Higher-Order Automated Theorem Provers

Christoph Benzmüller

Department of Mathematics and Computer Science
Freie Universität Berlin, Germany
c.benzmueller@fu-berlin.de

1 Introduction

The automation of simple type theory, also referred to as classical higher-order logic (HOL), has significantly progressed recently. This paper provides an allowedly slightly biased survey on these developments.

A distinguishing characteristic of HOL is its support for higher-order quantification, that is quantification over predicate and/or function variables. Higher-order quantification was developed first by Frege in his Begriffsschrift [61] and then by Russell in his ramified theory of types [90], which was later simplified by others, including Chwistek and Ramsey [89; 54], Carnap, and finally Church in his simple theory of types [53].

In addition to higher-order quantification, HOL gains expressivity by permitting formulas and logical connectives to occur within terms, which is prohibited in first-order logic. This intertwining of terms and formulas is achieved by using the special primitive type o to denote those simply typed terms that are the formulas of the logic. Additionally, λ-abstractions over formulas allow the explicit naming of sets and predicates, something that is achieved in set theory via the comprehension axioms. Mixing λ-terms and logic as is done in HOL permits capturing many aspects of set theory without direct reference to axioms of set theory. Moreover, the complex rules for quantifier instantiation at higher-types is completely explained via the rules of λ-conversion (the so-called rules of α-, β-, and η-conversion) which were proposed earlier by Church [51; 52].

Church first introduced elementary type theory (ETT), an extension of first-order logic with quantification at all simple types and with the term structure upgraded to be all simply typed λ-terms. He extended this logic into his simple type theory by adding further axioms, including the axioms for extensionality, description, choice, and infinity. Functional extensionality expresses that two functions are equal if and only if they are point-wise equal. Boolean extensionality says that two formulas are equal if and only if they are equivalent (or, alternatively, that there are not more than two truth values). The choice operator selects a member from a non-empty set and description selects a member from a set only if this set is a singleton set. Infinity asserts that a set with infinitely many objects exists.

Initially much of the work on the automation of higher-order logic strongly focused on the automation of elementary type theory. Recently, however, significant progress has also been made for the automation of extensional type theory,

which is elementary type theory extended with functional and Boolean extensionality (and possibly choice or description). In the remainder we use the term HOL synonymous to extensional type theory; in [24] the abbreviation ExTT is used.

Interactive and automated theorem provers for HOL have been employed in a wide range of applications, for example, in mathematics and in hardware and software verification. Moreover, due to its expressivity, HOL is well suited as a meta-logic, and a range of (propositional and quantified) non-classical logics can be elegantly embedded in it. Exploiting this fact, automated theorem provers for HOL have recently been employed to automate reasoning within and about various non-classical logics.

There are several recommended sources providing more details on HOL and its applications [24; 6; 5; 7; 58; 73; 59]; the text presented here is partly based on [24].

1.1 Syntax of HOL

The set T of *simple types* in HOL is usually freely generated from a set of *basic types* $\{o, i\}$ using the function type constructor $\rightarrow$. o denotes the type of Booleans and i some non-empty domain of individuals. Further base types may be added.

Let $\alpha, \beta, o \in T$. The *terms* of HOL are defined by the grammar (c_α denotes typed constants and X_α typed variables distinct from c_α):

$$\begin{aligned} s, t ::= \; & c_\alpha \mid X_\alpha \mid (\lambda X_\alpha s_\beta)_{\alpha\rightarrow\beta} \mid (s_{\alpha\rightarrow\beta}\, t_\alpha)_\beta \mid \\ & (\neg_{o\rightarrow o}\, s_o)_o \mid (s_o \vee_{o\rightarrow o\rightarrow o}\, t_o)_o \mid (\Pi_{(\alpha\rightarrow o)\rightarrow o}\, s_{\alpha\rightarrow o})_o \end{aligned}$$

Complex typed HOL terms are thus constructed via *abstraction* and *application*, and HOL terms of type o are called formulas.

The *primitive logical connectives* (chosen here) are $\neg_{o\rightarrow o}, \vee_{o\rightarrow o\rightarrow o}$ and $\Pi_{(\alpha\rightarrow o)\rightarrow o}$ (for each type α), and, additionally, *choice operators* $\epsilon_{(\alpha\rightarrow o)\rightarrow\alpha}$ (for each type α) or *primitive equality* $=_{\alpha\rightarrow\alpha\rightarrow o}$ (for each type α), abbreviated as $=^\alpha$, may be added. From the selected set of primitive connectives, other logical connectives can be introduced as abbreviations: for example, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, and $\varphi \longleftrightarrow \psi$ abbreviate $\neg(\neg\varphi \vee \neg\psi)$, $\neg\varphi \vee \psi$, and $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$, respectively.

Binder notation $\forall X_\alpha\, s_o$ is used as an abbreviation for $\Pi_{(\alpha\rightarrow o)\rightarrow o}\lambda X_\alpha s_o$. Type information as well as brackets may be omitted if obvious from the context.

Equality can be defined by exploiting Leibniz' principle, expressing that two objects are equal if they share the same properties. *Leibniz equality* $\doteq^\alpha$ at type α is thus defined as $s_\alpha \doteq^\alpha t_\alpha := \forall P_{\alpha\rightarrow o}(\neg Ps \vee Pt)$.

Each occurrence of a variable in a term is either bound by a λ or free. We use $free(s)$ to denote the set of free variables of s (i.e., variables with a free occurrence in s). We consider two terms to be *equal* if the terms are the same up to the names of bound variables (i.e., we consider α-conversion implicitly). A term s is closed if $free(s)$ is empty.

Substitution of a term s_α for a variable X_α in a term t_β is denoted by $[s/X]t$, where it is assumed that the bound variables of t avoid variable capture.

Well known operations and relations on HOL terms include *βη-normalization* and *βη-equality*, denoted by $s =_{\beta\eta} t$, *β-reduction* and *η-reduction*. A *β-redex* $(\lambda X s)t$ β-reduces to $[t/X]s$. An *η-redex* $\lambda X(sX)$ where variable X is not free in s, η-reduces to s. We write $s =_\beta t$ to mean s can be converted to t by a series of β-reductions and expansions. Similarly, $s =_{\beta\eta} t$ means s can be converted to t using both β and η.

For each simply typed λ-term s there is a unique *β-normal form* (denoted $s\downarrow_\beta$) and a unique βη-normal form (denoted $s\downarrow_{\beta\eta}$). From this fact we know $s \equiv_\beta t$ ($s \equiv_{\beta\eta} t$) if and only if $s\downarrow_\beta \equiv t\downarrow_\beta$ ($s\downarrow_{\beta\eta} \equiv t\downarrow_{\beta\eta}$).

Remember, that formulas are defined as terms of type o. A *non-atomic formula* is any formula whose β-normal form is of the form $[c\overline{\mathbf{A}^n}]$ where c is a logical constant. An *atomic formula* is any other formula.

1.2 Semantics of HOL

The following sketch of HOL semantics closely follows [7]; for a more detailed introduction see [18] and the references therein.

A *frame* is a collection $\{D_\alpha\}_{\alpha\in T}$ of nonempty sets called *domains* such that $D_o = \{T, F\}$ where T represents truth and F falsehood, $D_i \neq \emptyset$ is chosen arbitrary, and $D_{\alpha\to\beta}$ are collections of total functions mapping D_α into D_β.

An *interpretation* is a tuple $\langle\{D_\alpha\}_{\alpha\in T}, I\rangle$ where $\{D_\alpha\}_{\alpha\in T}$ is a frame and function I maps each typed constant symbol c_α to an appropriate element of D_α, which is called the *denotation* of c_α. The denotations of $\neg, \vee$ and $\Pi_{(\alpha\to o)\to o}$ (and $\epsilon_{(\alpha\to o)\to\alpha}$ and $=_{\alpha\to\alpha\to o}$) are always chosen as usual. A variable assignment σ maps variables X_α to elements in D_α.

An interpretation is a *Henkin model (general model)* if and only if there is a binary valuation function V such that $V(\sigma, s_\alpha) \in D_\alpha$ for each variable assignment σ and term s_α, and the following conditions are satisfied for all σ, variables X_α, constants c_α, and terms $l_{\alpha\to\beta}, r_\alpha, s_\beta$ (for $\alpha, \beta \in T$): $V(\sigma, X_\alpha) = \sigma(X_\alpha)$, $V(\sigma, c_\alpha) = I(c_\alpha)$, $V(\sigma, l_{\alpha\to\beta}\, r_\alpha) = (V(\sigma, l_{\alpha\to\beta})V(\sigma, r_\alpha))$, and $V(\sigma, \lambda X_\alpha s_\beta)$ represents the function from D_α into D_β whose value for each argument $z \in D_\alpha$ is $V(\sigma[z/X_\alpha], s_\beta)$, where $\sigma[z/X_\alpha]$ is that assignment such that $\sigma[z/X_\alpha](X_\alpha) = z$ and $\sigma[z/X_\alpha]Y_\beta = \sigma Y_\beta$ when $Y_\beta \neq X_\alpha$.

If an interpretation $H = \langle\{D_\alpha\}_{\alpha\in T}, I\rangle$ is a Henkin model, the function V is uniquely determined and $V(\sigma, s_\alpha) \in D_\alpha$ is called the *denotation* of s_α. H is called a *standard model* if and only if for all α and β, $D_{\alpha\to\beta}$ is the set of all functions from D_α into D_β. It is easy to verify that each standard model is also a Henkin model. A formula s of HOL is *valid* in a Henkin model H if and only if $V(\sigma, s) = T$ for all variable assignments σ. In this case we write $H \models^{HOL} s$. Formula s is (Henkin) valid, denoted as $\models^{HOL} s$, if and only if $H \models^{HOL} s$ for all Henkin models H.

While Church axiomatized the logical connectives in a rather conventional fashion (using, for example, negation, conjunction, and universal quantification as the primitive connectives), Henkin [66] and Andrews [2; 6] provided alternative formulations in which the sole logical connective was primitive equality (at all types). Not only was a formulation of logic using just this one logical connective

perspicuous, it also improved on the notion of Henkin models. In fact, as Andrews shows in [2], the sets $\mathcal{D}_{\alpha\to o}$ may be so sparse when using a conventional set of logical connectives that Leibniz equality may denote a relation, which does not fulfill the functional extensionality principle. A solution is to presuppose the presence of the identity relations in all domains $\mathcal{D}_{\alpha\to\alpha\to o}$, which ensures the existence of unit sets $\{a\} \in \mathcal{D}_{\alpha\to o}$ for all elements $a \in \mathcal{D}_\alpha$. The existence of these unit sets in turn ensures that Leibniz equality indeed denotes the intended (fully extensional) identity relation. Syntactically, the existence of these sets can be enforced by working with primitive equality.

