Nitpicking Lambda-free Higher-Order Logic

Alexander Steen[1][0000−0001−8781−9462] and Christoph Benzmüller[2,1][0000−0002−3392−3093]

[1] University of Luxembourg, Esch/Alzette, Luxembourg
alexander.steen@uni.lu
[2] Freie Universität Berlin, Berlin, Germany
c.benzmueller@fu-berlin.de

Abstract. λ-free (comprehension free) higher-order logic has recently been proposed and studied as a starting point for the lifting of superposition based theorem proving techniques to classical higher-order logic. This promising line of research, which we fully support, has received good attention in the deduction systems community due to its high potential to enable significant further improvements in higher-order automated theorem proving. However, in this paper we challenge and nitpick the λ-free higher-order logic framework. Our motivation is to gain and share some insights about its non-trivial semantical and proof theoretical foundations. We present some practical examples, many of which are concerned with comprehension issues, that reveal some questionable, partly counter-intuitive, inference patterns. We believe that these issues should be known to further implementors of the logic and also to potential users to avoid confusion in practical applications.

Keywords: Higher-Order Logic · Comprehension · Semantics · ATP

1 Introduction

A primary research focus in the automated deduction community has for decades been on classical, single-sorted first-order logic (FOL). This has led to the development of very powerful first-order automated theorem provers such as Vampire, E and Spass. The success of these provers is built on at least two factors: (i) they all adopt the superposition approach [2] as their proof theoretical foundation, and (ii) they combine this approach with sophisticated system architectures and highly optimised technical implementations.

Practical applications often motivate the use of more expressive logics. Multiple sorts, formulas embedded at term-level, λ-abstraction and higher-order quantifiers, as provided in classical higher-order logic (HOL, [1]), enable more intuitive and often more adequate problem encodings, and they may even enable shorter and more efficient proofs in selected areas and cases. HOL also serves as the theoretical foundation of many modern proof assistants [15]. Another benefit of HOL is that, when utilised as a meta-logic, it even allows for elegant semantical embeddings of a multitude of expressive (quantified) non-classical logics, which

in turn triggers many further applications in prominent areas including artificial intelligence, metaphysics and natural language processing [5].

These observations have motivated a recent shift of interest in the automated theorem proving community from its traditional focus on FOL towards supporting increasingly expressive logics. We consider this a very relevant movement which fosters important ties between research activities and communities that not too long ago still considered each other rather as antipodes.

A prominent example of an ongoing research project in that spirit is Matryoshka[3], which *"propose[s] to deliver very high levels of automation to users of proof assistants by fusing and extending two lines of research: automatic and interactive theorem proving"* and to *"enrich superposition and SMT with higher-order (HO) reasoning in a careful manner, to preserve their desirable properties."*

As a starting point this project has selected λ-free higher-order logic (lFHOL, [3]) as an intermediate study subject of choice, situated in between FOL and HOL, in order to conduct some initial theoretical and practical research in direction of the above uttered goals. The term "λ-free" thereby does not mean to express that there is no λ-notation supported in the studied logic formalism, it instead intends to express that no a priori assumptions on comprehension principles are being made. In other words, the semantics of lFHOL, in contrast to HOL, does not prerequire the existence of certain functions and relations in its function spaces. This semantical set-up of lFHOL, however, triggers interesting and relevant questions. For example, what inference patterns does it support and what kind of peculiarities should users of the logic be aware of?

It is such questions this paper is concerned with. We are scrutinising the semantics of lFHOL and investigate whether is supports some unexpected or paradoxical reasoning patterns. Another concern is that, contrary to what has been claimed,[4] the theorem prover Zipperposition [7] seems not to implement the newly introduced theory, since the results we obtain in our experiments with the system diverge from the expected results in several crucial cases. In this sense we are *"nitpicking"* lFHOL and its implementation. Our intend is to positively influence this important research direction and to foster its further development by contributing a critical discussion. As a side result we contribute some benchmark problems for λ-free HOL that are eventually useful also to other related research projects, and at the same time allow potential users of the lFHOL to gain a better understanding of the logics unexpected peculiarities.

Our paper is structured as follows: In Sect. 2 we restate the syntax and semantics of lFHOL. We then present some apparently paradoxical examples in Sect. 3 that challenge the particular choice of semantics and we hint at the source of the problem. Moreover, we present some example conjectures for which the answers obtained from Zipperposition diverge from the expected results.

[3] `http://matryoshka.gforge.inria.fr`

[4] In [3] it is stated on p. 2 that *"The calculi are implemented in the Zipperposition prover"*, and on p. 11 *"We have now also implemented a complete λ-free higher-order mode based on the four calculi described in this paper, extended with polymorphism."* Zipperposition was used to evaluate the theoretical calculi presented in [3].

2 λ-Free Higher-Order Logic

In this section we recapitulate the notion of lFHOL as introduced in previous work [3]. The definitions given below are identical with the original ones unless stated otherwise. Since we are not suffering from space restrictions we may give them in a more explicit and detailed representation though.

2.1 Syntax

lFHOL is a typed logic in which every term is assigned a simple type.

Types. Fix a set of base types BT. The set of *simple types* (henceforth referred to as *types*) is then given by the following inductive definition: Every $\iota \in \mathrm{BT}$ is a type, and for types τ and ν we have that $\tau \to \nu$ is a type.

Remark 1 (Boolean Type). There is no dedicated, bivalent base type $o \in \mathrm{BT}$ for Boolean-typed entities explicitly mentioned in [3]. Clearly, the omission of such a base type o seems in line with the works' intended strict separation of terms and formulas. However, this omission drastically restricts the language and we give examples below that illustrate some questionable or undesirable effects.

