work to lift first-order ATP systems

based on superposition to fragments of HOL, including E [Sch02, VBCS19] and Zipperposition [Cru15, BBCW18].

Applications. 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 [Ben17], many-valued logics [SB16], deontic logic [BFP18], free logics [BS16], and combinations of such logics [Ben11] can be used to automate reasoning within those logics 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 [BW16] using LEO-II and Isabelle/HOL. Similar and further enhanced techniques were used to assess foundational questions in metaphysics [BWW17, KBZ19].

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 higher-order proof assistant HOL Light [Har09] played a key role in the verification of Kepler's conjecture within the Flyspeck project [H+15].

3 Extensional Higher-Order Paramodulation

Leo-III is a refutational ATP system. The initial, possibly empty, set of axioms and the negated conjecture are transformed into an equisatisfiable set of formulas in clause normal form (CNF), which is then iteratively saturated until the empty clause is found. Leo-III extends the complete, paramodulation based calculus EP for HOL (cf. further below) with practically motivated, partly heuristic inference rules. Paramodulation extends resolution by a native treatment of equality at the calculus level. In the context of first-order logic, it was developed in the late 1960s by G. Robinson and L. Wos [RW69] 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 speculative conditional rewriting step. In the context of first-order theorem proving, superposition based calculi [BG90, NR92, BG94] improve the naive paramodulation approach by imposing ordering restrictions on the inference rules such that only a relevant subset of all possible inferences are generated. However, due to the more complex structure of the term language of HOL, there do not exist suitable term orderings that allow a straightforward adaption of this approach to the higher-order setting.[3]

Higher-order paramodulation for extensional type theory was first presented by Benzmüller [Ben99a, Ben99b]. This calculus was mainly theoretically motivated and extended a resolution calculus with a paramodulation rule instead

[3] As a simple counterexample, consider a (strict) term ordering $\succ$ for HOL terms that satisfies the usual properties from first-order superposition (e.g., the subterm property) and is compatible with β-reduction. For any nonempty signature Σ, $\mathrm{c} \in \Sigma$, the chain $\mathrm{c} \equiv_\beta (\lambda X.\, \mathrm{c})\ \mathrm{c} \succ \mathrm{c}$ can be constructed, implying $\mathrm{c} \succ \mathrm{c}$ and thus contradicting irreflexivity of $\succ$. Note that $(\lambda X.\, \mathrm{c})\ \mathrm{c} \succ \mathrm{c}$ since the right-hand side is a proper subterm of the left-hand side (assuming an adequately lifted definition of subterm property to HO terms).

of being based on a paramodulation rule alone. Additionally, that calculus contained a rule that expanded equality literals by their definition due to Leibniz.[4] As Leibniz equality formulas effectively enable cut-simulation [BBK09], the proposed calculus seems unsuited for automation. The calculus EP presented in the following, in contrast, avoids the expansion of equality predicates but adapts the use of dedicated calculus rules for extensionality principles from Benzmller [Ben99b].

An equation, denoted $s \simeq t$, is a pair of HOL terms of the same type, where $\simeq$ is assumed to be symmetric. A literal ℓ is a signed equation, written $[s \simeq t]^\alpha$, where $\alpha \in \{\mathfrak{t}, \mathfrak{f}\}$ is the polarity of ℓ. Literals of form $[s_o]^\alpha$ are shorthand for $[s_o \simeq \top]^\alpha$ and negative literals $[s \simeq t]^\mathfrak{f}$ are also referred to as unification constraints. A negative literal ℓ is called a flex-flex unification constraint if ℓ is of the form $\ell \equiv [X\ \overline{s^i} \simeq Y\ \overline{t^j}]^\mathfrak{f}$, where X, Y are variables. A clause $\mathcal{C}$ is a multiset of literals, denoting its disjunction. For brevity, if $\mathcal{C}, \mathcal{D}$ are clauses and ℓ is a literal, $\mathcal{C} \vee \ell$ and $\mathcal{C} \vee \mathcal{D}$ denote the multi-union $\mathcal{C} \cup \{\ell\}$ and $\mathcal{C} \cup \mathcal{D}$, respectively. $s|_\pi$ is the subterm of s at position π, and $s[r]_\pi$ denotes the term that is created by replacing the subterm of s at position π by r.

The EP calculus can be divided into four groups of inference rules:

Clause normalization. The clausification rules of EP are standard [Ste18, §3.2]. Every non-normal clause is transformed into an equisatisfiable set of clauses in CNF. Note that the clausification rules are proper inference rules rather than a dedicated meta operation. This is due to the fact that non-CNF clauses may be generated from the application of the remaining inferences rules, hence renormalization may be necessary. In the following we use CNF to refer to the entirety of the CNF rules.

For the elimination of existential quantifiers, the sound Skolemization technique of Miller [Mil83, Mil91] is assumed.

Primary inferences. The primary inference rules of Leo-III are paramodulation (Para), equality factoring (EqFac) and primitive substitution (PS), cf. Fig. 1. The first two rules introduce unification constraints that are encoded as negative equality literals: A generating inference is semantically justified if the unification constraint(s) can be solved. Since higher-order unification is not decidable, these constraints are explicitly encoded into the result clause for subsequent analysis. Note that both (Para) and (EqFac) are unordered and produce numerous redundant clauses. In practice, Leo-III tries to remedy this situation by using heuristics to restrict the number of generated clauses, including a higher-order term ordering, cf. §4.

Rule (PS) instantiates free predicate variables at top level with approximations of formulas using general bindings $\mathcal{GB}^t_\tau$ [BSPT15, §2]. This is

[4] 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 equality predicate. 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. [BBK04] to which the authors refer to for further details.

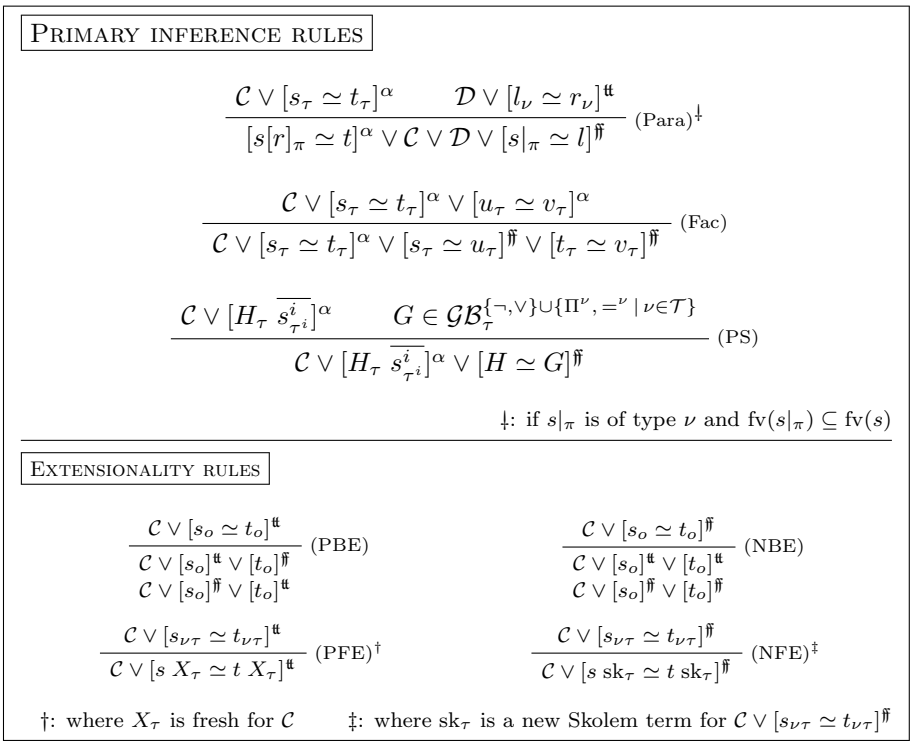

Figure 1: Primary inference rules and extensionality rules of EP.

necessary to ensure that variable heads may be instantiated by arbitrary formulas during proof search.