Modulo the above observation, Henkin models are *fully extensional*, that is, they validate the functional and Boolean extensionality principles. The construction of non-functional models for elementary type theory has been pioneered by Andrews [1]. In Andrews' so-called v-complexes, which are based on Schütte's semi-valuation method [91], both the functional and the Boolean extensionality principles fail. Assuming β-equality, functional extensionality splits into two weaker and independent principles η ($f \doteq \lambda X f X$, if X is not free in term f) and ξ (from $\forall X(f \doteq g)$ infer $(\lambda X f) \doteq (\lambda X g)$, where X may occur free in f and g). Conversely, $\beta\eta$-conversion, which is built-in in many modern implementations of HOL, together with ξ implies functional extensionality. Boolean extensionality, however, is independent of any of these principles. A whole landscape of respective notions of models structures for HOL between Andrews' v-complexes and Henkin semantics that further illustrate and clarify the above connections is developed in [9; 18; 45], and an alternative development and discussion has been contributed in [80].

2 Proof Systems and Proof-Theoretical Properties

2.1 Cut-free Sequent Calculi

Cut-free sequent calculi for elementary type theory and fragments of it have been studied by Takeuti [103], Schütte [91], Tait [101], Takahashi [102], Prawitz [88], and Girard [63]. Andrews [1] used the *abstract consistency principle* of Smullyan [92] in order to give a proof of the completeness of resolution in elementary type theory. Takeuti [105] presented a cut-free sequent calculus with extensionality that is complete for Henkin models. The abstract consistency proof technique, as used by Andrews, has been further extended and applied in [71; 9; 45; 18; 19; 20; 48] to obtain cut-elimination results for different systems between elementary type theory and HOL.

We here present the cut-free, sound and complete, one-sided sequent calculi for HOL (without choice) from [20]. In the context of this work, a sequent is a finite set Δ of β-normal closed formulas. A sequent calculus $\mathcal{G}$ provides an inductive definition for when $\vdash^{\mathcal{G}} \Delta$ holds. A sequent calculus rule

$$\frac{\Delta_1 \quad \cdots \quad \Delta_n}{\Delta}\, r$$

is *admissible* in $\mathcal{G}$ if $\vdash^{\mathcal{G}} \Delta$ holds whenever $\vdash^{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

Definition 1 (Sequent calculi $\mathcal{G}_\beta$ and $\mathcal{G}_{\beta\mathfrak{fb}}$). *Let Δ and Δ' be finite sets of β-normal closed formulas of HOL and let Δ, s denote the set $\Delta \cup \{s\}$. The following sequent calculus rules are introduced:*

Basic Rules

$$\frac{\Delta, s}{\Delta, \neg\neg s}\,\mathcal{G}(\neg) \qquad \frac{\Delta, \neg s \quad \Delta, \neg t}{\Delta, \neg(s \vee t)}\,\mathcal{G}(\vee_-) \qquad \frac{\Delta, s, t}{\Delta, (s \vee t)}\,\mathcal{G}(\vee_+)$$

$$\frac{\Delta, \neg\,(sl)\big\downarrow_\beta \quad l_\alpha \text{ closed term}}{\Delta, \neg\Pi^\alpha s}\,\mathcal{G}(\Pi^l_-) \qquad \frac{\Delta, (sc)\big\downarrow_\beta \quad c_\delta \text{ new symbol}}{\Delta, \Pi^\alpha s}\,\mathcal{G}(\Pi^c_+)$$

Initialization

$$\frac{s \text{ atomic (and } \beta\text{-normal)}}{\Delta, s, \neg s}\,\mathcal{G}(init) \qquad \frac{\Delta, (s \doteq^o t) \quad s,t \text{ atomic}}{\Delta, \neg s, t}\,\mathcal{G}(Init^{\doteq})$$

Extensionality

$$\frac{\Delta, (\forall X_\alpha sX \doteq^\beta tX)\big\downarrow_\beta}{\Delta, (s \doteq^{\alpha\to\beta} t)}\,\mathcal{G}(\mathfrak{f}) \qquad \frac{\Delta, \neg s, t \quad \Delta, \neg t, s}{\Delta, (s \doteq^o t)}\,\mathcal{G}(\mathfrak{b})$$

Decomposition

$$\frac{\Delta, (s^1 \doteq^{\alpha_1} t^1) \;\cdots\; \Delta, (s^n \doteq^{\alpha_n} t^n) \quad \begin{array}{c} n \geq 1, \beta \in \{o, \iota\}, \\ h_{\overline{\alpha^n}\to\beta} \in \Sigma \end{array}}{\Delta, (h\overline{s^n} \doteq^\beta h\overline{t^n})}\,\mathcal{G}(d)$$

Sequent calculus $\mathcal{G}_\beta$ is defined by the rules $\mathcal{G}(init)$, $\mathcal{G}(\neg)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi^l_-)$ and $\mathcal{G}(\Pi^c_+)$. Sequent calculus $\mathcal{G}_{\beta\mathfrak{fb}}$ extends $\mathcal{G}_\beta$ by the additional rules $\mathcal{G}(\mathfrak{b})$, $\mathcal{G}(\mathfrak{f})$, $\mathcal{G}(d)$, and $\mathcal{G}(Init^{\doteq})$.

Theorem proving in these calculi works as follows: In order to prove that a (closed) conjecture formula c logically follows from a (possibly empty) set of (closed) axioms $\{a^1, \ldots, a^n\}$, we start from the initial sequent $\Delta := \{c, \neg a^1, \ldots, \neg a^n\}$ and reason backwards by applying the respective calculus rules. We are done, if all branches of the proof tree can be closed by an application of the $\mathcal{G}(init)$ rule. In this case $\vdash^{\mathcal{G}_\beta/\mathcal{G}_{\beta\mathfrak{fb}}} \Delta := \{c, \neg a^1, \ldots, \neg a^n\}$ holds, which means that the conjecture c logically follows from the axioms $a^1, \ldots, a^n$ within calculus $\mathcal{G}_\beta$, respectively $\mathcal{G}_{\beta\mathfrak{fb}}$.

Soundness and completeness results for $\mathcal{G}_\beta$ and $\mathcal{G}_{\beta\mathfrak{fb}}$ have been established in [20].

Theorem 1 (Soundness and Completeness).

1. *$\mathcal{G}_\beta$ is sound and complete for ETT:* $\models^{ETT} c$ *if and only if* $\vdash^{\mathcal{G}_\beta} \{c\}$
   ($\mathcal{G}_\beta$ is thus also sound for HOL).

2. $\mathcal{G}_{\beta\mathfrak{fb}}$ is sound and complete for HOL: $\models^{HOL} c$ *if and only if* $\vdash^{\mathcal{G}_{\beta\mathfrak{fb}}} \{c\}$

Similarly, $\{a^1, \ldots, a^n\} \models^{ETT/HOL} c$ *if and only if* $\vdash^{\mathcal{G}_\beta/\mathcal{G}_{\beta\mathfrak{fb}}} \{c, \neg a^1, \ldots, \neg a^n\}$.

Rule $\mathcal{G}(cut)$

$$\frac{\Delta, s \quad \Delta, \neg s}{\Delta}\,\mathcal{G}(cut)$$

is available neither in $\mathcal{G}_\beta$ nor in $\mathcal{G}_{\beta\mathfrak{fb}}$, and cut-elimination holds for both calculi [20].

Theorem 2 (Cut-elimination). *The rule $\mathcal{G}(cut)$ is admissible in $\mathcal{G}_\beta$ and $\mathcal{G}_{\beta\mathfrak{fb}}$.*

In spite of their cut-freeness, both calculi are obviously only mildly suited for automation. One reason is that they are blindly guessing instantiations l in rule $\mathcal{G}(\Pi^l_-)$. Another reason is that the treatment of equality in both calculi relies on Leibniz equality $\doteq$. Support for primitive equality is not provided. The problem with Leibniz equality (or other forms of defined equality) is that it threatens cut-freeness of the calculi by allowing for simulations (admissibility) of the cut rule. The problem of cut-simulation, which is problematic for effective proof automation, analogously applies to a wide range of other prominent HOL axioms. We therefore address this issue in more depth within the next subsection. In §4 we will then outline a proof procedure that is better suited for proof automation than sequent calculi $\mathcal{G}_\beta$ and $\mathcal{G}_{\beta\mathfrak{fb}}$ above.

2.2 Cut-Simulation

We illustrate why Leibniz equality implies cut-simulation. Assume we want to study in $\mathcal{G}_\beta$ or $\mathcal{G}_{\beta\mathfrak{fb}}$ whether a conjecture c logically follows from an equality axiom $l = r$ (where l and r are some arbitrary closed terms of type α). Since primitive equality is not available we formalize the axiom as $l \doteq^\alpha r$ and initialize the proof process with sequent $\Delta := \{c, \neg(l \doteq^\alpha r)\}$, that is, with $\Delta := \{c, \neg\Pi(\lambda P_{\alpha\to o}(\neg Pl \vee Pr))$.

Now consider the following derivation, where s is an arbitrary (cut) formula:

$$\dfrac{\dfrac{\dfrac{\Delta, s}{\Delta, \neg\neg s}\,\mathcal{G}(\neg) \qquad \Delta, \neg s}{\Delta, \neg(\neg s \vee s)}\,\mathcal{G}(\vee_-)}{\Delta, \neg\Pi(\lambda P_{\alpha\to o}(\neg Pl \vee Pr))}\,\mathcal{G}(\Pi^{\lambda Xs}_-)$$

It is easy to see that this derivation introduces a cut on formula s; in the left branch s occurs positively and in the right branch negatively.