Typed Variables. Typed variables have the form $x : \tau$, where τ is a type. The set of typed variables is denoted $\mathcal{V}$.

Signatures. A signature is a non-empty set Σ of symbols with type declarations of the form $\mathrm{f} : \overline{\tau}_n \Rightarrow \tau$ (with $n \geq 0$), where the τ_i and τ are types. In case $n = 0$ we simply write $\mathrm{f} : \tau$. Note that $\Rightarrow$ is not a type constructor, that is, $\overline{\tau}_n \Rightarrow \tau$ is not a valid *type* itself. A *type declaration* is only indicating that f needs to be applied to n mandatory arguments (see also below). An example signature is given by $\Sigma = \{\mathrm{f} : \iota \times \iota \Rightarrow \iota \to \iota, \mathrm{g} : \iota \to \iota \to \iota\}$, where f has two mandatory arguments and one optional argument, g is a function of two (optional) arguments, and $\iota \in \mathrm{BT}$ is a type. The distinction of mandatory and optional arguments is motivated by the fact that naive Skolemization is unsound with respect to higher-order logics without the axiom of choice [9,10], because Skolem symbols might be instantiated as proper functions symbols [4, pp. 20–22]. Following [3], $\overline{\tau}_n$ ambiguously denotes $\tau_1 \times \ldots \times \tau_n$ and $(\tau_1, \ldots, \tau_n)$ for use in type declarations and for mandatory arguments, respectively.

Terms. Given a signature Σ and a set $\mathcal{V}$ of typed variables, the set $\mathcal{T}_\Sigma^X$ of lFHOL *terms* is inductively defined as follows:

(i) Every variable $x : \tau \in X \subseteq \mathcal{V}$ is a term.
(ii) If $\mathrm{f} : \overline{\tau}_n \Rightarrow \tau \in \Sigma$ is a symbol from the signature and, for all $i \in \{1, \ldots, n\}$, every $u_i : \tau_i$ is a term, then $\mathrm{f}(\overline{u}_n) : \tau$ is a term.
(iii) If $t : \tau \to \nu$ and $u : \tau$ are terms, then $t\,u : \nu$ is a term, called *application*.

Non-application terms are called heads. Terms can be decomposed in a unique way into a head applied to zero or more arguments. If $\mathrm{f} : \overline{\tau}_{\mathrm{n}} \Rightarrow \tau \in \Sigma$, for $n > 0$, is a function symbol with n mandatory arguments, then f alone, i.e. without any arguments applied to it, is not considered a proper term. Only the fully applied form $\mathrm{f}(t_1, \ldots, t_n)$ is a term in this case, assuming that all $t_i : \tau_i \in \mathcal{T}_\Sigma^X$ are terms. All terms and subterms $t : \tau$ are uniquely typed. The type information $: \tau$ may be omitted if obvious from the context or irrelevant.

Formulas. The formulas of lFHOL are inductively defined by:

(i) If $t : \tau$ and $s : \tau$ are terms, then $t \approx_\tau s$ is a formula.
(ii) $\bot$ and $\top$ are formulas.
(iii) If φ and ψ are formulas, then $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$ and $\varphi \longrightarrow \psi$ are formulas.
(iv) If x is variable and φ a formula, then $\forall x.\, \varphi$ and $\exists x.\, \varphi$ are formulas.

By convention $t \not\approx s$ abbreviates $\neg(s \approx t)$. Diverging from the original presentation, we require the terms s and t in $s \approx t$ to have the same type. We conjecture that the omission of this requirement in [3] has been an oversight, so that our restriction does not change the intended syntax of lFHOL. Analogous to e.g. classical first-order logic, lFHOL strictly separates between formulas and terms. This is different to the term-logic HOL, where formulas are treated as special terms, namely those with Boolean type o. A relevant syntactic difference between HOL and lFHOL thus is that in the latter formulas may not occur as proper subterms in any expression.

Remark 2 (Propositional symbols). An obvious question is to which degree the practical limitations imposed by the mentioned restrictions on the syntax of lFHOL are impeding practical applications. In particular, there seems no obvious way of expressing atomic propositions or predicates in lFHOL as usually expected. For example, formulas such as "rains $\vee \neg$rains" (with an atomic proposition rains) and formulas expressing a predicate application "tall x" (for some predicate symbol tall) are apparently excluded from the syntax of lFHOL. In lFHOL corresponding expressions could only be formed on the term level. However, without a Boolean type $o \in \mathrm{BT}$, the usual approach taken in equality-based first-order logic, where non-equality predicates are treated as special equations (with artificially introduced symbols for truth and falsehood as in "(raining $\approx \top) \vee \neg$(raining $\approx \top$)" or in "tall$(x) \approx \top$"), is syntactically prevented in lFHOL. If lFHOL, in contrast, is really intended to strictly exclude such formulas as mentioned, then the question arises whether it is justified to call it a logic language. ⌟

Remark 3 (Typed quantification). Our experiments conducted with lFHOL's implementation in Zipperposition, described further below, suggests that lFHOL can indeed express predicate symbols and atomic propositions. Contrary to the discussion above, this observation is in fact suggesting the existence of a dedicated base type o for Booleans in lFHOL. This in turn raises the question

whether quantification over predicate variables, relations, and formulas are allowed in lFHOL. For example, assuming the existence of a Boolean type $o \in \mathrm{BT}$, the formulas

$$\forall(p : \iota \to o).\, \exists(q : \iota \to o).\, \forall(x : \iota).\, (p\ x) \longleftrightarrow \neg(q\ x)$$

$$\exists(p : o).p \vee \neg p$$

are syntactically unproblematic regarding the lFHOL core restriction that formulas may not be nested under terms. ⌟