Extensionality rules. The rules (NBE) and (PBE), as well as (NFE) and (PFE) are the extensionality rules of EP, cf. Fig. 1. These rules eliminate the need for explicit extensionality axioms in the search space, which would enable cut-simulation [BBK09] and hence drastically hamper proof search. While the functional extensionality rules gradually ground the literals to base types and provide witnesses for the (in-)equality of function symbols to the search space, the Boolean extensionality rules enable the application of clausification rules to the Boolean typed sides of the literal, thereby lifting them into semantical proof search.

Unification. The unification rules of EP are a variant of Huet's unification rules and described in detail in previous work [Ste18, §3.2]. They can be eagerly applied to the unification constraints in clauses. In an extensional setting, syntactical search for unifiers and semantical proof search coincide, and unification transformations are regarded proper calculus rules. As a result, the unification rules might only partly solve (i.e., simplify) unification constraints and unification constraints themselves are eligible to subsequent inferences. The bundled unification rules are referred to as UNI.

A set Φ of sentences has a refutation in EP, denoted $\Phi \vdash \square$, iff the empty clause can be derived in EP. A clause is the empty clause, written $\square$, if it only consists

of flex-flex unification 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 necessarily unsatisfiable [Hen50].

Theorem 1 (Soundness and Completeness of EP)**.** *EP is sound and refutationally complete for HOL with Henkin semantics.*

Proof. See [Ste18, §3]. □ □

An example for a refutation in EP is given in the following:

Example 1 (Cantor's Theorem)**.** *Cantor's Theorem states that, given a set A, the power set of A has a strictly greater cardinality than A itself. The core argument of the proof can be formalized as follows:*

$$\neg\exists f_{o\iota\iota}.\, \forall Y_{o\iota}.\, \exists X_{\iota}.\, f\ X = Y \qquad (\mathbf{C})$$

Formula **C** *states that there exists no surjective function f from a set to its power set. A proof of* **C** *in EP makes use of functional extensionality, Boolean extensionality, primitive substitution as well as nontrivial higher-order pre-unification; it is given below.*

By convention, the application of a calculus rule (or of a compound rule) is stated with the respective premise clauses enclosed in parentheses after the rule name. For rule (PS), the second argument describes which general binding was used for the instantiation; e.g., $\mathrm{PS}(\mathcal{C}, \mathcal{GB}^t_\tau)$ *denotes an instantiation with an approximation of term t for goal type τ, cf. [BSPT15] for further details on general bindings.*

$\mathsf{CNF}(\neg\mathbf{C})$: $\mathcal{C}_1\colon [sk^1\ (sk^2\ \mathrm{X}^1) \simeq \mathrm{X}^1]^{\mathfrak{t}}$
$\mathrm{PFE}(\mathcal{C}_1)$: $\mathcal{C}_2\colon [sk^1\ (sk^2\ \mathrm{X}^1)\ \mathrm{X}^2 \simeq \mathrm{X}^1\ \mathrm{X}^2]^{\mathfrak{t}}$
$\mathrm{PBE}(\mathcal{C}_2)$: $\mathcal{C}_3\colon [sk^1\ (sk^2\ \mathrm{X}^1)\ \mathrm{X}^2]^{\mathfrak{t}} \vee [\mathrm{X}^1\ \mathrm{X}^2]^{\mathfrak{f}}$;
$\mathcal{C}_4\colon [sk^1\ (sk^2\ \mathrm{X}^3)\ \mathrm{X}^4]^{\mathfrak{f}} \vee [\mathrm{X}^3\ \mathrm{X}^4]^{\mathfrak{t}}$
$\mathrm{PS}(\mathcal{C}_3, \mathcal{GB}^{\neg}_{o\iota}), \mathsf{CNF}$: $\mathcal{C}_5\colon [sk^1\ \big(sk^2\ (\lambda Z_\iota.\, \neg(\mathrm{X}^5\ Z))\big)\ \mathrm{X}^2]^{\mathfrak{t}} \vee [\mathrm{X}^5\ \mathrm{X}^2]^{\mathfrak{t}}$
$\mathrm{PS}(\mathcal{C}_4, \mathcal{GB}^{\neg}_{o\iota}), \mathsf{CNF}$: $\mathcal{C}_6\colon [sk^1\ \big(sk^2\ (\lambda Z_\iota.\, \neg(\mathrm{X}^6\ Z))\big)\ \mathrm{X}^4]^{\mathfrak{f}} \vee [\mathrm{X}^6\ \mathrm{X}^4]^{\mathfrak{f}}$
$\mathrm{Fac}(\mathcal{C}_5), \mathsf{UNI}$: $\mathcal{C}_7\colon [sk^1\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\ Z)\big)\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\ Z)\big)]^{\mathfrak{t}}$
$\mathrm{Fac}(\mathcal{C}_6), \mathsf{UNI}$: $\mathcal{C}_8\colon [sk^1\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\ Z)\big)\ \big(sk^2\ \lambda Z_\iota.\, \neg(sk^1\ Z\ Z)\big)]^{\mathfrak{f}}$
$\mathrm{Para}(\mathcal{C}_7, \mathcal{C}_8), \mathsf{UNI}$: □

The Skolem symbols sk^1 and sk^2 used in the above proof have type $o\iota\iota$ and $\iota(o\iota)$, respectively and the X^i denote fresh free variables of appropriate type. A unifier $\sigma_{\mathcal{C}_7}$ generated by UNI *for producing $\mathcal{C}_7$ is given by (analogously for $\mathcal{C}_8$):*

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

Note that, together with the substitution $\sigma_{\mathcal{C}_3} \equiv \{\lambda Z_\iota.\, \neg(\mathrm{X}^5\ Z)/\mathrm{X}^1\}$ generated by approximating $\neg_{oo}$ via (PS) on $\mathcal{C}_3$, the free variable X^1 in $\mathcal{C}_1$ is instantiated 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 diagonal set of sk^1, given by $\{x \mid x \notin sk^1(x)\}$, as used in the traditional proofs of Cantor's Theorem; see, e.g., Andrews [AMCP84].

The TSTP representation of Leo-III's proof for this problem is presented in Fig. 4.

3.1 Extended Calculus

Leo-III implements several additional calculus rules that are not captured by the EP calculus from Fig. 1. These rules are practically motivated, partly heuristic, and primarily target technical issues that complicate effective automation in practice. They are as follows (see earlier publications [SB18, Ste18] for further details):

Improved clausification. Leo-III employs definitional clausification [WSKB16] to reduce the number of clauses created during clause normalization. Moreover, miniscoping is employed prior to clausification.

Clause contraction. Leo-III implements equational simplification procedures, including subsumption, destructive equality resolution, heuristic rewriting and contextual unit cutting (simplify-reflect) [Sch02].

Defined equalities. Common notions of defined equality predicates (Leibniz equality, Andrews equality) are heuristically replaced with primitive equality predicates.

Choice. Leo-III implements additional calculus rules for reasoning with choice.

Function synthesis. If plain unification fails for a set of unification constraints, Leo-III may try to synthesize functions that meet the specifications represented by the unification constraint. This is done using special choice instances that simulate if-then-else terms which explicitly enumerate the desired input output relation of that function. In general, this rule tremendously increases the search space, and it also enables Leo-III to solve some hard problems with TPTP rating 1.0 that were not solved by any ATP system before.

Injective functions. Leo-III addresses improved reasoning with injective functions by postulating the existence of left inverses for function symbols that are inferred to be injective, see also below.

Further rules. Prior to clause normalization, Leo-III might instantiate universally quantified variables with heuristically chosen terms. This includes the exhaustive instantiation of finite types (such as o, oo, etc.) as well as partial instantiation for other interesting types (such as $o\tau$ for some type τ).