Cut-simulation is also enabled by the functional and Boolean extensionality axioms. The Boolean extensionality axiom (abbreviated as $\mathcal{B}_o$) is given as

$$\forall A_o \forall B_o (A \longleftrightarrow B) \to A \doteq^o B$$

The infinitely many functional extensionality axioms (abbreviated as $\mathcal{F}_{\alpha\beta}$) are parameterized over $\alpha, \beta \in T$. They are given as

$$\forall F_{\alpha\to\beta} \forall G_{\alpha\to\beta} (\forall X_\alpha FX \doteq^\beta GX) \to F \doteq^{\alpha\to\beta} G$$

Instead of the extensionality rules $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\mathfrak{b})$, as provided in calculus $\mathcal{G}_{\beta\mathfrak{f}\mathfrak{b}}$, we could alternatively postulate the validity of these axioms. For this we could replace the rules $\mathcal{G}(\mathfrak{f})$ and $\mathcal{G}(\mathfrak{b})$ in $\mathcal{G}_\beta$ by the following axiomatic extensionality rules $\mathcal{G}(\mathcal{F}_{\alpha\beta})$ and $\mathcal{G}(\mathcal{B})$:

$$\frac{\Delta, \neg\mathcal{F}_{\alpha\beta} \quad \alpha \to \beta \in T}{\Delta}\,\mathcal{G}(\mathcal{F}_{\alpha\beta}) \qquad \frac{\Delta, \neg\mathcal{B}_o}{\Delta}\,\mathcal{G}(\mathcal{B})$$

This calculus is still Henkin complete (even if rules $\mathcal{G}(d)$ and $\mathcal{G}(Init^{\doteq})$ are additionally removed) [20]. However, the modified calculus suffers severely from cut-simulation. For axiom $\mathcal{B}_o$ this is illustrated by the following derivation (a_o is new constant symbol):

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{derivable in 7 steps}}{\Delta, a \longleftrightarrow a}}{\Delta, \neg\neg(a \longleftrightarrow a)}\,\mathcal{G}(\neg) \qquad \begin{array}{c}\Delta, s \quad \Delta, \neg s \\ \vdots \text{ derivable in 3 steps, see above} \\ \Delta, \neg(a \doteq^o a)\end{array}}{\Delta, \neg(\neg(a \longleftrightarrow a) \vee a \doteq^o a)}\,\mathcal{G}(\vee_-)}{\Delta, \neg\mathcal{B}_o}\,2\times\mathcal{G}(\Pi^a_-)$$

The left branch is closed and on the right branch an arbitrary cut formula s is introduced. A similar derivation is enabled with axiom $\mathcal{F}_{\alpha\beta}$ (b_α is new constant symbol):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{derivable in 3 steps}}{\Delta, fb \doteq^\beta fb}}{\Delta, (\forall X_\alpha fX \doteq^\beta fX)}\,\mathcal{G}(\Pi^b_+) \qquad \begin{array}{c}\Delta, s \quad \Delta, \neg s \\ \vdots \text{ derivable in 3 steps} \\ \Delta, \neg(f \doteq^{\alpha\to\beta} f)\end{array}}{\Delta, \neg\forall X_\alpha fX \doteq^\beta fX}\,\mathcal{G}(\neg)}{\Delta, \neg(\neg(\forall X_\alpha fX \doteq^\beta fX) \vee f \doteq^{\alpha\to\beta} f)}\,\mathcal{G}(\vee_-)}{\Delta, \neg\mathcal{F}_{\alpha\beta}}\,2\times\mathcal{G}(\Pi^f_-)$$

In all cut-simulations above we have exploited the fact that predicate variables may be instantiated with terms that introduce arbitrary new formulas s. At these points the subformula property breaks. At the same time this offers the opportunity to mimic cut-introductions by appropriately selecting such instantiations for predicate variables. In addition to Leibniz equations and the Boolean and functional extensionality axioms, cut-simulations are analogously enabled by many prominent other axioms, including excluded middle, description, choice,

comprehension, and induction. We may thus call these axioms cut-strong. More details on such cut-strong axioms are provided in [20].[1]

Cut-simulations have in fact been extensively used in literature. For example, Takeuti showed that a conjecture of Gödel could be proved without cut by using the induction principle instead [104]; [75] illustrates how the induction rule can be used to hide the cut rule; and [91] used excluded middle to similarly mask the cut rule.

For the development of automated proof procedures for HOL we thus learn an important lesson, namely that cut-elimination and cut-simulation should always be considered in combination: a pure cut-elimination result may indeed mean little if at the same time axioms are assumed that support effective cut-simulation. The challenge is to develop cut-free calculi for HOL that also try to avoid the pitfall of cut-simulations (as far as possible).

Church's use of the λ-calculus to build comprehension principles into the language can therefore be seen as a first step in the program to eliminate cut-strong axioms. Significant progress in the automation of HOL in existing prover implementations has been achieved after providing calculus level support for extensionality and also choice (avoiding cut-simulation effects). Respective extensionality rules have been provided for resolution [9; 10], expansion and sequent calculi [45; 46], and tableaux [48]. Similarly, choice rules have been proposed for the various settings: sequent calculus [78], tableaux [8] and resolution [38].

In §4 we outline the extensional RUE-resolution approach of the Leo-II theorem prover [39; 38]. In this approach some pragmatic improvements are offered regarding most pressing challenges for effective proof automation.

3 Proof-Theoretical Properties via Semantic Embeddings

Cut-elimination and cut-simulation in HOL have been addressed in the previous section. In particular, with sequent calculus $\mathcal{G}_{\beta\mathfrak{fb}}$ an example of a cut-free calculus for HOL has been provided.

The development of cut-free calculi for expressive logics is generally a non-trivial task. By modeling and studying these logics as fragments of HOL — a research direction proposed in [16] — existing results for HOL (for example, the cut-free sequent calculus $\mathcal{G}_{\beta\mathfrak{fb}}$) can easily be reused. The idea is illustrated next by choosing quantified conditional logics [95] as an example.

3.1 Quantified Conditional Logic

As an exemplary challenging logic we consider quantified conditional logic (QCL).

[1] Obviously, any universally quantified predicate variable (occuring negatively in the above approach) is a possible source for cut-simulation. The challenge thus is to avoid those predicate variables as far as possible. An axiomatic approach based on cut-strong axioms, as proposed by several authors including e.g. [68; 69], is therefore hardly a suitable option for the automation of HOL.

Let IV be a set of first-order (individual) variables, PV a set of propositional variables, and SYM a set of predicate symbols of any arity. Formulas of QCL are given by the following grammar (where $X^i \in \text{IV}, P \in \text{PV}, k \in \text{SYM}$, and where $\Rightarrow$ represents conditionality):

$$\varphi, \psi ::= \quad P \mid k(X^1, \ldots, X^n) \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid \forall^{co} X\varphi \mid \forall^{va} X\varphi \mid \forall P\varphi$$

From the selected set of primitive connectives, other logical connectives can be introduced as abbreviations: for example, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ (material implication), $\varphi \longleftrightarrow \psi$ and $\Box\varphi$ abbreviate $\neg(\neg\varphi \vee \neg\psi)$, $\neg\varphi \vee \psi$, $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$ and $\neg\varphi \Rightarrow \varphi$, respectively. $\forall^{co}$ and $\forall^{va}$ are associated with constant domain and variable domain quantification. For $* \in \{co, va\}$, $\exists^* X\varphi$ abbreviates $\neg\forall^* X\neg\varphi$. Syntactically, QCL can be seen as a generalization of quantified multimodal logic where the index of modality $\Rightarrow$ is a formula of the same language. For instance, in $(\varphi \Rightarrow \psi) \Rightarrow \delta$ the subformula $\varphi \Rightarrow \psi$ is the index of the second occurrence of $\Rightarrow$.

Regarding semantics, different formalizations have been proposed (see [83]). Here we build on *selection function semantics* [95; 49], which is based on possible world structures and has been successfully used in [85] to develop proof methods for some propositional CLs.

An *interpretation* is a structure $M = \langle S, f, D, D', Q, I\rangle$ where, S is a set of items called possible worlds, $f : S \times 2^S \mapsto 2^S$ is the selection function, D is a non-empty set of *individuals* (the constant first-order domain), D' is a function that assigns a non-empty subset $D'(w)$ of D to each possible world w (the $D'(w)$ are the varying domains), Q is a non-empty collection of subsets of S (the propositional domain), and I is a classical interpretation function where for each n-ary predicate symbol k, $I(k, w) \subseteq D^n$.

A *variable assignment* $g = (g^i, g^p)$ is a pair of maps where, $g^i : \text{IV} \mapsto D$ maps each individual variable in IV to an object in D, and g^p : maps each propositional variable in PV to a set of worlds in Q.

Satisfiability of a formula φ for an interpretation $M = \langle S, f, D, D', Q, I\rangle$, a world $s \in S$, and a variable assignment $g = (g^i, g^p)$ is denoted as $M, g, s \models \varphi$ and defined as follows, where $[a/Z]g$ denote the assignment identical to g except that $([a/Z]g)(Z) = a$:

$M, g, s \models k(X^1, \ldots, X^n)$ if and only if $\langle g^i(X^1), \ldots, g^i(X^n)\rangle \in I(k, w)$

$M, g, s \models P$ if and only if $s \in g^p(P)$

$M, g, s \models \neg\varphi$ if and only if $M, g, s \not\models \varphi$ (that is, not $M, g, s \models \varphi$)

$M, g, s \models \varphi \vee \psi$ if and only if $M, g, s \models \varphi$ or $M, g, s \models \psi$

$M, g, s \models \forall^{co} X\varphi$ if and only if $M, ([d/X]g^i, g^p), s \models \varphi$ for all $d \in D$

$M, g, s \models \forall^{va} X\varphi$ if and only if $M, ([d/X]g^i, g^p), s \models \varphi$ for all $d \in D'(s)$

$M, g, s \models \forall P\varphi$ if and only if $M, (g^i, [p/P]g^p), s \models \varphi$ for all $p \in Q$

$M, g, s \models \varphi \Rightarrow \psi$ if and only if $M, g, t \models \psi$ for all $t \in S$ such that $t \in f(s, [\varphi])$
where $[\varphi] = \{u \mid M, g, u \models \varphi\}$

An interpretation $M = \langle S, f, D, D', Q, I\rangle$ is a QCL *model* if for every variable assignment g and every formula φ, the set of worlds $\{s \in S \mid M, g, s \models \varphi\}$ is a member of Q. (This requirement, which is inspired by Fitting [60], Def. 3.5, ensures a natural correspondence to Henkin models in HOL.) As usual, a conditional formula φ is *valid in a QCL model* $M = \langle S, f, D, D', Q, I\rangle$, denoted with $M \models \varphi$, if and only if for all worlds $s \in S$ and variable assignments g holds $M, g, s \models \varphi$. A formula φ is *valid*, denoted $\models \varphi$, if and only if it is valid in every QCL model.

f is defined to take $[\varphi]$ (called the *proof set* of φ with respect to a given QCL model M) instead of φ. This approach has the consequence of forcing the so-called *normality* property: given a QCL model M, if φ and φ' are equivalent (i.e., they are satisfied in the same set of worlds), then they index the same formulas with respect to the $\Rightarrow$ modality. The axiomatic counterpart of the normality condition is given by the rule RCEA (which expresses a replacement property for equivalent formulas on the left-hand side of a conditional formula):

$$\frac{\varphi \leftrightarrow \varphi'}{(\varphi \Rightarrow \psi) \leftrightarrow (\varphi' \Rightarrow \psi)}\,(RCEA)$$

Moreover, it can be easily shown that the above semantics forces also the following rules to hold (RCEC expresses a right-hand side replacement property analogous to RCEA, and RCK expresses compatibility of the right-hand side of conditional formulas with conjunction):

$$\frac{\varphi \leftrightarrow \varphi'}{(\psi \Rightarrow \varphi) \leftrightarrow (\psi \Rightarrow \varphi')}\,(RCEC)$$

$$\frac{(\varphi_1 \wedge \ldots \wedge \varphi_n) \leftrightarrow \psi}{(\varphi_0 \Rightarrow \varphi_1 \wedge \ldots \wedge \varphi_0 \Rightarrow \varphi_n) \rightarrow (\varphi_0 \Rightarrow \psi)}\,(RCK)$$

We refer to QCK (cf. CK in [50]) as the minimal QCL closed under rules RCEA, RCEC and RCK. In what follows, only QCLs extending QCK are considered.