2.2 Semantics

A lFHOL interpretation $\mathcal{M}$ is a tuple $\mathcal{M} = (\mathcal{U}, \mathcal{E}, \mathcal{I})$ where

(i) $\mathcal{U}$ is a typed-indexed family of non-empty sets $\mathcal{U}_\tau$ (called *domains*),
(ii) $\mathcal{E}$ is a family of functions $\mathcal{E}_{\tau,\nu} : \mathcal{U}_{\tau\to\nu} \to (\mathcal{U}_\tau \to \mathcal{U}_\nu)$, and
(iii) $\mathcal{I}$ is a function that maps each constant symbol $\mathrm{f} : \overline{\tau}_n \Rightarrow \tau \in \Sigma$ to an element $\mathcal{I}(\mathrm{f})$ of $\mathcal{U}_{\overline{\tau}_n} \to \mathcal{U}_\tau$.

Note that lFHOL domains $\mathcal{U}_\tau$ do not contain set-theoretic functions, but rather some objects that are assigned a applicative behaviour by $\mathcal{E}$. The interpretation function $\mathcal{I}$, in contrast, assigns proper functions to function symbols with mandatory arguments. The introduced semantical structures thus share some similarities with so-called *general pre-Henkin structures* (or applicative structures) that generalize Henkin models, in which the function domains only contain proper functions. This transition from functions to applicative structures is needed to support the intended non-extensional base setting of lFHOL, in which functional extensionality is considered optional but not mandatory (for a similar construction see [6]).

A valuation ξ is a function that maps variables $x : \tau$ to elements of $\mathcal{U}_\tau$. Given an lFHOL interpretation $\mathcal{M} : (\mathcal{U}, \mathcal{E}, \mathcal{I})$ and a valuation ξ, the denotation $[\![t]\!]^\xi_\mathcal{M}$ of a term t is defined as follows:

(i) $[\![x]\!]^\xi_\mathcal{M} := \xi(x)$
(ii) $[\![\mathrm{f}(\overline{t}_n)]\!]^\xi_\mathcal{M} := \mathcal{I}(\mathrm{f})\big([\![\overline{t}_n]\!]^\xi_\mathcal{M}\big)$
(iii) $[\![s\ t]\!]^\xi_\mathcal{M} := \mathcal{E}\big([\![s]\!]^\xi_\mathcal{M}\big)\big([\![t]\!]^\xi_\mathcal{M}\big)$

The truth-value $[\![\phi]\!]^\xi_\mathcal{M} \in \{0, 1\}$ of a lFHOL formula ϕ is defined "as in first-order logic". Although not explicitly stated, we assume that this means that the following identities hold (in addition to the evaluation rules of quantified formulas):

(i) $[\![s \approx t]\!]^\xi_\mathcal{M} := \begin{cases} 1 & \text{if } [\![s]\!]^\xi_\mathcal{M} = [\![t]\!]^\xi_\mathcal{M} \\ 0 & \text{otherwise} \end{cases}$

(ii) $[\![\neg\phi]\!]^\xi_\mathcal{M} := \begin{cases} 1 & \text{if } [\![\phi]\!]^\xi_\mathcal{M} = 0 \\ 0 & \text{otherwise} \end{cases}$

(iii) $[\![\phi \wedge \psi]\!]^{\xi}_{\mathcal{M}} := \begin{cases} 1 & \text{if } [\![\phi]\!]^{\xi}_{\mathcal{M}} = 1 \text{ and } [\![\psi]\!]^{\xi}_{\mathcal{M}} = 1 \\ 0 & \text{otherwise} \end{cases}$

(iv) etc.

A formula ϕ is true in $\mathcal{M} \equiv (\mathcal{U}, \mathcal{E}, \mathcal{I})$ under assignment ξ, written $\mathcal{M}, \xi \models \phi$ if and only if $[\![\phi]\!]^{\xi}_{\mathcal{M}} \equiv 1$.

Remark 4. The choice of the term "λ-free higher-order logic" can be criticised as suboptimal. From the definition of semantics as just presented it becomes clear that the key aspect is that no comprehension principles are *a priori* assumed. An alternative name would thus have been "comprehension-free higher-order logic". This is also because syntactical variants of higher-order logic with comprehension but without λ-notation are already existing in the philosophy literature [13]. Choosing the term "λ-free higher-order logic" to characterise a comprehension-free higher-order logic with λ-notation, could thus lead to some avoidable confusion in interdisciplinary communication. ⌟

Remark 5. Also the "higher-order" part in "λ-free higher-order logic" is at least debatable in a setting without any comprehension principles. Philosophical logicians, for example, state that it is exactly these principles that make a logic higher-order (or second-order, for that matter). As an example, Shapiro defines the broad range of higher-order logics as follows:

"*A [...] language is called second-order or higher-order if it has variables that range over relations, propositional functions, properties [...]. In any case, the distinguishing feature of second- or higher-order languages is not so much the nature of the individual items that fall in the range of the extra variables, but rather their extension or totality.*" [12].

Here, the focus is on the nature of the domains the quantifiers range over. Enderton argues that "*[...] using general pre-structures amounts to treating a second-order language as a many-sorted first-order language.*" [8, §3]. This argument generalizes to finite-order (higher-order) languages. ⌟

3 Discussion

This section summarises our example driven experiments and assessments of the lFHOL approach. The motivation of our work was to gain clarity about several questions that we could not precisely answer from the dense presentation in [3]. The particular questions addressed are highlighted in gray boxes. The examples we present here point to some questionable and partly counter-intuitive reasoning patterns that, as we believe, require attention by users and implementors of the logic.