The addition of the above calculus rules to EP in Leo-III enables solving various problems that can otherwise not be solved (in reasonable resource limits). An example problem that could not be solved by any higher-order ATP system before is the following:

Example 2 (Cantor's Theorem, revisited)**.** *Another possibility to encode Cantor's theorem is by using a formulation based on injectivity:*

$$\neg\exists f_{\iota(o\iota)}.\, \forall X_{o\iota}.\, \forall Y_{o\iota}.\, (f\ X = f\ Y) \Rightarrow X = Y \qquad (\mathbf{C}')$$

Here, the nonexistence of an injective function from a set's power set to the original set is postulated. This conjecture can easily be proved using Leo-III's

injectivity rule (INJ) *that, given a fact stating that some function symbol f is injective, introduces the left inverse of f, say f^{inv}, as fresh function symbol to the search space. The advantage is that f^{inv} is then available to subsequent inferences and can act as an explicit evidence for the existence of such a function. The full proof of $\mathbf{C}'$ is as follows:*

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

The introduced Skolem symbol sk is of type $\iota(o\iota)$ and its left inverse, denoted sk^{inv} of type $o\iota\iota$, is inferred by (INJ) *based on the injectivity specification given by clause $\mathcal{C}_0$. This problem is part of the TPTP library as problem* `SYO037^1` *and could not be solved by any existing HO ATP system before.*

4 System Architecture and Implementation

As mentioned before, the main goal of the Leo-III prover is to achieve effective automation of reasoning in HOL, and, in particular, to address the shortcomings of resolution based approaches when handling equality. To that end, the complete EP calculus presented in §3 has been implemented as the theoretical foundation underlying the automated theorem prover Leo-III. Although EP is still unordered and Leo-III therefore generally suffers from the same drawbacks as experienced in first-order paramodulation, including state space explosions and a prolific proof search, the idea is to anyway use EP as a basis for Leo-III and to pragmatically tackle the problems with additional calculus rules (cf. §3.1), and optimizations resp. heuristics on the implementation level.

An overview of Leo-III's top level architecture is displayed in Fig. 2. After parsing the problem statement, a symbol based relevance filter adopted from Meng and Paulson [MP09] is employed for premise selection. The input formulas that pass the relevance filter are translated into polymorphically typed λ-terms (Interpreter) and then passed to a saturation procedure. The saturation process is controlled by a dedicated module (Control) that manages, selects and applies different heuristics that may restrict or guide the application of calculus rules. Leo-III makes use of external (first-order) ATP systems for discharging proof obligations. If any external 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. Invocation, translation and utilization of the external results is also controlled by the Control module. Indexing data structures are employed for speeding up frequently used procedures.

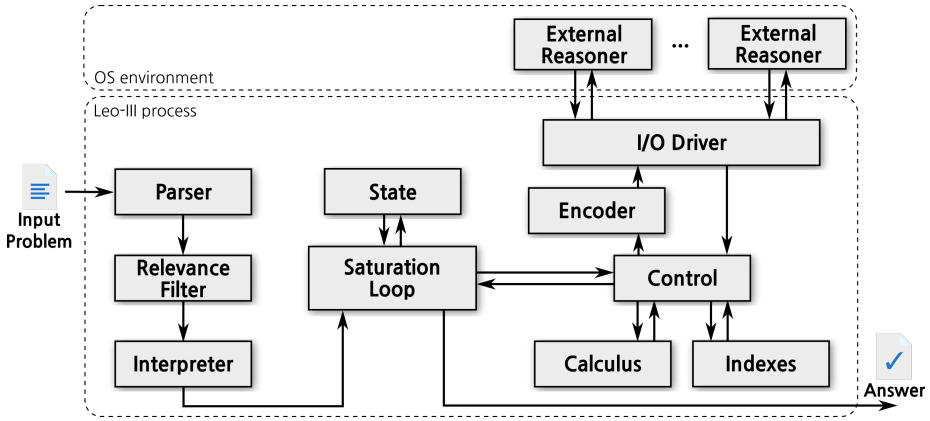

Figure 2: Schematic diagram of Leo-III's architecture. The arrows indicate directed information flow. The external reasoners are executed asynchronously (non-blocking) as dedicated processes of the operating system.

The current status of the saturation process, including selected parameters and statistical information, is maintained in the State component. The strict separation between saturation, state, control and calculus makes the Leo-III fairly simple to maintain and extend.

4.1 Proof search

The overall proof search procedure of Leo-III consists of three consecutive phases: preprocessing, saturation and proof reconstruction.

During preprocessing, the input formulas are transformed into a fully Skolemized $\beta\eta$-normal clausal normal form. In addition, methods including definition expansion, simplification, miniscoping, replacement of defined equalities, and clause renaming [WSKB16] are applied, cf. Steen's thesis for details [Ste18].

Saturation is organized as a sequential procedure that iteratively saturates the set of input clauses with respect to EP (and its extensions) until the empty clause is derived. 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. Since in most cases the number of clauses that can be generated during proof search is infinite, the saturation process is limited artificially using time resource bounds that can be configured by the user.

Leo-III employs a variant of the DISCOUNT [DKS97] loop that has its intellectual roots in the E prover [Sch02]. Nevertheless, some modifications are necessary to address the specific requirements of reasoning in HOL. 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 renormalized. 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 as an isolated procedure on the meta-level as done in first-order procedures. As opposed to the first-order case, clauses that have unsolvable unification constraints are not discarded but nevertheless inserted into the search space. This is necessary in order to retain completeness.

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 refutations [Sut07], cf. §4.4 for details.

4.2 Polymorphic Reasoning

Proof assistants such as Isabelle/HOL [NPW02] and Coq [BC04] are based on type systems that extend simple types with, e.g., polymorphism, type classes, dependent types and further type concepts. Expressive type systems allow structuring knowledge in terms of reusability and are of major importance in practice.

Leo-III supports reasoning in first-order 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 standard THF syntax [SB10] for HOL. The extension of Leo-III to polymorphic reasoning does not require modifications of the general proof search process as presented further above. Also, the data structures of Leo-III are already expressive enough to represent polymorphic formulas, cf. technical details in earlier work [SWB17].

Central to the polymorphic adaption of Leo-III's calculus is the notion of type unification. Type unification between two types τ and ν yields a substitution σ such that $\tau\sigma \equiv \nu\sigma$, if such a substitution exists. The most general type unifier is then defined analogously to term unifiers. Since unification on rank-1 polymorphic types is essentially a first-order unification problem, it is decidable and unitary, i.e., it yields a unique most general unifier if one exists. Intuitively, whenever a calculus rule of EP requires two premises to have the same type, it then suffices in the polymorphic extension of EP to require that the types are unifiable. For a concrete inference, the type unification is then applied first to the clauses, followed by the standard inference rule itself.

Additionally, 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. A related approach is employed by Wand in the extension of the first-order ATP system SPASS to polymorphic types [Wan17].

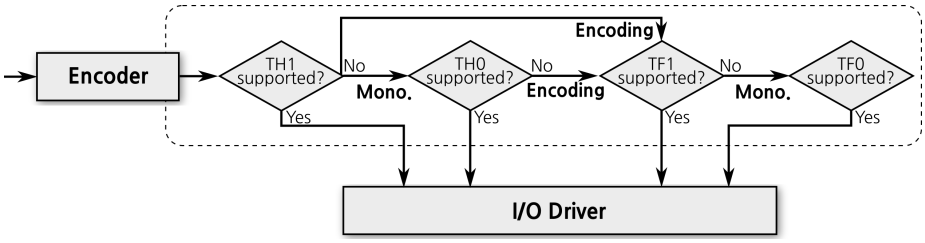

Figure 3: Translation process in the encoder module of Leo-III. Depending on the supported logic fragments of the respective external reasoner, the clause set is translated to different logic formalism: Polymorphic HOL (TH1), monomorphic HOL (TH0), polymorphic first-order logic (TF1) or monomorphic (many-sorted) first-order logic (TF0).

4.3 External Cooperation

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 (cf. Encoder module in Fig. 2) that translates higher-order clauses to polymorphic or 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 the encoding of 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 provers. Various different systems, including first-order and higher-order ATP systems and model finders, can in fact be used simultaneously, provided that they comply with some common TPTP language standard. Fig. 3 displays the translation pipeline of Leo-III for connecting to external ATP systems. The Control module of Leo-III will automatically select the most suitable encoding for each system [SWSB17] and translate the proof obligations to one of the four TPTP dialects TF0 [SSCB12], TF1 [BP13b], TH0 [SB10] and TH1 [KSR16]. The translation process combines heuristic monomorphization [Böh12, BBPS16] steps with standard encodings of higher-order language features [MP08] in first-order logic.