QCLs have many applications, including action planning, counterfactual reasoning, default reasoning, deontic reasoning, metaphysical modeling and reasoning about knowledge. While there is broad literature on propositional conditional logics only a few authors have addressed first-order extensions [56; 62]. Most interestingly, QCLs subsume normal modal logics ($\Box\varphi$ can be defined as $\neg\varphi \Rightarrow \varphi$, see [95]).

Modeling QCLs as fragments of HOL. Regarding the particular choice of HOL, we here assume a set of basic types $\{o, i, u\}$, where o denotes the type of Booleans as before. Without loss of generality, i is now identified with a (non-empty) set of worlds and u with a (non-empty) domain of individuals.

QCL formulas are now identified with certain HOL terms (predicates) of type $i \rightarrow o$. They can be applied to terms of type i, which are assumed to denote possible worlds. Type $i \rightarrow o$ is abbreviated as τ in the remainder.

The mapping $\lfloor\cdot\rfloor$ translates QCL formulas φ into HOL terms $\lfloor\varphi\rfloor$ of type τ. The mapping is recursively defined:

$$\begin{array}{ll}
\lfloor P\rfloor & = P_\tau \\
\lfloor k(X^1,\ldots,X^n)\rfloor & = k_{u^n\to\tau}\, X^1_u \ldots X^n_u \\
\lfloor\neg\varphi\rfloor & = \neg_\tau \lfloor\varphi\rfloor \\
\lfloor\varphi\vee\psi\rfloor & = \vee_{\tau\to\tau\to\tau} \lfloor\varphi\rfloor\lfloor\psi\rfloor \\
\lfloor\varphi\Rightarrow\psi\rfloor & = \Rightarrow_{\tau\to\tau\to\tau} \lfloor\varphi\rfloor\lfloor\psi\rfloor \\
\lfloor\forall^{co}X\varphi\rfloor & = \Pi^{co}_{(u\to\tau)\to\tau}\, \lambda X_u \lfloor\varphi\rfloor \\
\lfloor\forall^{va}X\varphi\rfloor & = \Pi^{va}_{(u\to\tau)\to\tau}\, \lambda X_u \lfloor\varphi\rfloor \\
\lfloor\forall P\varphi\rfloor & = \Pi_{(\tau\to\tau)\to\tau}\, \lambda P_\tau \lfloor\varphi\rfloor
\end{array}$$

P_τ and $X^1_u,\ldots,X^n_u$ are variables and $k_{u^n\to\tau}$ is a constant symbol. $\neg_\tau$, $\vee_{\tau\to\tau\to\tau}$, $\Rightarrow_{\tau\to\tau\to\tau}$, $\Pi^{co,va}_{(u\to\tau)\to\tau}$ and $\Pi_{(\tau\to\tau)\to\tau}$ realize the QCL connectives in HOL. They abbreviate the following HOL terms:[2]

$$\begin{array}{ll}
\neg_{\tau\to\tau} & = \lambda A_\tau \lambda X_i \neg(A\, X) \\
\vee_{\tau\to\tau\to\tau} & = \lambda A_\tau \lambda B_\tau \lambda X_i (A\, X \vee B\, X) \\
\Rightarrow_{\tau\to\tau\to\tau} & = \lambda A_\tau \lambda B_\tau \lambda X_i \forall V_i (f\, X\, A\, V \to B\, V) \\
\Pi^{co}_{(u\to\tau)\to\tau} & = \lambda Q_{u\to\tau} \lambda V_i \forall X_u (Q\, X\, V) \\
\Pi^{va}_{(u\to\tau)\to\tau} & = \lambda Q_{u\to\tau} \lambda V_i \forall X_u (eiw\, V\, X \to Q\, X\, V) \\
\Pi_{(\tau\to\tau)\to\tau} & = \lambda R_{\tau\to\tau} \lambda V_i \forall P_\tau (R\, P\, V)
\end{array}$$

Constant symbol f in the mapping of $\Rightarrow$ is of type $i\to\tau\to\tau$. It realizes the selection function. Constant symbol eiw (for 'exists in world'), which is of type $(\tau\to u)\to\tau$, is associated with the varying domains. The interpretations of f and eiw are chosen appropriately below. Moreover, for the varying domains a non-emptiness axiom is postulated:

$$\forall W_i \exists X_u (eiw\, W\, X) \tag{NE}$$

Analyzing the validity of a translated formula $\lfloor\varphi\rfloor$ for a world represented by term t_i corresponds to evaluating the application $(\lfloor\varphi\rfloor\, t_i)$. In line with [21] (and analogous to [33; 34; 35]), we define $\mathrm{vld}_{\tau\to o} = \lambda A_\tau \forall S_i (A\, S)$. With this definition, validity of a QCL formula φ in QCK corresponds to the validity of $(\mathrm{vld}\lfloor\varphi\rfloor)$ in HOL, and vice versa.

Soundness and completeness of this embedding of QCL in HOL has been studied in [15].

Theorem 3 (Soundness and Completeness of QCL-embedding). *$\models^{QCL}\varphi$ if and only if $\{NE\}\models^{HOL} vld\lfloor\varphi\rfloor$*

Combining Theorem 3 with Theorem 1 we obtain:

[2] Note the predicate argument A of f in the term for $\Rightarrow_{\tau\to\tau\to\tau}$ and the second-order quantifier $\forall P_\tau$ in the term for $\Pi_{(\tau\to\tau)\to\tau}$. FOL encodings of both constructs, if feasible, will be less natural.

Theorem 4 (Soundness and Completeness of $\mathcal{G}_{\beta\mathfrak{fb}}$ for QCL).
$\models^{QCL} \varphi$ *if and only if* $\vdash^{\mathcal{G}_{\beta\mathfrak{fb}}} \{vld\lfloor\varphi\rfloor, \neg NE\}$

Since $\mathcal{G}_{\beta\mathfrak{fb}}$ is cut-free (Theorem 2), we thus obtain a cut-elimination result for QCL for free. However, we need to point again to the subtle issue of cut-simulation. For example, when postulating additional axioms for the embedded logics in HOL (for example, QCL axiom ID: $\forall\varphi(\varphi \Rightarrow \varphi)$), cut-simulation effects may apply. In some cases the semantical conditions which correspond to such axioms can be postulated instead in order to circumvent the effect. This is for example possible for many prominent modal logic axioms. for example, the corresponding semantical condition for modal axiom T: $\forall\varphi(\Box\varphi \supset \varphi)$ is $\forall x(rxx)$ (where constant r denotes the associated accessibility relation). The latter axiom obviously does not support cut-simulations and should therefore be preferred. However, the semantical condition that corresponds to ID, $\forall A_\tau \forall W_i(f\,W\,A \subseteq A)$, unfortunately still introduces some problematic predicate variables.

3.2 Other Logic Embeddings in HOL

Recent work has shown that many other challenging logics can be characterized as HOL fragments via semantic embeddings. The logics studied so far comprise prominent non-classical logics, including modal logics, tense logics, intuitionistic logic, security logics, conditional logics, hybrid logics and logics for time and space [33; 34; 35; 12; 14; 22; 21; 32; 107; 108]. These fragments also comprise first-order and even higher-order extensions of non-classical logics, for which only little practical automation support has been available so far. Most importantly, however, combinations of embedded logics can be elegantly achieved in this approach. And, similar to above, cut-elimination results for these embedded logics can be obtained 'for free' by exploiting the results already achieved for HOL.

The embeddings approach bridges between the Tarski view of logics (for 'meta logic' HOL) and the Kripke view (for the embedded source logics) and exploits the fact that well known translations of logics, such as the relational translation [84], can be easily formalized in HOL. This way HOL-ATP systems can be uniformly applied to reason *within* and also *about* embedded logics and their combinations.

4 Pragmatic Properties

In this Section we outline the calculus and working principles of the higher-order automated theorem prover Leo-II [39; 38].

4.1 Extensional RUE-Resolution Calculus of Leo-II

Leo-II is an automated theorem prover for HOL. It supports primitive equality and choice.

In Leo-II, logical consequence of a conjecture c from a (possibly empty) set of axioms $\{a^1, \ldots, a^n\}$ is established by refuting the initial set of (non-normalized)

unit clauses $\{[c]^{\mathtt{ff}}, [a^1]^{\mathtt{tt}}, \ldots, [a^n]^{\mathtt{tt}}\}$. Refuting means deriving the empty clause from this initial set by subsequent forward applications of the rules presented below. These rules operate on the clauses of a given clause set and they add their result clauses to this clause set. The superscripts $\mathtt{tt}$ and $\mathtt{ff}$ denote the polarity of clause literals. Clauses are generally depicted as $C \vee [s]^\alpha$ below, where $[s]^\alpha$ is a literal (for $\alpha \in \{\mathtt{tt}, \mathtt{ff}\}$) and where C is a clause rest.

LEO-II's calculus is based on an adaption of the RUE-resolution approach [57], originally developed for first-order logic with equality, to HOL [9; 10]. In RUE-resolution unification constraints are explicitly represented and manipulated as disagreement pairs, that is, negated equations. A unification constraint $[l = r]^{\mathtt{ff}}$ in clause $C \vee [l = r]^{\mathtt{ff}}$ can also be seen as a condition (obligation to make terms l and r equal) under which the clause rest C follows. In LEO-II, such unification constraints are amenable to resolution and factorization.

The calculus rules of LEO-II are presented next.