For the experiments we used the Zipperposition reasoner[5], which implements all four different calculi (extensional and intensional variants) presented in [3] in a sound and complete way.

All problem statements that are used for practically evaluating the following discussion points are presented in TPTP THF format in Appendix A.

3.1 Boolean Type

Is there a Boolean type or not?

As discussed in Sect. 2 before, lFHOL does not guarantee the existence of a dedicated Boolean type $o \in \mathrm{BT}$. At first sight this suggest that there are no atomic propositions, no predicate symbols and that quantification over relations, predicates, etc. is excluded. Even very simple formulas such as instances of the law of excluded middle can then not even be formulated and the question naturally arises why we should call the language a logic. However, from benchmark measurements [3] the contrary can be inferred. These benchmark example in particular suggest that (i) there is a type o of Booleans, and that (ii) quantifications over relations, predicates and other symbols whose result type is o is indeed covered by the approach. This is because the hand-selected benchmark set[6] used in [3] for evaluating lFHOL, resp. its sound and complete implementation in Zipperposition, does contain a number of TPTP THF problems that include exactly these syntactical constructs. At the same time this benchmark set explicitly excludes problems that contain formulas in proper subterm positions. For example, problem `SYO013^1` from the benchmark set in in [3] contains atomic propositions, `SET014^5` uses quantification over predicates, and `SYO019^1` and `SYO021^1` uses quantification over propositions.

Since these language features are used for evaluating lFHOL, we must conjecture that a Boolean type o does in fact exist and we therefore assume this in the remainder.

3.2 Comprehension

Does the syntactical presentation of a formula influence its validity?

In a logic setting with a many-sorted language including (higher-order) functions and quantification over symbols of functional type, the question quite naturally arises what the quantifiers actually range over. Very often in second-order and

[5] We use the exact same version of the Zipperposition prover (commit `7fe2ebe`) and the same run scripts as provided by Bentkamp et al. for their evaluation in [3], see `http://matryoshka.gforge.inria.fr/pubs/lfhosup_data/` for details. The prover was built and executed as explained there.

[6] See `http://matryoshka.gforge.inria.fr/pubs/lfhosup_data/list_THF.txt` for the full list of TPTP THF problems.

higher-order logics comprehension principles are assumed that clarify this questions. The following comprehension principle (an axiom schema, where P does not occur free in ϕ), for example, postulates the existence of predicates that can be defined in terms of formulas ϕ (with free variables $x_1 \dots x_n$) of the logical language:

$$\exists P.\, \forall x_1 \dots x_n.\, P\ x_1 \dots x_n \longleftrightarrow \phi$$

Comprehension thus augments the logic with some knowledge about the underlying semantical structure, reflecting that the semantical domain of functions is chosen to be a subset of the powerset of total functions between the respective types (the complete power set in the case of standard semantics).

Without comprehension axioms, second- and higher-order languages are essentially treated as a many-sorted first-order logic with additional syntactical means of quantifying about function symbols. This is because the set $\mathcal{U}_{\tau\to\nu}$, for some types τ and ν, is chosen to be *any subset* of set of total functions from $\mathcal{U}_\tau$ to $\mathcal{U}_\nu$, without the requirement that certain functions need to exist (modelled by $\mathcal{E}$ in the case of lFHOL). While this topic is certainly of theoretical interest, and quite a number of discussions exist in the field of philosophical logic, there are also consequences for the practical employment of such languages in reasoning.

As a running example, consider the following meta-logical specification of abbreviations LEQ and AEQ in lFHOL, also referred to as Leibniz equality and Andrews equality, respectively:

$$\begin{aligned}
&\mathrm{LEQ}[s_\tau, t_\tau] =_{def} \forall(p : \tau \to o).\, (p\ s) \longrightarrow (p\ t)\\
&\mathrm{AEQ}[s_\tau, t_\tau] =_{def} \forall(r : \tau \to \tau \to o).\, (\forall(z : \tau).\, r\ z\ z) \longrightarrow (r\ s\ t)
\end{aligned}$$

Note that LEQ and AEQ are meta-logical abbreviations and do not constitute logical symbols of the signature Σ. By convention, meta-logical symbols are written in uppercase letters in the following.

In contrast to classical higher-order, where Leibniz equality and Andrews equality coincide and additionally denote a proper equality under certain conditions [6], LEQ and AEQ do not coincide in lFHOL. In other words, the formula

$$\forall(x : \iota).\, \forall(y : \iota).\, \mathrm{LEQ}[x, y] \longleftrightarrow \mathrm{AEQ}[x, y] \tag{1}$$

is not a theorem of lFHOL. This is due to the fact that the functional domain $\mathcal{U}_{\iota\to\iota}$ might not contain enough elements to enforce this identity. As an example, consider the lFHOL model structure $\mathcal{M} = (\mathcal{U}, \mathcal{E}, \mathcal{I})$, where $\mathcal{U}_\iota = \{\mathfrak{a}, \mathfrak{b}\}, \mathfrak{a} \neq \mathfrak{b}$, $\mathcal{U}_{\iota\to o} = \{\mathrm{const}^\iota_\top, \mathrm{id}_\mathfrak{a}\}$ and $\mathcal{U}_{\iota\to\iota\to o} = \{\mathrm{const}^{\iota\iota}_\top\}$. Let furthermore $\mathcal{E}(\mathrm{const}^\iota_\top)(\mathfrak{a}) = \mathcal{E}(\mathrm{const}_\top)(\mathfrak{b}) = \top$, $\mathcal{E}(\mathrm{id}_\mathfrak{a})(x) = \top$ if and only if $x = \mathfrak{a}$, and $\mathcal{E}(\mathrm{const}^{\iota\iota}_\top)(x) = \mathrm{const}^\iota_\top$ for every $x \in \mathcal{U}_\iota$. It holds that $\mathcal{M} \not\models \forall(x : \iota).\, \forall(y : \iota).\, \mathrm{LEQ}[x, y] \longleftrightarrow \mathrm{AEQ}[x, y]$. Note that there does not exist an equivalent Henkin model for classical HOL, since the domain of relational symbols $\mathcal{U}_{\iota\to\iota\to o}$ does not contain enough functions to ensure that every λ-term is assigned a proper denotation.