4.4 Input and Output

Leo-III accepts all common TPTP dialects [Sut17], including untyped clause normal form (CNF), untyped and typed first-order logic (FOF and TFF, respectively) and, as primary input format, monomorphic higher-order logic (THF) [SB10]. Additionally, as one of the first higher-order ATP systems, Leo-III supports reasoning in rank-1 polymorphic variants of such logics using the TF1 [BP13b] and TH1 [KSR16] languages.

Leo-III rigorously implements the machine-readable TSTP result standard [Sut07] and hence outputs appropriate SZS ontology values [Sut08]. The use of the TSTP output format allows for simple means of communi-

cation and exchange of reasoning results between different reasoning tools and, consequently, eases the employment of Leo-III within external tools. Novel to the list of supported SZS result values for the Leo prover family is `ContradictoryAxioms` [Sut08], which is printed if the input axioms were found to be inconsistent during the proof run (i.e., if the empty clause could be derived without using the conjecture even once). Using this simple approach, Leo-III identified 15 problems from the TPTP library to be inconsistent without any special setup.

Additional to the above described SZS result value, Leo-III produces machine readable proof certificates if a proof was found and such a certificate has been requested. 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 in the refutation. The proof output records all intermediate inferences. The representation again follows the TSTP format and records the inferences using annotated `THF` formulas. 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]. A detailed description of Leo-III's proof output format and the information contained therein can be found in Steen's PhD thesis [Ste18, §4.5]. An example of such a proof output is displayed in Fig. 4.

4.5 Implementation Details

Leo-III implements a combination of term representation techniques; term datastructures are provided that admit expressive typing, efficient basic term operations and reasonable memory consumption [BSW17]. Leo-III employs 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 leads to an internal distinction between β-normal and (possibly) non-β-normal spine terms. The first kind has an optimized representation, where the head is only associated with an integer representing a constant symbol or variable.

Additionally, the term representation incorporates explicit substitutions [ACCL91]. 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.

Furthermore, Leo-III implements a locally nameless representation using de Bruijn indices [Bru72]. In the setting of polymorphism [SWB17], 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 represen-

```
% SZS status Theorem for sur_cantor_th1.p
% 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, ! [T: $tType]: (
                                ~ ( ?[F:T > (T > $o)]: (
                                      ![Y:T > $o]: ?[X:T]: ((F @ X) = Y) ) )),
    file('sur_cantor_th1.p',sur_cantor) ).
thf(2, negated_conjecture, ~ ! [T:$tType]: (
                                ~ ( ?[F:T > (T > $o)]: (
                                      ![Y:T > $o]: ?[X:T]: ((F @ X) = Y) ) )),
    inference(neg_conjecture,[status(cth)],[1]) ).
thf(4,plain,! [A:skt1 > $o]: (sk1 @ (sk2 @ A) = A),
    inference(cnf,[status(esa)],[2]) ).
thf(6,plain,! [B:skt1, A:skt1 > $o]: ((sk1 @ (sk2 @ A) @ B) = (A @ B)),
    inference(func_ext,[status(esa)],[4])).
thf(8,plain,! [B:skt1, A:skt1 > $o]: ((sk1 @ (sk2 @ A) @ B) | ~ (A @ B)),
    inference(bool_ext,[status(thm)],[6])).
thf(272,plain,! [B:skt1, A:skt1 > $o]: ( sk1 @ (sk2 @ A) @ B) |
                            ((A @ B) != ~ (sk1 @ (sk2 @ A) @ B)) | ~$true),
    inference(eqfactor_ordered,[status(thm)],[8])).
thf(294,plain,sk1 @ (sk2 @ (^ [A:skt1]: ~ (sk1 @ A @ A)))
                  @ (sk2 @ (^ [A:skt1]: ~ (sk1 @ A @ A))),
    inference(pre_uni,[status(thm)],[272:[
              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(33,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, $thf(sk2 @ (^ [C:skt1]: ~ (sk1 @ C @ C))))]])).
thf(320,plain,$false,inference(rewrite,[status(thm)],[294,33])).
% SZS output end CNFRefutation for sur_cantor_th1.p
```

Figure 4: Proof output of Leo-III for the polymorphic variant (TH1 syntax) of the surjective variant of Cantor's theorem.

tations 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.

Terms are perfectly shared within Leo-III, meaning that each term is only constructed once and then reused between different occurrences. This reduces memory consumption in large knowledge bases and it allows constant time term comparison for syntactic equality using the term's pointer to its unique physical representation. For fast basic term retrieval operations (such as access of a head symbol, subterm occurrences, etc.) terms are kept in β-normal η-long form.

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 [WSB15], which is freely available at GitHub.[5] This framework provides many 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 proof representations.

[5] Leos Parallel Architecture and Datastructures (LEOPARD) can be found at `https://github.com/leoprover/LeoPARD`.

5 Reasoning in Non-Classical Logics

Computer-assisted reasoning in non-classical logics (NCL) is of increasing relevance for applications in artificial intelligence, computer science, mathematics and philosophy. However, with a few exceptions, most of the available systems focus on classical logics only, including common contemporary first-order and higher-order theorem proving systems. In particular for quantified NCLs there are only very few systems available to date.

As an alternative to the development of specialized theorem proving systems, usually one for each targeted NCL, a shallow semantical embedding (SSE) approach allows for a simple adaptation of existing higher-order reasoning systems to a broad variety of expressive logics [Ben19]. In the SSE approach, the non-classical target logic is shallowly embedded in HOL by providing a direct encoding of its semantics, typically a set theoretic or relational semantics, within the term language of HOL. As a consequence, deciding validity in the target logic is reduced to higher-order reasoning and HOL ATP systems can be applied for this task. Note that this technique, in principal, also allows off-the-shelf automation even for quantified NCLs as quantification and binding mechanisms of the HOL meta logic can be utilized. This is an interesting option in many application areas, e.g., in ethical and legal reasoning, as the respective community do not yet agree on which logical system should actually be preferred. The resource intensive implementation of dedicated new provers for each potential system is not an adequate option for rapid prototyping of prospective candidate logics and can be avoided using SSEs.

Leo-III is addressing this gap. In addition to its HOL reasoning capabilities, it is the first system that natively supports reasoning in a wide range of normal higher-order modal logics (HOMLs) [GSB17]. To achieve this, Leo-III internally implements the SSE approach for quantified modal logics based on their Kripke-style semantics [BvBW06, BP10].

Quantified modal logics are associated many different notions of semantics [BvBW06]. Differences may, e.g., occur in the interaction between quantifiers and the modal operators, as expressed by the Barcan formulas [Bar46], or regarding the interpretation of constant symbols as rigid or non-rigid. Hence, there are various subtle but meaningful variations in multiple individual facets of which each combination potentially yields a distinct modal logic. Since many of those variations have their particular applications, there is no reasonably small subset of generally preferred modal logics to which a theorem proving system should be restricted. This, of course, poses a major practical challenge. Leo-III, therefore, supports a very wide range of quantified modal logics [GSB17]. In contrast, other ATP systems for (first-order) quantified modal logics such as MleanCoP [Ott14] and MSPASS [HS00] only support a comparably small subset of all possible variants.