Definition 2 (Extensional Higher-order RUE-Resolution). *The following rules implicitly assume symmetry and associativity of the clause-level $\vee$-operator. Moreover, they assume that the formulas in clauses are always kept in $\beta\eta$-normal form.*

Basic Rules

$$\frac{C \vee [\neg s]^{\mathtt{tt}}}{C \vee [s]^{\mathtt{ff}}}\,\mathcal{R}(\neg_{\mathtt{tt}}) \qquad \frac{C \vee [\neg s]^{\mathtt{ff}}}{C \vee [s]^{\mathtt{tt}}}\,\mathcal{R}(\neg_{\mathtt{ff}})$$

$$\frac{C \vee [s \vee t]^{\mathtt{tt}}}{C \vee [s]^{\mathtt{tt}} \vee [t]^{\mathtt{tt}}}\,\mathcal{R}(\vee_{\mathtt{tt}}) \qquad \frac{C \vee [\Pi^\alpha s]^{\mathtt{tt}} \quad X_\alpha \textit{ fresh variable}}{C \vee [s\, X]^{\mathtt{tt}}}\,\mathcal{R}(\Pi^X_{\mathtt{ff}})$$

$$\frac{C \vee [s \vee t]^{\mathtt{ff}}}{C \vee [s]^{\mathtt{ff}} \quad C \vee [t]^{\mathtt{ff}}}\,\mathcal{R}(\vee_{\mathtt{ff}}) \qquad \frac{C \vee [\Pi^\alpha s]^{\mathtt{ff}} \quad \mathtt{sk}_\alpha \textit{ Skolem term}}{C \vee [s\, \mathtt{sk}_\alpha]^{\mathtt{ff}}}\,\mathcal{R}(\Pi^{\mathtt{sk}}_{\mathtt{tt}})$$

These rules deal with the normalization of clauses. The rules are straightforward – for instance, $\mathcal{R}(\vee_{\mathtt{tt}})$ lifts object-level disjunction to meta-level (i.e. clause-level) disjunction. Similarly, the rule $\mathcal{R}(\neg_{\mathtt{ff}})$ removes a dominant negation from a literal and flips the literal's polarity. Normalization cannot be treated as a pre-process as in first-order logic, since instantiations of predicate variables, for example with rules $\mathcal{R}(Subst)$ and $\mathcal{R}(PrimSubst)$ may introduce non-normal clauses and require subsequent clause normalization steps. Instead of blindly guessing instantiations as in sequent rule $\mathcal{G}(\Pi^l_-)$, LEO-II thus introduces free variables in the search space. The hope is that suitable instantiations for these variables can be determined by pre-unification within the subsequent proof search process. However, as we will explore further below, this idea can only be partly realized in HOL.

Resolution

$$\frac{C \vee [s]^{\mathtt{tt}} \quad D \vee [t]^{\mathtt{ff}}}{C \vee D \vee [s = t]^{\mathtt{ff}}}\,\mathcal{R}(Res) \qquad \frac{C \vee [s]^p \vee [t]^p}{C \vee [s]^p \vee [s = t]^{\mathtt{ff}}}\,\mathcal{R}(Fac)$$

In LEO-II, resolution and factorization are applied only to proper clauses, that is, clauses in which all literal formulas are atomic (or unification constraints).

Note that both rules introduce unification constraints $[s = t]^{\mathtt{ff}}$. *On these unification constraints the pre-unification rules given below operate. Instead of the factorization rule as shown here, factorization in* LEO-*II is (for pragmatic reasons and at the cost of completeness) restricted to binary clauses only.*

Extensionality

$$\frac{C \vee [s =^{\alpha\to\beta} t]^{\mathtt{tt}} \quad X_\alpha \textit{ fresh variable}}{C \vee [sX =^{\beta} tX]^{\mathtt{tt}}}\,\mathcal{R}(\mathfrak{f}_{\mathtt{tt}}) \qquad \frac{C \vee [s =^{o} t]^{\mathtt{tt}}}{C \vee [s \longleftrightarrow t]^{\mathtt{tt}}}\,\mathcal{R}(\mathfrak{b}_{\mathtt{tt}})$$

$$\frac{C \vee [s =^{\alpha\to\beta} t]^{\mathtt{ff}} \quad \mathtt{sk}_\alpha \textit{ Skolem term}}{C \vee [s\,\mathtt{sk} =^{\beta} t\,\mathtt{sk}]^{\mathtt{ff}}}\,\mathcal{R}(\mathfrak{f}_{\mathtt{ff}}) \qquad \frac{C \vee [s =^{o} t]^{\mathtt{ff}}}{C \vee [s \longleftrightarrow t]^{\mathtt{ff}}}\,\mathcal{R}(\mathfrak{b}_{\mathtt{ff}})$$

Conceptually the extensionality rules $\mathcal{R}(\mathfrak{f}_{\mathtt{tt}})$ *and* $\mathcal{R}(\mathfrak{b}_{\mathtt{tt}})$ *belong to the normalization rules of* LEO-*II, while* $\mathcal{R}(\mathfrak{f}_{\mathtt{ff}})$ *and* $\mathcal{R}(\mathfrak{b}_{\mathtt{ff}})$ *are integrated with the pre-unification rules. Similar to sequent rules* $\mathcal{G}(\mathfrak{f})$ *and* $\mathcal{G}(\mathfrak{b})$, *these rules realize full extensionality reasoning while avoiding cut-simulation effects. The extensionality principles for Leibniz equality are implied.*

Pre-Unification
$$\frac{C \vee [h\overline{s^n} =^{\beta} h\overline{t^n}]^{\mathtt{ff}} \quad n \geq 1, \beta \in \{o, \iota\}, h_{\overline{\alpha^n}\to\beta} \in \Sigma}{C \vee [s^1 =^{\alpha_1} t^1]^{\mathtt{ff}} \quad \cdots \quad C \vee [s^n =^{\alpha_n} t^n]^{\mathtt{ff}}}\,\mathcal{R}(d)$$

$$\frac{C \vee [s = s]^{\mathtt{ff}}}{C}\,\mathcal{R}(\mathit{Triv}) \qquad \frac{C \vee [X = s]^{\mathtt{ff}} \quad X \notin \mathit{free}(s)}{[s/X]C}\,\mathcal{R}(\mathit{Subst})$$

$$\frac{C \vee [F\overline{s^n} =^{\beta} h\overline{t^m}]^{\mathtt{ff}} \qquad n \geq 1, h_{\overline{\gamma^m}\to\beta} \in \Sigma, l \in \mathcal{AB}^{(h)}_{\overline{\alpha^n}\to\beta}}{C \vee [F = l]^{\mathtt{ff}} \;\vee\; [F\overline{s^n} =^{\beta} h\overline{t^m}]^{\mathtt{ff}}}\,\mathcal{R}(\mathit{FlexRigid})$$

This set of rules implements pre-unification, cf. [67; 93]. LEO-*II actually packages its unification steps into a single, abstract rule called* `extuni`. *This package also integrates the extensionality rules* $\mathcal{R}(\mathfrak{f}_{\mathtt{ff}})$ *and* $\mathcal{R}(\mathfrak{b}_{\mathtt{ff}})$. *The integration of these two rules, in particular, of* $\mathcal{R}(\mathfrak{b}_{\mathtt{ff}})$, *is the reason why pre-unification in* LEO-*II should rather be called extensional pre-unification; cf. [11]. Moreover, logical constants, such as disjunction, equality, etc. are interpreted and a special treatment is provided for them (an example would be symmetric decomposition in rule* $\mathcal{R}(d)$ *in case* h *is* $\vee$ *or* $=^{\alpha}$*).*

In rule $\mathcal{R}(\mathit{FlexRigid})$ *(and again in rule* $\mathcal{R}(\mathit{PrimSubst})$ *below), we use the symbol* $\mathcal{AB}^{(h)}_{\overline{\alpha^n}\to\beta}$ *to denote the set of approximating/partial bindings parametric to a type* $\overline{\alpha^n} \to \beta$ *and to a constant* $h_{\overline{\gamma^m}\longrightarrow\beta}$. *This is explained further next; see also [93]. Given a name* k *(where a name is either a constant or a variable) of type* $\overline{\gamma^m} \longrightarrow \beta$, *term* l *having form* $\lambda\overline{X^n_{\alpha^n}}(k\overline{r^m})$ *is a partial binding of type* $\overline{\alpha^n} \to \beta$ *and head* k. *Each* $r^{i\leq m}$ *has form* $H^i\overline{X^n_{\alpha^n}}$ *where* $H^{i\leq m}$ *are fresh variables typed* $\overline{\alpha^n} \longrightarrow \gamma^{i\leq m}$. *Projection bindings are partial bindings whose head* k *is one of* $X^{i\leq l}$. *Imitation bindings are partial bindings whose head* k *is identical to the*

given symbol h in the superscript of $\mathcal{AB}^{(h)}_{\overline{\alpha^n}\to\beta}$. $\mathcal{AB}^{(h)}_{\overline{\alpha^n}\to\beta}$ is the set of all projection and imitation bindings modulo type $\overline{\alpha^n}\to\beta$ and h.

LEO-*II follows Huet [68; 69] in regarding flexflex clauses, that is, clauses consisting only of flexflex unification literals $[F\overline{s^n} =^\beta G\overline{t^m}]^{\mathrm{ff}}$ (where both F and G are variables), to be empty.*

$$\textbf{\textit{Primitive Substitution}}\quad \frac{C \vee [Q_{\overline{\alpha^n}\to o}\overline{s^n}]^p \quad l \in \mathcal{AB}^{(\neg,\vee,\Pi^\alpha,=^\alpha)}_{\overline{\alpha^n}\to o}}{[l/Q](C \vee [Q_{\overline{\alpha^n}\to o}\overline{s^n}]^p)}\,\mathcal{R}(PrimSubst)$$

In this rule, which is related to Huet's splitting rule [68; 69] and Andrews's primitive substitutions [4], $\mathcal{AB}^{(\neg,\vee,\Pi^\alpha,=^\alpha)}_{\overline{\alpha^n}\to o}$ stands for $\mathcal{AB}^{(\neg)}_{\overline{\alpha^n}\to o}\cup\mathcal{AB}^{(\vee)}_{\overline{\alpha^n}\to o}\cup\mathcal{AB}^{(\Pi^\alpha)}_{\overline{\alpha^n}\to o}\cup\mathcal{AB}^{(=^\alpha)}_{\overline{\alpha^n}\to o}$. This rule introduces a certain amount of blind guessing into the proof procedure. However, unlike in sequent calculus rule $\mathcal{G}(\Pi^l_-)$ only the top-level logical structure of the instantiation term l is guessed, while further decisions on l are delayed. The hope is that they can eventually be determined by pre-unification in subsequent resolution steps. Generally, however, subsequent applications of rule $\mathcal{R}(PrimSubst)$ are permitted and the deeper logical structure of l may thus be guessed later. It is an open challenge to suitably restrict this rule without threatening completeness.