Another formulation of the problem postulates two logical symbols, i.e. $\{\mathrm{leq} : \iota \to \iota \to o, \mathrm{aeq} : \iota \to \iota \to o\} \subseteq \Sigma$, that are assigned the properties of Leibniz

and Andrew equality, LEQ and AEQ, respectively via appropriate axioms:

$$\begin{aligned} &\forall(x:\iota).\,\forall(y:\iota).\,\mathrm{leq}\ x\ y \longleftrightarrow \mathrm{LEQ}[x,y] \\ &\forall(x:\iota).\,\forall(y:\iota).\,\mathrm{aeq}\ x\ y \longleftrightarrow \mathrm{AEQ}[x,y] \end{aligned} \tag{2}$$

The following lFHOL formula then states the equivalence between both relational concepts:

$$\forall(x:\iota).\,\forall(y:\iota).\,\mathrm{leq}\ x\ y \longleftrightarrow \mathrm{aeq}\ x\ y$$

Similar to the version above, this equivalence compares Leibniz and Andrews equality with the difference that now these concepts are represented by dedicated symbols. Surprisingly, this representional variant of the proposition becomes a theorem in lFHOL.[7] This is due to the fact that the semantical domains are now guaranteed to contain elements that can be used as witnesses within refutations. In other words, we have now formulas which intuitively are equivalent, but which are not so in the investigated logic.

Another example for comprehension issues is reasoning with equality itself. It seems odd to the authors that $\forall(x:\iota).x \approx x$ is a theorem of the system while $\exists(eq:\iota\to\iota\to o).\,\forall(x:\iota).\,eq\ x\ x$ is not, given that partial applications are supported by the language and that equality exists by design (on the level of formulas).

Whether the above examples constitute desired reasoning patterns remains an open question. For potential users of the logic the answer should clearly be as transparent as possible.

3.3 Proofs about the Booleans type

What is the interpretation of the (assumed) Boolean type?

Following the observations from §3.1, there seems to be a type of Booleans (otherwise we could not quantify over it). As a consequence, the type of Booleans is necessarily reflected by a dedicated domain $\mathcal{U}_o$ in the semantical level. In classical logics a domain of Booleans is usually bivalent, i.e. consisting of exactly two distinct semantical objects $\mathcal{U}_o = \{\mathfrak{t},\mathfrak{f}\}$, representing truth and falsehood, respectively.

In a bivalent classical logic with quantification over formulas, it should be provable that there exists truth, i.e. that $\exists(p:o).\,(p \longleftrightarrow \top)$ is a theorem. And indeed, this seems to be a theorem of lFHOL (seconded by Zipperposition). However, the dual assertion that there exists falsehood, $\exists(p:o).\,(p \longleftrightarrow \bot)$, cannot be proven in lFHOL. After negating, clausification and simplification yields the clause set $\{x \approx \top\}$ from which there seems to be no refutation in any of the four lFHOL calculi presented by Bentkamp et al.. In the first case (existence of truth), a refutation can be found using an instance of equality

[7] This is what our experiments with the sound and complete implementation of the lFHOL calculi in Zipperposition confirm.

resolution[8] which is not possible for positive literals. As a direct consequence, the formula $\exists(p : o).\, \exists(q : o).\, \neg(p \longleftrightarrow q)$ is not derivable in the calculi.

Now, one could argue that the semantics of lFHOL does not guarantee that the domain of Booleans is strictly bivalent and that falsehood needs to exist. However, the formula $\exists(p : o).\, \exists(q : o).\, \exists(r : o).\, \big(\neg(p \longleftrightarrow q) \wedge \neg(q \longleftrightarrow r) \wedge \neg(p \longleftrightarrow r)\big)$, stating that there exist three distinct objects of Boolean type, is unsatisfiable and Zipperposition quickly finds a refutation for this. In fact, both the assumptions that there exists only one Boolean object and that there exist three or more elements in the Boolean domain are unsatisfiable and can be refuted[9]. This, in contrary to the above, suggests that the Boolean domain is bivalent.

Surprisingly, when reformulating the bivalence conjecture using a bijection $\mathrm{f} : \iota \to o$ between the domain of individuals and the Boolean domain, the desired result can be proven in Zipperposition. This now raises the question whether the proof calculi are too weak to deduce all desired theorems (and are thus incomplete) or if bivalence is not supposed to be a theorem of the system (and the calculi are thus unsound). In any case, there should be a discussion about the originally intended semantics of the system.

3.4 Boolean Extensionality

What are the extensionality properties of relation types?

There seems to be an issue with full extensionality principles. Consider the running example of Leibniz and Andrews equality, in the variant with explicit logical symbols representing the concepts, cf. (2).