Unlike in classical logic, a problem statement comprised only of axioms and a conjecture to prove does not yet fully specify a reasoning task in quantified modal logic. It is necessary to also explicitly state the intended semantical details in which the problem is to be attacked. This is realized by including a meta-logical specification entry in the header of the modal logic problem file in form of a TPTP THF formula of role `logic`. This formula then specifies respective details for each relevant semantic dimension, cf. [GS18] for more details on the specification syntax. An example is displayed in Fig. 5. The

```
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))))))).
```

Figure 5: A corollary of Becker's postulate formulated in modal THF, representing the formula $\forall P_{\iota\to o}\forall F_{\iota\to\iota}\forall X_\iota\exists G_{\iota\to\iota}(\Diamond\Box P(F(X)) \Rightarrow \Box P(G(X)))$. The first statement specifies the modal logic to be logic S5 with constant domain quantification, rigid constant symbols and a global consequence relation.

identifiers `$constants`, `$quantification` and `$consequence` in the given case specify that constant symbols are rigid, that the quantification semantics is constant domain, and that the consequence relation is global, respectively, and `$modalities` specifies the properties of the modal connectives by means of fixed modal logic system names, such as S5 in the given case, or, alternatively, by listing individual names of modal axiom schemes. This logic specification approach was developed in earlier work [WSB16] and subsequently improved and enhanced to a work-in-progress TPTP language extension proposal.[6]

When being invoked on a modal logic problem file as displayed in Fig. 5, Leo-III parses and analyses the logic specification part, automatically selects and unfolds the corresponding definitions of the SSE approach, adds appropriate axioms and then starts reasoning in (meta logic) HOL. This process is visualized in Fig. 6. Subsequently, Leo-III returns SZS compliant result information and, if successful, also a proof object in TSTP format. Leo-III's proof output for the example from Fig. 5 is displayed in appendix A; it shows the relevant SSE definitions that have been automatically generated by Leo-III according to the given logic specification, and this file can be verified by GDV [Sut17]. Previous experiments [GSB17, BR13] have shown that the SSE approach offers an effective automation of embedded non-classical logics for the case of quantified modal logics.

As of version 1.2, Leo-III supports, but is not limited to, first-order and higher-order extensions of the well known modal logic cube [BvBW06]. When taking the different parameter combinations into account this amounts to more than 120 supported HOMLs. The exact number of supported logics is in fact much higher, since Leo-III also supports multi-modal logics with independent modal system specification for each modality. Also, user-defined combinations of rigid and non-rigid constants and different quantification semantics per type domain are possible. In addition to modal logic reasoning, Leo-III also integrates SSEs of deontic logics [BFP18].

6 Evaluation

In order to quantify the performance of Leo-III, an evaluation based on various benchmarks was conducted, cf. [SB18]. Three benchmark data sets were used:

- *TPTP TH0* (2463 problems) is the set of all monomorphic HOL (TH0) problems from the TPTP library v7.0.0 [Sut17] that are annotated as

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

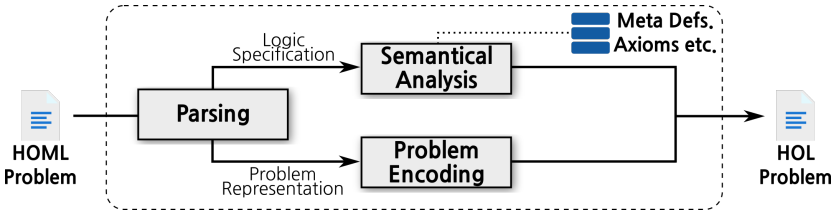

Figure 6: Schematic structure of the embedding preprocessing procedure in Leo-III.

theorems. The TPTP library is a de facto standard for the evaluation of ATP systems.

- *TPTP TH1* (442 problems) is the subset of all 666 polymorphic HOL (TH1) problems from TPTP v7.0.0 that are annotated as theorems and do not contain arithmetic. The problems mainly consist of HOL Light core exports and Sledgehammer translations of various Isabelle/HOL theories.
- *QMLTP* (580 problems) is the subset of all mono-modal benchmarks from the QMLTP library 1.1 [RO12]. The QMLTP library only contains propositional and first-order modal logic problems. Since each problem may have a different validity status for each semantic notion of modal logic, all problems are selected. The total number of tested benchmarks in this category thus is 580 (raw problems) × 5 (modal systems) × 3 (quantification semantics). QMLTP assumes rigid constant symbols and a local consequence relation; this is adopted here.

The evaluation measurements were taken on the StarExec cluster in which each compute node is a 64 bit Red Hat Linux (kernel 3.10.0) machine with 2.40 GHz quad-core processors and a main memory of 128 GB. For each problem, every prover was given a CPU time limit of 240 s. The following theorem provers were employed in one or more of the benchmark sets (indicated in parentheses): Leo-III 1.2 (TH0, TH1, QMLTP) used with E, CVC4 and iProver as external first-order ATP systems, Isabelle/HOL 2016 [NPW02] (TH0, TH1), Satallax 3.0 [Bro12] (TH0), Satallax 3.2 (TH0), LEO-II 1.7.0 (TH0), Zipperposition 1.1 (TH0) and MleanCoP 1.3 [Ott14] (QMLTP).

The experimental results are discussed next; additional details on Leo-III's performance are presented in Steen's thesis [Ste18].

TPTP TH0. Table 1 (a) displays each system's performance on the TPTP TH0 data set. For each system the absolute number (Abs.) and relative share (Rel.) of solutions is displayed. Solution here means that a system is able to establish the SZS status `Theorem` and also emits a proof certificate that substantiates this claim. 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.[7] Additionally, the average and sum of all CPU times and wall clock (WC)

[7]Remark on CAX: In this special case of THM (theorem) the given axioms are inconsistent, so that anything follows, including the given conjecture. Hence, it is counted against solved problems.

Table 1: Detailed result of the benchmark measurements.

Systems	Solutions		SZS Results				Avg. Time [s]		Σ Time [s]	
	Abs.	Rel.	THM	CAX	GUP	TMO	CPU	WC	CPU	WC
Satallax 3.2	2140	86.89	2140	0	2	321	12.26	12.31	26238	26339
Leo-III	2053	83.39	2045	8	16	394	15.39	5.61	31490	11508
Satallax 3.0	1972	80.06	2028	0	2	433	17.83	17.89	36149	36289
LEO-II	1788	72.63	1789	0	43	631	5.84	5.96	10452	10661
Zipperposition	1318	53.51	1318	0	360	785	2.60	2.73	3421	3592
Isabelle/HOL	0	0.00	2022	0	1	440	46.46	33.44	93933	67610

(a) TPTP TH0 data set (2463 problems)

Systems	Solutions		SZS Results				Avg. Time [s]		Σ Time [s]	
	Abs.	Rel.	THM	CAX	GUP	TMO	CPU	WC	CPU	WC
Leo-III	185	41.86	183	2	8	249	49.18	24.93	9099	4613
Isabelle/HOL	0	0.00	237	0	23	182	93.53	81.44	22404	19300

(b) TPTP TH1 data set (442 problems)

times over all solved problems is presented.

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 (3.2) that successfully solves 2140 problems, which is approximately 4.24 % more than Leo-III. Isabelle currently does not emit proof certificates (hence zero solutions). Even if results without explicit proofs are counted, Leo-III would still have a slightly higher number of problems solved than Satallax 3.0 and Isabelle/HOL with 25 (1.22 %) and 31 (1.51 %) additional solutions, respectively. Leo-III, Satallax (3.2), Zipperposition and LEO-II produce 18, 17, 15 and 3 unique solutions, respectively. Evidently, Leo-III currently produces more unique solutions than any other ATP system in this setting. Leo-III solves twelve problems that are currently not solved by any other system indexed by TPTP.[8]

Satallax, LEO-II and Zipperposition show only small differences between their individual 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 used CPU time to used wall clock time over all solved problems. The core usage of Leo-III for the TPTP TH0 data set is 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.

TPTP TH1. Currently, there exist only few ATP systems that are capable of reasoning within polymorphic HOL as specified by TPTP TH1. The only exceptions are HOL(y)Hammer and Isabelle/HOL that schedule proof tactics within HOL Light and Isabelle/HOL, respectively. Unfortunately, only Isabelle/HOL was available for instrumentation in a reasonably recent and stable version. Table 1 (b) displays the measurement results for the TPTP TH1 data set. When

[8] This information is extracted from the TPTP problem rating information that is attached to each problem. The unsolved problems are `NLP004^7`, `SET013^7`, `SEU558^1`, `SEU683^1`, `SEV143^5`, `SYO037^1`, `SYO062^4.004`, `SYO065^4.001`, `SYO066^4.004`, `MSC007^1.003.004`, `SEU938^5` and `SEV106^5`.