A simple, prominent example to illustrate the need for splittings is $\exists P_o.P$. When using resolution the formula is first negated and then normalized to clause $[X_o]^{\mathrm{ff}}$, where X is a predicate variable. There is no resolution partner for this clause available, hence the empty clause can not be derived. However, when guessing some top-level, logical structure for X, here the substitution $[\neg Y/X]$ is suitable, then $[\neg Y]^{\mathrm{ff}}$ is derived, which normalizes into a new clause $[Y]^{\mathrm{tt}}$. Now, resolution between the clauses $[X]^{\mathrm{ff}}$ and $[Y]^{\mathrm{tt}}$ with substitution $[Y/X]$ directly leads to the empty clause.

$$\textbf{\textit{Choice}}\qquad \frac{[PX]^{\mathrm{ff}} \vee [P(f_{(\alpha\to o)\to\alpha}P)]^{\mathrm{tt}}}{\mathsf{CFs} \longleftarrow \mathsf{CFs}\cup\{f_{(\alpha\to o)\to\alpha}\}}\,\mathcal{R}(DetectChoice)$$

$$\frac{C := C' \vee [s[E_{(\alpha\to o)\to\alpha}t]]^p \qquad \begin{array}{c} E=\epsilon \text{ for } \epsilon\in\mathsf{CFs} \text{ or } E\in free(C),\\ free(t)\subseteq free(C), Y \text{ fresh}\end{array}}{[t\ Y]^{\mathrm{ff}} \vee [t\ (\epsilon_{(\alpha\to o)\to\alpha}t)]^{\mathrm{tt}}}\,\mathcal{R}(Choice)$$

Rule $\mathcal{R}(Choice)$ investigates whether a term $\epsilon_{(\alpha\to o)\to\alpha}t_{\alpha\to o}$ (where ϵ is a choice function, registered and memorized in a special set CFs, or a free variable) is contained as a subterm of a literal $[s]^p$ in a clause C. In this case it adds the instantiation of the choice axiom at type $(\alpha\to o)\to\alpha$ with term $t_{\alpha\to o}$ to the search space.[3] *Side-conditions guard against unsound reasoning, such as the 'uncapturing' of free variables in t. Additionally, rule $\mathcal{R}(DetectChoice)$ detects and removes uninstantiated choice-axiom clauses from the search space (remember that they are cut-strong) and registers the corresponding choice function sym-*

[3] Note that the instantiation of the choice axiom (scheme) $\forall F_{\tau\to o}((\exists Y_\tau FY)\to F(\epsilon_{(\tau\to o)\to\tau}F))$ for term t leads to the clause $[t\ Y]^{\mathrm{ff}}\vee[t\ (\epsilon_{(\alpha\to o)\to\alpha}t)]^{\mathrm{tt}}$.

bols f in CFs. By default, CFs contains at least one choice function symbol for each choice type. The rule does not describe a typical logical inference, since the conclusion of the rule indicates a side-effect which extends the set CFs of choice functions and which removes a clause. Both rules are obviously motivated by the idea to avoid cut-simulation effects. Moreover, it is easy to see that they are sound: $\mathcal{R}(DetectChoice)$ simply removes (cut-strong) clauses from the search space and registers choice functions, and for any registered choice function f, the rule $\mathcal{R}(Choice)$ only introduces new instances of the corresponding choice axiom.

Both choice rules can be disabled in LEO-II with the help of a special flag.

We now briefly summarize the organization of proof search in Leo-II.

In first-order logic, unification is decidable, and it is used as an eager filter during resolution. Unification in HOL is undecidable in general, so it is used more carefully. Therefore, Leo-II relies on a variant of Huet's pre-unification, which is a semi-decidable procedure. It works by accumulating flexflex unification pairs as unification constraints. When a clause consists only of flexflex constraints then it is considered to be empty, since, as Huet showed [67], such a system of equations always has solutions. An additional aspect of unification in Leo-II is the integration of the extensionality rule $\mathcal{R}(\mathfrak{b}_{tt})$. In addition to flexflex constraints, pre-unification in Leo-II may thus return negated equivalence literals.

Though it was originally intended as an alternative option for Leo-II's architecture, lazy unification has never been implemented. Instead eager unification is used in Leo-II, which works as follows: extensional pre-unification is applied to clauses with a predefined depth bound (for example, maximally five[4] nestings of the branching flex-rigid rule; modulo this depth-bound higher-order pre-unification becomes decidable, but at the cost of completeness; however, regarding the unification depth an iterative deepening approach is actually provided in Leo-II). The solved unification constraints are exhaustively applied in the resulting clauses, and any remaining flexflex unification pairs or negated equivalence literals generated by $\mathcal{R}(\mathfrak{b}_{tt})$ are kept as literals of the result clause.

Leo-II's calculus-level treatment of the axiom of choice is inspired by the work of Mints [78]. Choice is related to Skolemization. In HOL, Skolemization is not as straightforward as in first-order logic [77]. Naïve Skolemization is unsound with respect to Henkin models that invalidate choice, and incomplete with respect to Henkin models that validate choice [8; 17].

Like many other provers, Leo-II spends its time looping during its exploration of the search space — executing its *main loop*. By *search space* we mean the totality of clauses surveyed by Leo-II during its execution. Each iteration of this loop might generate new clauses, thus contributing to the representation of the search space that is kept by Leo-II. Each iteration does *not* change the satisfiability of the problem and its search space; this is an invariant of a prover's main loop.

[4] The pre-unification depth is a parameter in Leo-II that can be specified at the command line. By default Leo-II currently operates with values up to depth 8. So far there has been no exhaustive empirical investigation of the optimal setting of the pre-unification depth.

Unlike many provers Leo-II keeps an additional representation of the search space. This is used to store the input to external provers. The contents of this store are produced by translating the clauses in the main store. The source clauses consist of higher-order clauses, and the target clauses are encoded in the target logic. Since Leo-II currently only cooperates with first-order provers, the target clauses consist of first-order clauses.

The first-order clauses are accumulated during iterations of Leo-II's main loop, and are periodically sent to the external prover with which Leo-II is cooperating. If the external prover finds the first-order clauses to be inconsistent then, assuming that the translation was sound, it means that the original HOL clauses must also be inconsistent. This refutation is accepted by Leo-II, and presented to the user as a refutation of the initial conjecture.

Various translations from HOL to first-order logic are implemented in Leo-II [38]. These translations differ in the amount of information they encode in the resulting FO formulas. Encoding less information can lead to incompleteness. Leo-II also implements a method devised in [55], which describes an analysis on the cardinalities of types in order to safely erase some information. As part of this analysis, SAT problems are generated, and these are processed by MiniSat via an interface adapted from the Satallax prover [44; 47].

4.2 Leo-II's Proof Certificates

Running Leo-II on a problem can have several outcomes: the conjecture could be found to be a theorem, or found to be a non-theorem, or the prover could give up (because of a timeout, for instance). Leo-II conforms to the SZS standard ontology [97] for communicating the outcome of a proof attempt. This makes it easier for external tools to interpret this outcome.

In addition to this, Leo-II can also output a proof certificate. This details the justification for the outcome given by Leo-II, by providing the reasoning steps used by Leo-II to derive a refutation. This could then be used by an independent system to check Leo-II's reasoning, or to use that derivation in a bigger formalisation.

Leo-II can generate proof certificates in two levels of detail. When called with the option `-po 1`, Leo-II produces a proof containing the reasoning steps made by Leo-II alone — information on the reasoning made by the cooperating FO ATP are omitted. When called with option `-po 2`, Leo-II tries to merge the proof steps of the cooperating FO ATP with its own steps in order to return a joint THF-FOF proof object [96]. The '-po 2' mode is unfortunately still very brittle and therefore not yet recommended for extensive use.

Leo-II's proof certificates are encoded in the TPTP TSTP syntax [98; 99], in which each inference is encoded as an annotated formula. The inference's conclusion appears as the formula (e.g., in THF0 or FOF syntax), and the inference's hypotheses and other meta-data are referenced resp. encoded in the formula's annotations. Examples of proofs of both levels of detail are provided on the Leo-II website, at `http://page.mi.fu-berlin.de/cbenzmueller/leo/download.html`.

5 Tools and Provers

5.1 The TPTP THF Initiative

To foster the systematic development and improvement of higher-order automated theorem proving systems the TPTP THF infrastructure [100] has been initiated (THF stands for *typed higher-order form*). This project, which was supported by several other members of the community, has introduced the THF syntax for higher-order logic, it has developed a library of benchmark and example problems, and it provides various support tools for the new THF0 language fragment. The THF0 language supports HOL with choice.

Version 6.0.0 of the TPTP library contains more than 3000 problems in the THF0 language. The library also includes the entire problem library of Andrews's TPS project, which, among others, contains formalizations of many theorems of his textbook [6]. The first-order TPTP infrastructure [99] provides a range of resources to support usage of the TPTP problem library. Many of these resources are now immediately applicable to the higher-order setting although some have required changes to support the new features of THF. The development of the THF0 language has been paralleled and significantly influenced by the development of the Leo-II prover. Several other provers have quickly adopted this language, leading to fruitful mutual comparisons and evaluations. Several implementation bugs in different systems have been detected this way.

5.2 Automated Theorem Provers for HOL

We briefly describe the currently available, fully automated theorem provers for HOL (with choice). These systems all support the new THF0 language and they can be employed online (avoiding local installations) via Sutcliffe's SystemOnTPTP facility [98; 99].[5] The descriptions below have been adapted from [24].

TPS The TPS prover can be used to prove theorems of ETT or HOL automatically, interactively, or semi-automatically. When searching for a proof automatically, TPS first searches for an expansion proof [76] or an extensional expansion proof [45] of the theorem. Part of this process involves searching for acceptable matings [3]. Using higher-order unification, a pair of occurrences of subformulas (which are usually literals) is mated appropriately on each vertical path through an expanded form of the theorem to be proved. Skolemization and pre-unification is employed, and calculus rules for extensionality reasoning are provided. The behavior of TPS is controlled by sets of flags, also called modes. About fifty modes have been found that collectively suffice for automatically proving virtually all the theorems that TPS has proved automatically thus far. A simple scheduling mechanism is employed in TPS to sequentially run these modes for a limited amount of time. The resulting fully automated system is called *TPS (TPTP)*.

[5] See also `http://www.tptp.org/cgi-bin/SystemOnTPTP`.

LEO-*II* [39; 38], the successor of LEO-I [23], has been described in more detail in §4. Communication between LEO-II and the cooperating first-order theorem prover uses the TPTP language and standards. LEO-II outputs proofs in TPTP TSTP syntax. An incremental communication between LEO-II and the first-order prover(s) would clearly make sense, and this has been on the list of planned improvements for quite some time. However, such a solution has not yet been implemented. The latest versions of LEO-II provide some modest (counter-)model finding capabilities.