In an extensional setting, one would expect that if both concepts coincide point-wise, then both concepts are equal. In particular, as discussed in §3.2, we have that

$$\forall(x : \iota).\, \forall(y : \iota).\, \mathrm{leq}\ x\ y \longleftrightarrow \mathrm{aeq}\ x\ y$$

is a theorem of lFHOL (and there exists a refutation by Zipperposition). However, using both extensional calculi Zipperposition cannot prove

$$\mathrm{leq} \approx \mathrm{aeq}$$

and terminates with a saturated set. Under the assumption that Boolean types exist, a fully extensional semantics is also extensional for types containing Boolean-typed results[10] and therefore, the above equality should hold.

The authors conjecture that this might be the case because, in an extensional setting, lFHOL and Zipperposition depend on additional extensionality axioms which might be not properly instantiated for Boolean-type symbols.

[8] Zipperposition internally identifies non-equality predicates p with an equality to an explicit truth symbol, i.e. as in $p \approx \top$.

[9] This can be modeled by postulating the existence of an bijection between the type of individuals and the type of Booleans with respective size restrictions.

[10] Cf. the definition of extensional interpretations in [3].

3.5 Zipperposition

Why is Zipperposition not consistently answering the above questions?

While we argue that the aspects discussed further above are non-trivial for users of lFHOL and possible implementors of further systems based on that logic, our case is emphasized by the fact that the implementors of lFHOL reasoning in Zipperposition too did overlook some subtle consequences of its non-trivial semantics.

Pre-processing and comprehension. A popular pre-processing technique for reducing the size of the initial clause set during clausification is formula renaming [11], or in the context of HOL reasoning argument extraction in general [16]. Here, subformulas are replaced by fresh propositional symbols that are then, in a second step, logically linked to the replaced formula using additional axioms. While this technique can improve reasoning effectivity in first-order and higher-order logic, in the setting of lHFOL this technique threatens soundness.

Consider the statement of equality between Andrews and Leibniz equality without explicit symbols, cf. equation (1). As discussed before, formula (1) is counter-satisfiable in lFHOL. For the sake of the example, syntactical representation of the formula is changed as follows:

$$\forall(x:\iota).\forall(y:\iota).\big(\mathrm{LEQ}[x,y] \longleftrightarrow \mathrm{AEQ}[x,y]\big) \longleftrightarrow \top$$

This formula is arguably equivalent to (1), however Zipperposition classifies it as theorem in every of the four calculi. When inspecting the proof certificates provided by Zipperposition, it becomes apparent that this unexpected result is caused by illegitimate formula renaming during pre-processing. The original problem is internally transformed into the following problem statement (TPTP THF):

```
%%% introduced by pre-processing
thf(fresh1_type, type, fresh1: ($i>$o) > $i > $i > $o).
thf(fresh1, axiom,
  ![P:$i>$o,X:$i,Y:$i]: (
    (fresh1 @ P @ Y @ X) <=> ((P @ X) => (P @ Y)))).

thf(fresh2_type, type, fresh2: ($i>$i>$o) > $i > $i > $o).
thf(fresh2, axiom,
  ![R:$i>$i>$o,X:$i,Y:$i]: (
    (fresh2 @ R @ Y @ X) <=> ((![Z:$i]: (R @ Z @ Z)) => R @ X @ Y))).

%%% transformed conjecture
thf(c, conjecture, (
  (![X:$i,Y:$i]: (
        (![P:$i>$o]: (fresh1 @ P @ Y @ X))
    <=> (![R:$i>$i>$o]: (fresh2 @ R @ Y @ X)))
  ) <=> $true
)).
```

This problem, however, resembles the second variant discussed in §3.2 which has, due to lack comprehension of principles, a different semantics.

It is an intriguing question whether there are more pre-processing techniques where such a behaviour occurs and which, as a consequence, can not be employed within lFHOL provers.

Extensionality. After reading the input problem, Zipperposition transforms equalities $a \approx b$ between objects of Booleans type into equivalences $a \longleftrightarrow b$. As the language of lFHOL allows equalities between terms only and it is not clear whether Boolean terms qualify for this, this might not be an error in a strict sense. However, we argue that this is hardly desired behaviour, as such an transformation easily introduces unsoundness when reasoning in an intensional setting. As an example, in the evaluation of lFHOL and Zipperposition [3] the problem `SYO015^1` is used as a benchmark. This problem conjectures the inequality $\mathrm{p} \not\approx \neg\mathrm{p}$ between a formula p and its negation, which is not a theorem in an intensional setting but a theorem in an extensional logic. Using the above transformation, Zipperposition finds a prove in all four calculi, including the intensional variants. This behaviour is in a way opposing the approach of handling extensionality as presented in §3.4.

4 Summary and Further Work

Our objective has been to rationally reconstruct and assess the language and logic of λ-free higher-order logic as introduced in [3]. We have been unsuccessful in the sense that we could not reach a satisfying point of clarification. Is lFHOL really intended to exclude very basic statements such as "rains $\vee$ $\neg$rains" or "tall x" from being expressable, and if so, what justifies the use of term "logic"? If in contrast the exclusion of such basic logical expressions is not intended it remains unclear, for example, whether certain (restricted) comprehension principles are silently assumed in lFHOL and if Boolean extensionality is addressed. Also a thorough study of Zipperposition, the de-facto reference implementation of lFHOL, did not help to clarify the raised questions. In fact, several answers that were provided by the system distorted the picture even more.