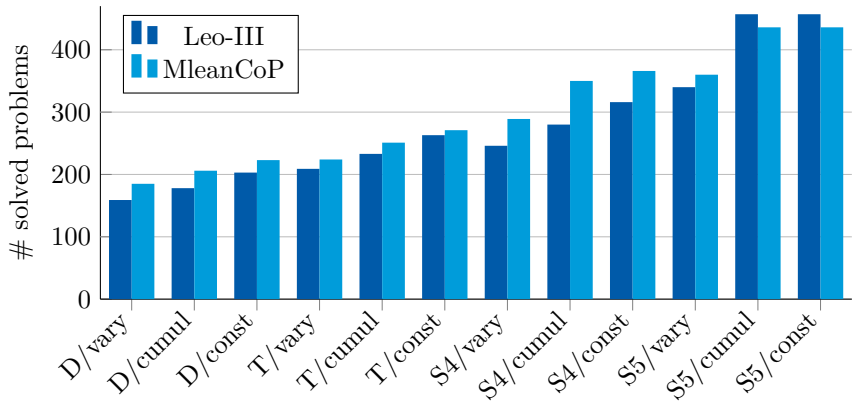

Figure 7: Comparison of Leo-III and MleanCoP on the QMLTP data set (580 problems).

disregarding proof certificates, Isabelle/HOL finds 237 theorems (53.62 %) which is roughly 28.1 % more than the number of solutions founds by Leo-III. Leo-III and Isabelle/HOL produce 35 and 69 unique solutions, respectively.

QMLTP. For each semantical setting supported by MleanCoP, which is the strongest first-order modal logic prover available to date [BOR12], the number of theorems found by both Leo-III and MleanCoP in the QMLTP data set is presented in Fig. 7. Leo-III is fairly competitive with MleanCoP (weaker by maximal 14.05 %, minimal 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 maximal 20.00 %, minimal 13.66 % and 16.18 % on average). For **S5** variants, Leo-III is very effective (stronger by 1.36 % on average), and it is ahead of MleanCoP for **S5**/const and **S5**/cumul (which coincide). This is due to the encoding of the **S5** accessibility relation in Leo-III 1.2 as the universal relation between possible worlds as opposed to its prior encoding as an equivalence relation [GSB17]. Leo-III contributes 199 solutions to previously unsolved problems.

On polymorphism. The GRUNGE evaluation by Brown et al. [BGK+19] aims at comparing ATP systems across different supported logics. For this purpose, theorems from the HOL4 standard library [SN08] are translated into multiple different logical formalisms, including untyped first-order logic, typed first-order logic (with and without polymorphic types) and higher-order logic (with and without polymorphic types) using the different TPTP language dialects as discussed in §4.4. Of the many first-order and higher-order ATP systems that are evaluated on these data sets, Leo-III is one of the few to support polymorphic types.[9] This seems a major strength in the context of GRUNGE: Leo-III is identified as the most effective ATP system overall in terms of solved problems in any formalism, with approx. 19% more solutions than the next best system, and as the best ATP system in all higher-order formalisms, with up to

[9] HOLyHammer (HOL ATP) and Zipperposition (first-order ATP) are the only other systems supporting polymorphism.

94% more solutions than the next best higher-order system. Remarkably, it can be seen that over 90% of all solved problems in the GRUNGE evaluation are contributed by Leo-III on the basis of the polymorphic higher-order data set, and the next best result in any other formalism is down by approx. 25%.

This suggests that reasoning in polymorphic formalisms is of particular benefit for applications in mathematics and, possibly, further domains. For systems without native support for (polymorphic) types, types are usually encoded as terms, or they are removed by monomorphization. This increases the complexity of the problem representation and decreases reasoning effectivity. Leo-III, on the other hand, handles polymorphic types natively and requires no such indirection.

7 Conclusion and Future Work

Leo-III is an ATP system for classical HOL with Henkin semantics, and it natively supports also various propositional and quantified non-classical logics. This includes typed and untyped first-order logic, polymorphic HOL, and a wide range of HOMLs, which makes Leo-III, up to our knowledge, the most widely applicable theorem proving system available to date. Recent evaluations show that Leo-III is also very effective and that in particular Leo-III's extension to polymorphic HOL is practically relevant.

Future work includes extensions and specializations of Leo-III for selected deontic logics and logic combinations, with the ultimate goal to support effective automation of normative reasoning. Additionally, stemming from the success of polymorphic reasoning in Leo-III, a polymorphic adaption of the modal logic SSE approach is planned, potentially improving modal logic reasoning performance.

References

[AB06] Peter B. Andrews and Chad E. Brown. TPS: A hybrid automatic-interactive system for developing proofs. *J. Applied Logic*, 4(4):367–395, 2006.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[AMCP84] Peter B Andrews, Dale A Miller, Eve Longini Cohen, and Frank Pfenning. Automating higher-order logic. *Contemporary Mathematics*, 29:169–192, 1984.

[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.

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory.* Applied Logic Series. Springer, 2002.

[B+11] Clark Barrett et al. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[Bar46] Ruth C. Barcan. A functional calculus of first order based on strict implication. *J. Symb. Log.*, 11(1):1–16, 1946.

[BBCW18] Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 28–46. Springer, 2018.

[BBK04] Christoph Benzmüller, Chad E. Brown, and Michael Kohlhase. Higher-order semantics and extensionality. *J. Symb. Log.*, 69(4):1027–1088, 2004.

[BBK09] Christoph Benzmüller, Chad E. Brown, and Michael Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1), 2009.

[BBP13] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with SMT solvers. *J. Autom. 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.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[BDS13] Henk P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types.* Perspectives in logic. Cambridge University Press, 2013.

[Ben99a] Christoph Benzmüller. *Equality and extensionality in automated higher order theorem proving.* PhD thesis, 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*, volume 1632 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 1999.

[Ben11] Christoph Benzmüller. Combining and automating classical and non-classical logics in classical higher-order logics. *Ann. Math. Artif. Intell.*, 62(1-2):103–128, 2011.

[Ben17] Christoph Benzmüller. Cut-elimination for quantified conditional logic. *J. Philosophical Logic*, 46(3):333–353, 2017.

[Ben19] Christoph Benzmüller. Universal (meta-)logical reasoning: Recent successes. *Science of Computer Programming*, 172:48–62, March 2019.

[BFP18] Christoph Benzmüller, Ali Farjami, and Xavier Parent. A dyadic deontic logic in HOL. In Jan M. Broersen, Cleo Condoravdi, Nair Shyam, and Gabriella Pigozzi, editors, *Deontic Logic and Normative Systems - 14th International Conference, DEON 2018, Utrecht, The Netherlands, July 3-6, 2018.*, pages 33–49. College Publications, 2018.

[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.

[BGK+19] C. Brown, T. Gauthier, C. Kaliszyk, G. Sutcliffe, and J. Urban. GRUNGE: A Grand Unified ATP Challenge. In Pascal Fontaine, editor, *Automated Deduction - CADE-27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 23-30, 2019. Proceedings*, LNCS. Springer, 2019. In print.

[BK98] 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 Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 215–254. Elsevier, 2014.

[BN10] 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.

[Böh12] Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

[BOR12] Christoph Benzmüller, Jens Otten, and Thomas Raths. Implementing and evaluating provers for first-order modal logics. In Luc De Raedt et al., editors, *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 163–168. IOS Press, 2012.

[BP10] Christoph Benzmüller and Lawrence C. Paulson. Multimodal and intuitionistic logics in simple type theory. *Logic Journal of the IGPL*, 18(6):881–892, 2010.

[BP13a] Christoph Benzmüller and Lawrence C. Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis*, 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.

[BR13] Christoph Benzmüller and Thomas Raths. HOL based first-order modal logic provers. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 2013.

[BR18] Ahmed Bhayat and Giles Reger. Set of support for higher-order reasoning. In Boris Konev, Josef Urban, and Philipp Rümmer, editors, *Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning co-located with Federated Logic Conference 2018 (FLoC 2018), Oxford, UK, July 19th, 2018.*, volume 2162 of *CEUR Workshop Proceedings*, pages 2–16. CEUR-WS.org, 2018.

[BR19] Ahmed Bhayat and Giles Reger. Restricted combinatory unification. In Pascal Fontaine, editor, *Automated Deduction - CADE-27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 23-30, 2019. Proceedings*, LNCS. Springer, 2019. In print.