Isabelle/HOL The Isabelle/HOL [82] system has originally been designed as an interactive prover. However, in order to ease user interaction several automatic proof tactics have been added over the years. By appropriately scheduling a subset of these proof tactics, some of which are quite powerful, Isabelle/HOL has in recent years been turned also into an automatic theorem prover, that can be run from a command shell like other provers. The latest releases of this automated version of Isabelle/HOL provide native support for different TPTP syntax formats, including THF0. The most powerful proof tactics that are scheduled by Isabelle/HOL include the *sledgehammer* tool [41], which invokes a sequence of external first-order and higher-order theorem provers, the model finder *Nitpick* [42], the equational reasoner *simp* [81], the untyped tableau prover *blast* [87], the simplifier and classical reasoners *auto*, *force*, and *fast* [86], and the best-first search procedure *best*. The TPTP incarnation of Isabelle/HOL does not yet output proof terms.

Satallax The higher-order automated theorem prover Satallax [44; 47] comes with model finding capabilities. The system is based on a complete ground tableau calculus for HOL (with choice) [8]. An initial tableau branch is formed from the assumptions of a conjecture and negation of its conclusion. From that point on, Satallax tries to determine unsatisfiability or satisfiability of this branch. Satallax progressively generates higher-order formulas and corresponding propositional clauses. Satallax uses the SAT solver MiniSat as an engine to test the current set of propositional clauses for unsatisfiability. If the clauses are unsatisfiable, the original branch is unsatisfiable. Satallax employs restricted instantiation and pre-unification, and it provides calculus rules for extensionality and choice. If there are no quantifiers at function types, the generation of higher-order formulas and corresponding clauses may terminate. In that case, if MiniSat reports the final set of clauses as satisfiable, then the original set of higher-order formulas is satisfiable (by a standard model in which all types are interpreted as finite sets). Satallax outputs proofs in different formats, including Coq proof scripts and Coq proof terms.

Nitpick and Refute These systems are (counter-)model finders for HOL. The ability of Isabelle to find (counter-)models using the *Refute* and *Nitpick* [42] commands has also been integrated into automatic systems. They provide the capability to find models for THF0 formulas, which confirm the satisfiability of

axiom sets, or the counter-satisfiability of non-theorems. The generation of models is particularly useful for exposing errors in some THF0 problem encodings, and revealing bugs in the THF0 theorem provers. Nitpick employs Skolemization.

agsyHOL The agsyHOL prover [74] is based on a generic lazy narrowing proof search algorithm. Backtracking is employed and a comparably small search state is maintained. The prover outputs proof terms in sequent style which can be verified in the Agda system.

coqATP The coqATP prover [40] implements (the non-inductive) part of the calculus of constructions. The system outputs proof terms which are accepted as proofs by Coq (after the addition of a few definitions). The prover has axioms for functional extensionality, choice, and excluded middle. Propositional extensionality is not supported yet. In addition to axioms, a small library of basic lemmas is employed.

Satallax-MaLeS and Leo-*II-MaLeS* MaLeS is an automatic tuning framework for automatic theorem provers. It combines random-hill climbing based strategy finding with strategy scheduling via learned runtime predictions. The MaLeS system has been successfully combined with Satallax and with Leo-II [72].

6 Proof Applications

With respect to full proof automation the TPS system has long been the leading system, and the system has been employed to build up the TPS library of formalized and automated mathematical proofs. More recently, however, TPS is outperformed by several other THF0 theorem provers. Below we briefly point to some selected recent applications of the leading systems.

Both Isabelle/HOL and Nitpick have been successfully employed to check a formalization of a C++ memory model against various concurrent programs written in C++ (such as a simple locking algorithm) [43]. Moreover, Nitpick has been employed in the development of algebraic formal methods within Isabelle/HOL [64].

Isabelle/HOL, Satallax, and Leo-II performed well in recent experiments related to the Flyspeck project [65], in which a formalized proof of the Kepler conjecture has been developed (mainly) in HOL Light; cf. the experiments reported in [70], which inter alia investigated the potential of several ATPs for automating subgoals in the Flyspeck corpus. In these experiments the higher-order automated teorem provers performed better than many prominent first-order theorem provers, and they contributed many unique solutions.

Most recently, Leo-II, Satallax, and Nitpick were employed to achieve a formalization, mechanization, and automation of Gödel's ontological proof of the existence of God [32; 31; 27; 28; 30; 29; 26]. This work employs a semantic embedding of quantified modal logic in HOL [35], similar to the embedding presented in §2. Leo-II was the first prover to fully automate the four key steps of

Dana Scott's version of the proof [94]. These results were subsequently confirmed by Satallax, and consistency of the axioms was shown with Nitpick. Moreover, some previously unknown results were contributed by the provers.

Using the semantic embeddings approach, a wide range of propositional and quantified non-classical logics, including parts of their meta-theory and their combinations, can be automated with THF0 reasoners [14]. Automation is thereby competitive, as recent experiments for first-order modal logics show [37; 25]. In [13] HOL theorem provers have been successfully employed to reason about meta-level properties of modal logics. More precisely, the provers have been employed to verify the modal logic cube.

THF0 reasoners can also be fruitfully employed for reasoning in expressive ontologies [36]. Furthermore, the heterogeneous toolset HETS [79] employs THF0 to integrate the automated higher-order provers Satallax, Leo-II, Nitpick, Refute, and Isabelle/HOL.

7 Trends and Open Problems

The development of automated theorem provers for HOL has significantly progressed recently. At the same time the systems still lag far behind the theoretical and technical maturity that has been achieved in first-order automated theorem proving. Existing HOL provers have nevertheless already demonstrated their competitiveness in various applications.

Further challenges in this field include the development and systematic improvement of the theoretical foundations, in particular, of the underlying calculi, and better proof search organization.

The challenges also include the automation of reasoning with polymorphism, subtypes, type classes, and possibly even dependent types as supported in modern interactive proof assistants. The rationale here is that automated theorem provers for HOL should ideally be applicable directly to the logics as provided and employed in these proof assistants. It will be relevant to improve and adapt the TPTP THF languages accordingly.

Moreover, challenges include the suitable automation of any of the remaining, prominent cut-strong principles of HOL, such as induction and description.

There are also significant challenges regarding proof representation and user interaction. Experience shows that HOL proofs, for example, as produced by Leo-II or Satallax, can be quite hard to read and understand by humans. To foster the use of our systems in contexts where human understanding and the generation of human level answers is relevant, significant further improvements are thus required.

Acknowledgements This text is based on several earlier publications, as referenced throughout the paper, with various co-authors, including Chad Brown, Valerio Genovese, Michael Kohlhase, Dale Miller, Larry Paulson, Nik Sultana, Frank Theiss. I am very grateful to all of them. I am also grateful to the anonymous reviewers and Bruno Woltzenlogel Paleo for many useful comments and

suggestions, and for proof reading this document. This work has been supported by the German National Research Foundation (DFG) under grants BE 2501/9-1 and BE 2501/11-1.

Bibliography

[1] Peter B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36(3):414–432, 1971.

[2] Peter B. Andrews. General models and extensionality. *Journal of Symbolic Logic*, 37(2):395–397, 1972.

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

[4] Peter B. Andrews. On connections and higher order logic. *Journal of Automated Reasoning*, 5(3):257–291, 1989.

[5] Peter B. Andrews. Classical type theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 15, pages 965–1007. Elsevier Science, Amsterdam, 2001.

[6] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.

[7] Peter B. Andrews. Church's type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, spring 2009 edition, 2009.

[8] Julian Backes and Chad Edward Brown. Analytic tableaux for higher-order logic with choice. *Journal of Automated Reasoning*, 47(4):451–479, 2011.

[9] Christoph Benzmüller. *Equality and Extensionality in Automated Higher-Order Theorem Proving*. PhD thesis, Saarland University, 1999.

[10] Christoph Benzmüller. Extensional higher-order paramodulation and RUE-resolution. In Harald Ganzinger, editor, *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, number 1632 in LNCS, pages 399–413. Springer, 1999.

[11] Christoph Benzmüller. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–235, 2002.

[12] Christoph Benzmüller. Automating access control logic in simple type theory with LEO-II. In Dimitris Gritzalis and Javier López, editors, *Emerging Challenges for Security, Privacy and Trust, 24th IFIP TC 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18-20, 2009. Proceedings*, volume 297 of *IFIP*, pages 387–398. Springer, 2009.

[13] Christoph Benzmüller. Verifying the modal logic cube is an easy task (for higher-order automated reasoners). In Simon Siegler and Nathan Wasser, editors, *Verification, Induction, Termination Analysis - Festschrift for Christoph Walther on the Occasion of His 60th Birthday*, volume 6463 of *LNCS*, pages 117–128. Springer, 2010.

[14] Christoph Benzmüller. Combining and automating classical and non-classical logics in classical higher-order logic. *Annals of Mathematics and*

Artificial Intelligence (Special issue Computational logics in Multi-agent Systems (CLIMA XI)), 62(1-2):103–128, 2011.

[15] Christoph Benzmüller. Automating quantified conditional logics in HOL. In Francesca Rossi, editor, *23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, Beijing, China, 2013.

[16] Christoph Benzmüller. A top-down approach to combining logics. In *Proc. of the 5th International Conference on Agents and Artificial Intelligence (ICAART)*, Barcelona, Spain, 2013. SciTePress Digital Library.

[17] Christoph Benzmüller and Chad Brown. A structured set of higher-order problems. In Joe Hurd and Thomas F. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*, number 3603 in LNCS, pages 66–81. Springer, 2005.

[18] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.

[19] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Cut elimination with xi-functionality. In Christoph Benzmüller, Chad Brown, Jörg Siekmann, and Richard Statman, editors, *Reasoning in Simple Type Theory — Festschrift in Honor of Peter B. Andrews on His 70th Birthday*, Studies in Logic, Mathematical Logic and Foundations, pages 84–100. College Publications, 2008.

[20] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1:6):1–21, 2009.

[21] Christoph Benzmüller, Dov Gabbay, Valerio Genovese, and Daniele Rispoli. Embedding and automating conditional logics in classical higher-order logic. *Annals of Mathematics and Artificial Intelligence*, 66(1-4):257–271, 2012.

[22] Christoph Benzmüller and Valerio Genovese. Quantified conditional logics are fragments of HOL. In *The International Conference on Non-classical Modal and Predicate Logics (NCMPL)*, Guangzhou (Canton), China, 2011. The conference had no published proceedings; the paper is available as arXiv:1204.5920v1.

[23] Christoph Benzmüller and Michael Kohlhase. 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*, number 1421 in LNCS, pages 139–143. Springer, 1998.