We think that a detailed explication of lFHOL's language restrictions and its semantics is required, and that the logics implementation in Zipperposition should be precisely aligned with it. This is particularly true since lFHOL, in our opinion, *is* a very promising starting and reference point for further work on the lifting the superposition approach from first-order logic to higher-order logic. However, there should be a solid and easy accessible semantical basis from which such a development can progress carefully. To reuse the logic and optimally gain from its achievements, the situation as described in this paper, in contrast, is kind of unsatisfying. In particular, any project interested in integrating Zipperposition as a trusted black box in other applications or tools, should rather be careful at the current state of development. Moreover, there is already ongoing work on implementing further theorem proving systems based on [3], including a lFHOL-version of the E prover [14]. Further work could try to clarify some of the questions raised here empirically by conducting further experiments with such emerging alternative implementations. However, a preferred solution is if the presented questions are answered in an update version of [3] that provides a precise and easy accessible language and semantics definition for lFHOL. This

way it will be possible to identify the apparent differences between the intended semantics lFHOL and its implementation in Zipperposition.

References

1. Andrews, P.: Church's type theory. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, summer 2018 edn. (2018)
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994). https://doi.org/10.1093/logcom/4.3.217
3. Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for Lambda-Free Higher-Order Logic. In: IJCAR. Lecture Notes in Computer Science, vol. 10900, pp. 28–46. Springer (2018). https://doi.org/10.1007/978-3-319-94205-6_3
4. Benzmüller, C.: Higher-order automated theorem provers. In: Delahaye, D., Woltzenlogel Paleo, B. (eds.) All about Proofs, Proof for All, pp. 171–214. Mathematical Logic and Foundations, College Publications, London, UK (2015)
5. Benzmüller, C.: Universal (meta-)logical reasoning: Recent successes. Science of Computer Programming **172**, 48–62 (2019). https://doi.org/10.1016/j.scico.2018.10.008
6. Benzmüller, C., Brown, C., Kohlhase, M.: Higher-order semantics and extensionality. Journal of Symbolic Logic **69**(4), 1027–1088 (2004). https://doi.org/10.2178/jsl/1102022211
7. Cruanes, S.: Extending superposition with integer arithmetic, structural induction, and beyond. Ph.D. thesis, École polytechnique (2015)
8. Enderton, H.B.: Second-order and higher-order logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, fall 2015 edn. (2015)
9. Miller, D.A.: Proofs in higher-order logic. Ph.D. thesis, Carnegie Mellon University (1983)
10. Miller, D.A.: A compact representation of proofs. Studia Logica **46**(4), 347–370 (1987). https://doi.org/10.1007/BF00370646
11. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Handbook of Automated Reasoning, pp. 335–367. Elsevier and MIT Press (2001)
12. Shapiro, S.: Second-order logic, foundations, and rules. The Journal of philosophy **87**(5), 234–261 (1990)
13. Van Benthem, J., Doets, K.: Higher-order logic. In: Handbook of philosophical logic, pp. 189–243. Springer (2001)
14. Vukmirović, P., Blanchette, J.C., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019). LNCS (2019), to be published
15. Wiedijk, F. (ed.): The Seventeen Provers of the World, Foreword by Dana S. Scott, Lecture Notes in Computer Science, vol. 3600. Springer (2006). https://doi.org/10.1007/11542384
16. Wisniewski, M., Steen, A., Kern, K., Benzmüller, C.: Effective normalization techniques for HOL. In: IJCAR. Lecture Notes in Computer Science, vol. 9706, pp. 362–370. Springer (2016). https://doi.org/10.1007/978-3-319-40229-1_25

A Experiment Summary

Problem	Intentional Non-purifying	Intentional Purifying	Extensional Non-purifying	Extensional Purifying
SYO013^1	✓[1]	✓[1]	✓	✓
SET014^5	✓	✓	✓	✓
SYO019^1	✓	✓	✓	✓
SYO021^1	✓[1]	✓[1]	✓	✓
leq-aeq	×	×	×	×
leq-aeq-variant	✓	✓	✓	✓
leq-aeq-symbols	✓	✓	✓	✓
leq-aeq-symbols-ext	×	×	×	×
truth-exists	✓	✓	✓	✓
falsehood-exists	×	×	†	†
two-different-bools	×	×	†	†
three-different-bools	✓	✓	✓	✓
one-boolean	✓	✓	✓	✓
more-booleans	✓	✓	✓	✓
exactly-two-bools	✓	✓	✓	✓
SYO015^1	✓[1]	✓[1]	✓	✓

✓: Theorem/Unsatisfiable
×: Saturated set (GaveUp)
†: Proof search diverges

Results with superscript [1] are not consistent with classical HOL semantics.

Problem Statments

Listing 1.1. SYO013^1 containing atomic propositions and predicate symbols.

```
thf(a_type,type,(
    a: $o )).

thf(b_type,type,(
    b: $o )).

thf(p_type,type,(
    p: $o > $o )).

thf(conj,conjecture,
    ( ( a
      & b
      & ( p @ a ) )
   => ( p @ b ) )).
```

Listing 1.2. SET014^5 containing quantification over predicates.

```
thf(a_type,type,(
    a: $tType )).

thf(cBOOL_PROP_32_pme,conjecture,(
    ! [X: a > $o,Y: a > $o,Z: a > $o] :
```

```
      ( ( ! [Xx: a] :
            ( ( X @ Xx )
           => ( Z @ Xx ) )
        & ! [Xx: a] :
            ( ( Y @ Xx )
           => ( Z @ Xx ) ) )
     => ! [Xx: a] :
          ( ( ( X @ Xx )
            | ( Y @ Xx ) )
         => ( Z @ Xx ) ) ) )).
```

Listing 1.3. SYO019^1 containing quantification over formulas.

```
thf(conj,conjecture,(
    ! [X: $o,Y: $o] :
      ( ( X
        & Y )
    <=> ~ ( ~ ( X )
          | ~ ( Y ) ) ) )).
```