[BREO+19] Haniel Barbosa, Andrew Reynolds, Daniel El Ouraoui, Cesare Tinelli, and Clark Barrett. Extending SMT solvers to higher-order logic. In Pascal Fontaine, editor, *Automated Deduction - CADE-27 - 27th International Conference on Automated Deduction, Natal,*

Brazil, August 23-30, 2019. Proceedings, LNCS. Springer, 2019. In print.

[Bro12] Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 111–117. Springer, 2012.

[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.

[BS16] Christoph Benzmüller and Dana S. Scott. Automating free logic in Isabelle/HOL. In Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew J. Sommese, editors, *Mathematical Software - ICMS 2016 - 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings*, volume 9725 of *Lecture Notes in Computer Science*, pages 43–50. Springer, 2016.

[BSPT15] Christoph Benzmüller, Nik Sultana, Lawrence C. Paulson, and Frank Theiss. The higher-order prover LEO-II. *J. Autom. Reasoning*, 55(4):389–404, 2015.

[BSW17] Christoph Benzmüller, Alexander Steen, and Max Wisniewski. Leo-III Version 1.1 (System description). 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*. EasyChair, 2017.

[BvBW06] Patrick Blackburn, Johan FAK van Benthem, and Frank Wolter. *Handbook of modal logic*, volume 3. Elsevier, 2006.

[BW16] Christoph Benzmüller and Bruno Woltzenlogel Paleo. The inconsistency in Gödel's Ontological Argument: A success story for AI in metaphysics. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 936–942. IJCAI/AAAI Press, 2016.

[BWB+11] Jasmin Christian Blanchette, Tjark Weber, Mark Batty, Scott Owens, and Susmit Sarkar. Nitpicking C++ concurrency. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 113–124. ACM, 2011.

[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] Christoph Benzmüller, L. Weber, and Bruno Woltzenlogel Paleo. Computer-assisted analysis of the Anderson-Hájek ontological controversy. *Logica Universalis*, 11(1):139–151, 2017.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.

[CP03] Iliano Cervesato and Frank Pfenning. A linear spine calculus. *J. Log. Comput.*, 13(5):639–688, 2003.

[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.

[DH86] Vincent J. Digricoli and Malcolm C. Harrison. Equality-based binary resolution. *J. ACM*, 33(2):253–289, 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.

[FB17] David Fuenmayor and Christoph Benzmüller. Types, tableaus and gödel's god in Isabelle/HOL. *Archive of Formal Proofs*, 2017, 2017.

[Fre79] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Verlag von Louis Nebert, Halle, 1879.

[Gol81] Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[GS18] Tobias Gleißner and Alexander Steen. The MET: The art of flexible reasoning with modalities. In Christoph Benzmüller, Francesco Ricca, Xavier Parent, and Dumitru Roman, editors, *Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings*, volume 11092 of *LNCS*, pages 274–284. Springer, 2018.

[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, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 14–30. EasyChair, 2017.

[H+15] Thomas C. Hales et al. A formal proof of the kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[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, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66. Springer, 2009.

[Hen50] Leon Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):81–91, 1950.

[HS00] Ullrich Hustadt and Renate A. Schmidt. MSPASS: modal reasoning by translation and first-order resolution. In Roy Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings*, volume 1847 of *Lecture Notes in Computer Science*, pages 67–71. Springer, 2000.

[Hue73] Gerard P. Huet. The undecidability of unification in third order logic. *Information and control*, 22(3):257–267, 1973.

[KBZ19] Daniel Kirchner, Christoph Benzmüller, and Edward N. Zalta. Computer science and metaphysics: A cross-fertilization. *Open Philosophy (Special Issue – Computer Modeling in Philosophy)*, 2019. To appear, preprint: `http://doi.org/10.13140/RG.2.2.25229.18403`.

[Kor08] Konstantin Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.

[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.

[Lei89] Gottfried Wilhelm Leibniz. Discourse on metaphysics. In Leroy E. Loemker, editor, *Philosophical Papers and Letters*, pages 303–330. Springer Netherlands, Dordrecht, 1989.

[Lin14] Fredrik Lindblad. A focused sequent calculus for higher-order logic. 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 61–75. Springer, 2014.

[Mil83] Dale A Miller. *Proofs in higher-order logic*. PhD thesis, Carnegie-Mellon University, 1983.

[Mil91] Dale A. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[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.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[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.

[Ott14] Jens Otten. MleanCoP: A connection prover for first-order modal logic. 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 269–276. Springer, 2014.

[RO12] Thomas Raths and Jens Otten. The QMLTP problem library for first-order modal logics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *LNCS*, pages 454–461. Springer, 2012.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[RW69] George Robinson and Larry Wos. Paramodulation and theorem-proving in first-order theories with equality. *Machine intelligence*, 4:135–150, 1969.

[SB10] Geoff Sutcliffe and Christoph Benzmüller. Automated reasoning in higher-order logic using the TPTP THF infrastructure. *J. Formalized Reasoning*, 3(1):1–27, 2010.

[SB16] Alexander Steen and Christoph Benzmüller. Sweet SIXTEEN: Automation via embedding into classical higher-order logic. *Logic and Logical Philosophy*, 25(4):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, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *LNCS*, pages 108–116. Springer, 2018.

[SBA06] Jörg H. Siekmann, Christoph Benzmüller, and Serge Autexier. Computer supported mathematics with ΩMEGA. *J. Applied Logic*, 4(4):533–559, 2006.

[Sch02] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2,3):111–126, 2002.

[SN08] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008.

[SSCB12] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP typed first-order form with arithmetic. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 406–419. Springer, 2012.

[Ste18] Alexander Steen. *Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III*, volume 345 of *DISKI*. Akademische Verlagsgesellschaft AKA GmbH, Berlin, 9 2018. Dissertation, Freie Universität Berlin, Germany.

[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, 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.

[Sut17] Geoff Sutcliffe. The TPTP problem library and associated infrastructure - from CNF to TH0, TPTP v6.4.0. *J. Autom. Reasoning*, 59(4):483–502, 2017.

[SWB16] Alexander Steen, Max Wisniewski, and Christoph Benzmüller. Agent-based HOL reasoning. In Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew J. Sommese, editors, *Mathematical Software - ICMS 2016 - 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings*, volume 9725 of *LNCS*, pages 75–81. Springer, 2016.

[SWB17] Alexander Steen, Max Wisniewski, and Christoph Benzmüller. Going polymorphic - TH1 reasoning for Leo-III. 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*. EasyChair, 2017.

[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*. EasyChair, 2017.

[VBCS19] Petar Vukmirovic, Jasmin Christian Blanchette, Simon Cruanes, and Stephan Schulz. Extending a brainiac prover to lambda-free higher-order logic. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 192–210. Springer, 2019.

[Wan17] Daniel Wand. *Superposition: Types and Induction. (Superposition: types et induction)*. PhD thesis, Saarland University, Saarbrücken, Germany, 2017.

[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, *Proceedings of the 2nd International Workshop Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2016) affiliated with the*

International Joint Conference on Automated Reasoning (IJCAR 2016)., Coimbra, Portugal, July 1, 2016., volume 1770 of *CEUR Workshop Proceedings*, pages 51–65. 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.