[24] Christoph Benzmüller and Dale Miller. Automation of higher-order logic. In Jörg Siekmann, Dov Gabbay, and John Woods, editors, *Handbook of the History of Logic, Volume 9 — Logic and Computation*. Elsevier, 2014. In print.

[25] Christoph Benzmüller, Jens Otten, and Thomas Raths. Implementing and evaluating provers for first-order modal logics. In *Proc. of the 20th*

European Conference on Artificial Intelligence (ECAI 2012), pages 163–168, Montpellier, France, 2012.

[26] Christoph Benzmüller, Leon Weber, and Bruno Woltzenlogel Paleo. Computer-assisted analysis of the anderson-hjek ontological controversy. In *1st World Congress on Logic and Religion*, 2015.

[27] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Gödel's god in isabelle/hol. *Archive of Formal Proofs*, 2013, 2013.

[28] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Gdel's god on the computer. In *10th International Workshop on the Implementation of Logics*, 2013.

[29] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Formalization and automated verification of gdel's proof of god's existence. In *4th World Congress on the Square of Opposition*, 2014.

[30] Christoph Benzmüller and Bruno Woltzenlogel Paleo. On logic embeddings and gdel's god. In *22nd International Workshop on Algebraic Development Techniques*, 2014.

[31] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Formalization, mechanization and automation of Gödel's proof of God's existence. 2013. Preprint available as arXiv:1308.4526.

[32] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Automating Gödel's ontological proof of god's existence with higher-order automated theorem prover. In *Proc. of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 93–98, 2014. 2014.

[33] Christoph Benzmüller and Lawrence Paulson. Exploring properties of normal multimodal logics in simple type theory with LEO-II. In Christoph Benzmüller, Chad Brown, Jörg Siekmann, and Richard Statman, editors, *Reasoning in Simple Type Theory — Festschrift in Honor of Peter B. Andrews on His 70th Birthday*, Studies in Logic, Mathematical Logic and Foundations, pages 386–406. College Publications, 2008.

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

[35] Christoph Benzmüller and Lawrence Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7, 2013.

[36] Christoph Benzmüller and Adam Pease. Higher-order aspects and context in SUMO. *Journal of Web Semantics (Special Issue on Reasoning with context in the Semantic Web)*, 12-13:104–117, 2012.

[37] Christoph Benzmüller and Thomas Raths. HOL based first-order modal logic provers. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 127–136, Stellenbosch, South Africa, 2013. Springer.

[38] Christoph Benzmüller and Nik Sultana. LEO-II version 1.5. In Jasmin Christian Blanchette and Josef Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 2–10. EasyChair, 2013.

[39] Christoph Benzmüller, Frank Theiss, Lawrence Paulson, and Arnaud Fietzke. LEO-II - a cooperative automatic theorem prover for higher-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 162–170. Springer, 2008.

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

[41] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 51(1):109–128, 2013.

[42] 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, *Proc. of ITP 2010*, volume 6172 of *LNCS*, pages 131–146. Springer, 2010.

[43] 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.

[44] Chad E. Brown. Satallax: an automatic higher-order prover. *Journal of Automated Reasoning*, pages 111–117, 2012.

[45] Chad E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 2004. See also Chad E. Brown, *Automated Reasoning in Higher-Order Logic*, College Publications, 2007.

[46] Chad E. Brown. Reasoning in extensional type theory with equality. In Robert Nieuwenhuis, editor, *Proc. of CADE-20*, volume 3632 of *LNCS*, pages 23–37. Springer, 2005.

[47] Chad E. Brown. Reducing higher-order theorem proving to a sequence of sat problems. *Journal of Automated Reasoning*, 51(1):57–77, 2013.

[48] Chad E. Brown and Gert Smolka. Analytic tableaux for simple type theory and its first-order fragment. *Logical Methods in Computer Science*, 6(2), 2010.

[49] Brian F. Chellas. Basic conditional logic. *Journal of Philosophical Logic*, 4(2):133–153, 1975.

[50] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge: Cambridge University Press, 1980.

[51] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[52] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.

[53] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[54] Leon Chwistek. *The Limits of Science: Outline of Logic and of the Methodology of the Exact Sciences*. London: Routledge and Kegan Paul, 1948.

[55] Koen Claessen, Ann Lilliesträm, and Nicholas Smallbone. Sort it out with monotonicity. In *Proceedings of CADE-23*, volume 6803 of *LNAI*, pages 207–221. Springer, 2011.

[56] James P. Delgrande. On first-order conditional logics. *Artificial Intelligence*, 105(1-2):105–137, 1998.

[57] Vincent J. Digricoli. Resolution by unification and equality. In William H. Joyner, editor, *4th Workshop on Automated Deduction*, Austin, Texas, 1979.

[58] Herbert B. Enderton. Second-order and higher-order logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, fall 2012 edition, 2012.

[59] William M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.

[60] Melvin Fitting. Interpolation for first order S5. *Journal of Symbolic Logic*, 67(2):621–634, 2002.

[61] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879. Translated in [106].

[62] Nir Friedman, Joseph Y. Halpern, and Daphne Koller. First-order conditional logic for default reasoning revisited. *ACM Transactions on Computational Logic*, 1(2):175–207, 2000.

[63] Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *2nd Scandinavian Logic Symposium*, pages 63–92. North-Holland, Amsterdam, 1971.

[64] Walter Guttmann, Georg Struth, and Tjark Weber. Automating algebraic methods in isabelle. In Shengchao Qin and Zongyan Qiu, editors, *Proc. of ICFEM 2011*, volume 6991 of *LNCS*, pages 617–632. Springer, 2011.

[65] Thomas Hales. Mathematics in the Age of the Turing Machine. *arXiv:1302.2898*, February 2013.

[66] Leon Henkin. A theory of propositional types. *Fundamatae Mathematicae*, 52:323–344, 1963.

[67] Gérard Huet. A Unification Algorithm for Typed Lambda-Calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[68] Gérard P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic*. PhD thesis, Case Western Reserve University, 1972.

[69] Gérard P. Huet. A mechanization of type theory. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 139–146, 1973.

[70] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with flyspeck. *Journal of Automated Reasoning*, 53:173–213, 2014.

[71] Michael Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle*. PhD thesis, Saarland University, 1994.

[72] Daniel Külwein. *Machine Learning for Automated Reasoning*. PhD thesis, Radboud University Nijmegen, Netherlands, 2014.

[73] Daniel Leivant. Higher-order logic. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 229–321. Oxford University Press, 1994.
[74] Fredrik Lindblad. agsyHOL website. `https://github.com/frelindb/agsyHOL`, 2013.
[75] Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, 2002.
[76] Dale Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.
[77] Dale A. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, 1983. 81 pp.
[78] Grigori Mints. Cut-elimination for simple type theory with an axiom of choice. *Journal of Symbolic Logic*, 64(2):479–485, 1999.
[79] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, Hets. In *Proceedings of TACAS 2007*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.
[80] Reinhard Muskens. Intensional models for the theory of types. *Journal of Symbolic Logic*, 72(1):98–118, 2007.
[81] Tobias Nipkow. Equational reasoning in Isabelle. *Sci. Comput. Program.*, 12(2):123–149, 1989.
[82] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002.
[83] D. Nute. *Topics in conditional logic*. Reidel, Dordrecht, 1980.
[84] Hans-Jürgen Ohlbach. Semantics Based Translation Methods for Modal Logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.
[85] N. Olivetti, G.L. Pozzato, and C. Schwind. A sequent calculus and a theorem prover for standard conditional logics. *ACM Transactions on Computational Logic*, 8(4), 2007.
[86] Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *LNCS*. Springer, 1994.
[87] Lawrence C. Paulson. A generic tableau prover and its integration with isabelle. *Journal of Universal Computer Science*, 5:51–60, 1999.
[88] Dag Prawitz. Hauptsatz for higher order logic. *Journal of Symbolic Logic*, 33:452–457, 1968.
[89] Frank P. Ramsey. The foundations of mathematics. In *Proceedings of the London Mathematical Society*, volume 25 of *2*, pages 338–384, 1926.
[90] Bertrand Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.
[91] Kurt Schütte. Semantical and syntactical properties of simple type theory. *Journal of Symbolic Logic*, 25(4):305–326, 1960.
[92] Raymond M. Smullyan. A unifying principle for quantification theory. *Proc. Nat. Acad Sciences*, 49:828–832, 1963.
[93] Wayne Snyder and Jean H. Gallier. Higher order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.

[94] John H. Sobel. *Logic and Theism: Arguments for and Against Beliefs in God*, chapter Appendix B. Notes in Dana Scott's Hand, pages 145–146. Cambridge U. Press, 2004.

[95] Robert C. Stalnaker. A theory of conditionals. In *Studies in Logical Theory*, pages 98–112. Blackwell, 1968.

[96] Nik Sultana and Christoph Benzmüller. Understanding LEO-II's Proofs. In Konstantin Korovin, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL 2012*, Easychair EPiC Series, 22:33-52, 2013.

[97] Geoff Sutcliffe. The SZS ontologies for automated reasoning software. In: The LPAR 2008 Workshops: KEAPPA and IWIL, CEUR Workshop Proceedings (http://ceur-ws.org/), vol. 418, 2008.

[98] Geoff Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR 2010*, volume 6355 of *LNCS*, pages 1–12. Springer, 2010.

[99] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

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

[101] William W. Tait. A nonconstructive proof of Gentzen's Hauptsatz for second order predicate logic. *Bulletin of the American Mathematical Society*, 72:980983, 1966.

[102] Moto-o Takahashi. A proof of cut-elimination theorem in simple type theory. *Journal of the Mathematical Society of Japan*, 19:399–410, 1967.

[103] Gaisi Takeuti. On a generalized logic calculus. *Japanese Journal of Mathematics*, 23:39–96, 1953. Errata: ibid, vol. 24 (1954), 149–156.

[104] Gaisi Takeuti. An example on the fundamental conjecture of GLC. *Journal of the Mathematical Society of Japan*, 12:238–242, 1960.

[105] Gaisi Takeuti. *Proof Theory*, volume 81 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1975.

[106] Jean van Heijenoort. *From Frege to Gödel: A Source Book in Mathematics, 1879-1931*. Source books in the history of the sciences series. Harvard Univ. Press, Cambridge, MA, 3rd printing, 1997 edition, 1967.

[107] Max Wisniewski and Alexander Steen. Embedding of quantified higher-order nominal modal logic into classical higher-order logic. In *Proc. of the 1st International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL)*, Vienna, 2014.

[108] Bruno Woltzenlogel Paleo. An Embedding of Neighbourhood-Based Modal Logics in HOL. Available at `https://github.com/Paradoxika/ModalLogic`.