Listing 1.4. SYO021^1 containing quantification over formulas and equality between Booleans.

```
thf(conj,conjecture,(
    ! [X: $o,Y: $o] :
      ( ( X
        & Y )
      = ( ~ ( ~ ( X )
            | ~ ( Y ) ) ) ) )).
```

Listing 1.5. leq-aeq stating the equivalence of LEQ and AEQ without explicit symbols.

```
thf(conj, conjecture,
     ![X:$i,Y:$i]: (
       (![P:$i>$o]: ((P @ X) => (P @ Y)))
        <=>
       (![R:$i>$i>$o]: (
                     (![Z:$i]: (R @ Z @ Z)) => (R @ X @ Y))))).
```

Listing 1.6. leq-aeq-symbols stating the equivalence of LEQ and AEQ using explicit symbols leq and aeq, respectively.

```
thf(leq_type, type, leq: $i>$i>$o).
thf(leq_def, axiom,
  ![X:$i,Y:$i]: (
    (leq @ X @ Y) <=> (![P:$i>$o]: ((P @ X) => (P @ Y)))) ).
thf(aeq_type, type, aeq: $i>$i>$o).
thf(aeq_def, axiom,
  ![X:$i,Y:$i]: (
    (aeq @ X @ Y) <=>
       (![R:$i>$i>$o]: (
           (![Z:$i]: (R @ Z @ Z)) => (R @ X @ Y)))) ).

thf(conj, conjecture,
         ![X:$i,Y:$i]: (
          (leq @ X @ Y) <=> (aeq @ X @ Y))).
```

Listing 1.7. leq-aeq-symbols-ext stating the equality of the symbols leq and aeq.

```
thf(leq_type, type, leq: $i>$i>$o).
thf(leq_def, axiom,
  ![X:$i,Y:$i]: (
```

```
    (leq @ X @ Y) <=> (![P:$i>$o]: ((P @ X) => (P @ Y)))) ).
thf(aeq_type, type, aeq: $i>$i>$o).
thf(aeq_def, axiom,
  ![X:$i,Y:$i]: (
    (aeq @ X @ Y) <=>
       (![R:$i>$i>$o]: (
           (![Z:$i]: (R @ Z @ Z)) => (R @ X @ Y)))) ).

thf(conj, conjecture, leq = aeq).
```

Listing 1.8. `truth-exists` stating the existence of a true formula.

```
thf(conj, conjecture, ?[P:$o]: (P <=> $true)).
```

Listing 1.9. `falsehood-exists` stating the existence of a false formula.

```
thf(conj, conjecture, ?[P:$o]: (P <=> $false)).
```

Listing 1.10. `two-different-bools` stating the existence of two non-equivalent formulas.

```
thf(conj, conjecture, ?[P:$o,Q:$o]: (~(P <=> Q))).
```

Listing 1.11. `three-different-bools` stating the existence of two non-equivalent formulas.

```
thf(conj, conjecture, ?[P:$o,Q:$o,R:$o]: (~(P <=> Q) & ~(Q <=> R) & ~(P <=> R))).
```

Listing 1.12. `one-boolean` stating the existence of only one Boolean object.

```
% Only one individual
thf(a,type, a:$i).
thf(exactlyOne, axiom, ![X:$i]: (X = a)).

% Bijection into Booleans possible?
thf(f, type, f:$i>$o).
thf(surjection, axiom, (
      (?[X:$i]: (f @ X))
    & (?[X:$i]: (~(f @ X)) )).
thf(injection, axiom, (
  ![X:$i,Y:$i]: (((f @ X) <=> (f @ Y)) => (X = Y))
)).
```

Listing 1.13. `more-booleans` stating the existence of three or more Boolean objects.

```
% At least three distinct individuals
thf(a,type, a:$i).
thf(b,type, b:$i).
thf(c,type, c:$i).
thf(distinct, axiom, (a != b) & (b != c) & (a != c)).

% Bijection into Booleans possible?
thf(f, type, f:$i>$o).
thf(surjection, axiom, (
      (?[X:$i]: (f @ X))
    & (?[X:$i]: (~(f @ X)) )).
thf(injection, axiom, (
  ![X:$i,Y:$i]: (((f @ X) <=> (f @ Y)) => (X = Y))
)).
```

Listing 1.14. `exactly-two-bools` stating the existence of exactly two distinct Boolean objects.

```
% Exactly two individuals
thf(a,type, a:$i).
thf(b,type, b:$i).
thf(distinct, axiom, (a != b)).
thf(exactly, axiom, ![X:$i]: ((X = a) | (X = b))).

% Are both sets of the same cardinality?
thf(f, type, f:$i>$o).
thf(surjection, axiom, (
      (?[X:$i]: (f @ X))
    & (?[X:$i]: (~(f @ X)) )).
thf(injection, conjecture, (
  ![X:$i,Y:$i]: (((f @ X) <=> (f @ Y)) => (X = Y))
)).
```

Listing 1.15. `leq-aeq-variant` stating the equivalence of LEQ and AEQ without explicit symbols (variant).

```
thf(c, conjecture, (
  (![X:$i,Y:$i]: (
        (![P:$i>$o]: ((P @ X) => (P @ Y)))
    <=> (![R:$i>$i>$o]: ((![Z:$i]: (R @ Z @ Z)) => (R @ X @ Y)))
  ) <=> $true
)).
```

Listing 1.16. `SY0015^1` stating the inequality of a formula and its negation.

```
thf(a,type,(
    a: $o )).

thf(conj,conjecture,(
    a != ( ~ ( a ) ) )).
```