A Leo-III Proof of Fig. 5

```
% SZS status Theorem for becker.p
% SZS output start CNFRefutation for becker.p
thf(mworld_type,type,(
    mworld: $tType )).

thf(mrel_type,type,(
    mrel: mworld > mworld > $o )).

thf(meuclidean_type,type,(
    meuclidean: ( mworld > mworld > $o ) > $o )).

thf(meuclidean_def,definition,
    ( meuclidean
    = ( ^ [A: mworld > mworld > $o] :
        ! [B: mworld,C: mworld,D: mworld] :
          ( ( ( A @ B @ C )
            & ( A @ B @ D ) )
         => ( A @ C @ D ) ) ) )).

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(mdia_type,type,(
    mdia: ( mworld > $o ) > mworld > $o )).

thf(mdia_def,definition,
    ( mdia
    = ( ^ [A: mworld > $o,B: mworld] :
        ? [C: mworld] :
          ( ( mrel @ B @ C )
          & ( A @ 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(mexists_const__o__d_i_t__d_i_c__type,type,(
    mexists_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o )
                                    > mworld > $o )).

thf(mexists_const__o__d_i_t__d_i_c__def,definition,
    ( mexists_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,B: mworld] :
        ? [C: $i > $i] :
          ( A @ C @ B ) ) )).

thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__type,type,(
    mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_: ( ( $i > mworld > $o )
                                                    > mworld > $o )
                                                    > mworld > $o )).

thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__def,definition,
    ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
    = ( ^ [A: ( $i > mworld > $o ) > mworld > $o,B: mworld] :
        ! [C: $i > mworld > $o] :
          ( A @ C @ B ) ) )).

thf(mforall_const__o__d_i_c__type,type,(
    mforall_const__o__d_i_c_: ( $i > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_c__def,definition,
    ( mforall_const__o__d_i_c_
    = ( ^ [A: $i > mworld > $o,B: mworld] :
        ! [C: $i] :
          ( A @ C @ B ) ) )).

thf(mforall_const__o__d_i_t__d_i_c__type,type,(
    mforall_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o )
                                    > mworld > $o )).

thf(mforall_const__o__d_i_t__d_i_c__def,definition,
    ( mforall_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,B: mworld] :
        ! [C: $i > $i] :
          ( A @ C @ B ) ) )).

thf(sk1_type,type,(
    sk1: mworld )).

thf(sk2_type,type,(
    sk2: $i > mworld > $o )).

thf(sk3_type,type,(
    sk3: $i > $i )).

thf(sk4_type,type,(
    sk4: $i )).

thf(sk5_type,type,(
    sk5: mworld )).

thf(sk6_type,type,(
    sk6: ( $i > $i ) > mworld )).

thf(1,conjecture,
    ( mvalid
    @ ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
      @ ^ [A: $i > mworld > $o] :
          ( mforall_const__o__d_i_t__d_i_c_
          @ ^ [B: $i > $i] :
              ( mforall_const__o__d_i_c_
              @ ^ [C: $i] :
                  ( mexists_const__o__d_i_t__d_i_c_
                  @ ^ [D: $i > $i] :
                      ( mimplies
                        @ ( mdia @ ( mbox @ ( A @ ( B @ C ) ) ) )
                        @ ( mbox @ ( A @ ( D @ C ) ) ) ) ) ) ) ) ) ),
```

```
    file('becker.p',1)).

thf(2,negated_conjecture,(
    ~ ( mvalid
      @ ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
        @ ^ [A: $i > mworld > $o] :
            ( mforall_const__o__d_i_t__d_i_c_
            @ ^ [B: $i > $i] :
                ( mforall_const__o__d_i_c_
                @ ^ [C: $i] :
                    ( mexists_const__o__d_i_t__d_i_c_
                    @ ^ [D: $i > $i] :
                        ( mimplies
                          @ ( mdia @ ( mbox @ ( A @ ( B @ C ) ) ) )
                          @ ( mbox @ ( A @ ( D @ C ) ) ) ) ) ) ) ) ) ) ),
    inference(neg_conjecture,[status(cth)],[1])).

thf(5,plain,(
    ~ ! [A: mworld,B: $i > mworld > $o,C: $i > $i,D: $i] :
      ? [E: $i > $i] :
        ( ? [F: mworld] :
            ( ( mrel @ A @ F )
            & ! [G: mworld] :
                ( ( mrel @ F @ G )
               => ( B @ ( C @ D ) @ G ) ) )
       => ! [F: mworld] :
            ( ( mrel @ A @ F )
           => ( B @ ( E @ D ) @ F ) ) ) ),
    inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).

thf(6,plain,(
    ~ ! [A: mworld,B: $i > mworld > $o,C: $i > $i,D: $i] :
        ( ? [E: mworld] :
            ( ( mrel @ A @ E )
            & ! [F: mworld] :
                ( ( mrel @ E @ F )
               => ( B @ ( C @ D ) @ F ) ) )
       => ? [E: $i > $i] :
          ! [F: mworld] :
            ( ( mrel @ A @ F )
           => ( B @ ( E @ D ) @ F ) ) ) ),
    inference(miniscope,[status(thm)],[5])).

thf(10,plain,(
    mrel @ sk1 @ sk5 ),
    inference(cnf,[status(esa)],[6])).

thf(4,axiom,(
    meuclidean @ mrel ),
    file('becker.p',mrel_meuclidean)).

thf(15,plain,(
    ! [A: mworld,B: mworld,C: mworld] :
      ( ( ( mrel @ A @ B )
        & ( mrel @ A @ C ) )
     => ( mrel @ B @ C ) ) ),
    inference(defexp_and_simp_and_etaexpand,[status(thm)],[4])).

thf(16,plain,(
    ! [C: mworld,B: mworld,A: mworld] :
      ( ~ ( mrel @ A @ B )
      | ~ ( mrel @ A @ C )
      | ( mrel @ B @ C ) ) ),
    inference(cnf,[status(esa)],[15])).

thf(17,plain,(
    ! [C: mworld,B: mworld,A: mworld] :
      ( ~ ( mrel @ A @ C )
      | ( mrel @ B @ C )
      | ( ( mrel @ sk1 @ sk5 )
       != ( mrel @ A @ B ) ) ) ),
    inference(paramod_ordered,[status(thm)],[10,16])).

thf(18,plain,(
```

```
    ! [A: mworld] :
      ( ~ ( mrel @ sk1 @ A )
      | ( mrel @ sk5 @ A ) ) ),
    inference(pattern_uni,[status(thm)],
      [17:[bind(A,$thf(sk1)),bind(B,$thf(sk5))]])).

thf(40,plain,(
    ! [A: mworld] :
      ( ~ ( mrel @ sk1 @ A )
      | ( mrel @ sk5 @ A ) ) ),
    inference(simp,[status(thm)],[18])).

thf(9,plain,(
    ! [A: mworld] :
      ( ~ ( mrel @ sk5 @ A )
      | ( sk2 @ ( sk3 @ sk4 ) @ A ) ) ),
    inference(cnf,[status(esa)],[6])).

thf(7,plain,(
    ! [A: $i > $i] :
      ~ ( sk2 @ ( A @ sk4 ) @ ( sk6 @ A ) ) ),
    inference(cnf,[status(esa)],[6])).

thf(11,plain,(
    ! [A: $i > $i] :
      ~ ( sk2 @ ( A @ sk4 ) @ ( sk6 @ A ) ) ),
    inference(simp,[status(thm)],[7])).

thf(206,plain,(
    ! [B: $i > $i,A: mworld] :
      ( ~ ( mrel @ sk5 @ A )
      | ( ( sk2 @ ( sk3 @ sk4 ) @ A )
       != ( sk2 @ ( B @ sk4 ) @ ( sk6 @ B ) ) ) ) ),
    inference(paramod_ordered,[status(thm)],[9,11])).

thf(213,plain,(
    ~ ( mrel @ sk5 @ ( sk6 @ sk3 ) ) ),
    inference(pre_uni,[status(thm)],
      [206:[bind(A,$thf(sk6 @ sk3)),bind(B,$thf(sk3))]])).

thf(257,plain,(
    ! [A: mworld] :
      ( ~ ( mrel @ sk1 @ A )
      | ( ( mrel @ sk5 @ A )
       != ( mrel @ sk5 @ ( sk6 @ sk3 ) ) ) ) ),
    inference(paramod_ordered,[status(thm)],[40,213])).

thf(258,plain,(
    ~ ( mrel @ sk1 @ ( sk6 @ sk3 ) ) ),
    inference(pattern_uni,[status(thm)],[257:[bind(A,$thf(sk6 @ sk3))]])).

thf(8,plain,(
    ! [A: $i > $i] : ( mrel @ sk1 @ ( sk6 @ A ) ) ),
    inference(cnf,[status(esa)],[6])).

thf(12,plain,(
    ! [A: $i > $i] : ( mrel @ sk1 @ ( sk6 @ A ) ) ),
    inference(simp,[status(thm)],[8])).

thf(272,plain,( ~ $true ),
    inference(rewrite,[status(thm)],[258,12])).

thf(273,plain,( $false ),
    inference(simp,[status(thm)],[272])).

% SZS output end CNFRefutation for becker.p
```