Superposition for Lambda-Free Higher-Order Logic

Alexander Bentkamp, Simon Cruanes, Jasmin Christian Blanchette, Uwe Waldmann

To cite this version:

Alexander Bentkamp, Simon Cruanes, Jasmin Christian Blanchette, Uwe Waldmann. Superposition for Lambda-Free Higher-Order Logic. IJCAR 2018 - 9th International Joint Conference on Automated Reasoning, Jul 2018, Oxford, United Kingdom. hal-01904595

HAL Id: hal-01904595
https://hal.inria.fr/hal-01904595

Submitted on 25 Oct 2018

Superposition for Lambda-Free Higher-Order Logic

Alexander Bentkamp[1(✉)], Jasmin Christian Blanchette[1,2,3],
Simon Cruanes[4,3], and Uwe Waldmann[2]

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
a.bentkamp@vu.nl
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
[3] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
[4] Aesthetic Integration, Austin, Texas, USA

Abstract. We introduce refutationally complete superposition calculi for intentional and extensional λ-free higher-order logic, two formalisms that allow partial application and applied variables. The calculi are parameterized by a term order that need not be fully monotonic, making it possible to employ the λ-free higher-order lexicographic path and Knuth–Bendix orders. We implemented the calculi in the Zipperposition prover and evaluated them on TPTP benchmarks. They appear promising as a stepping stone towards complete, efficient automatic theorem provers for full higher-order logic.

1 Introduction

Superposition is a highly successful calculus for reasoning about first-order logic with equality. We are interested in *graceful* generalizations to higher-order logic: calculi that, as much as possible, coincide with standard superposition on first-order problems and that scale up to arbitrary higher-order problems.

As a stepping stone towards full higher-order logic, in this paper we restrict our attention to a λ-free fragment of higher-order logic that supports partial application and application of variables (Section 2). This formalism is expressive enough to permit the axiomatization of higher-order combinators such as $\mathsf{pow}_\tau : \mathit{nat} \to (\tau \to \tau) \to \tau \to \tau$:

$$\mathsf{pow}\ 0\ h \approx \mathsf{id} \qquad\qquad \mathsf{pow}\ (\mathsf{S}\ n)\ h\ x \approx h\ (\mathsf{pow}\ n\ h\ x)$$

Conventionally, functions are applied without parentheses and commas, and variables are italicized. Notice the variable number of arguments to pow and the application of h. The expressiveness of full higher-order logic can be recovered by introducing SK-style combinators to represent λ-abstractions and proxies for the logical symbols [24, 32].

A widespread technique to support partial application and application of variables in first-order logic is to make all symbols nullary and to represent application of functions of type $\tau \to \upsilon$ by a family of binary symbols $\mathsf{app}_{\tau,\upsilon}$. Following this scheme, the higher-order term $\mathsf{f}\ (h\ \mathsf{f})$ is translated to $\mathsf{app}(\mathsf{f}, \mathsf{app}(h, \mathsf{f}))$, which can be processed by first-order methods. We call this the *applicative encoding.* The existence of such a reduction explains why λ-free higher-order terms are also called “applicative first-order terms.” Unlike for full higher-order logic, most general unifiers are unique for our λ-free fragment, just as they are for applicatively encoded first-order terms.

Although the applicative encoding is complete [24] and is employed fruitfully in tools such as Sledgehammer [9,27], it suffers from a number of weaknesses, all related to its gracelessness. Transforming all the function symbols into constants considerably restricts what can be achieved with term orders; for example, argument tuples cannot be compared using different methods for different symbols. In a prover, the encoding also clutters the data structures, slows down the algorithms, and neutralizes the heuristics that look at the terms' root symbols. But our chief objection is the sheer clumsiness of encodings and their poor integration with interpreted symbols. And they quickly accumulate; for example, using the traditional encoding of polymorphism relying on a distinguished binary function symbol t [8, Section 3.3] in conjunction with the applicative encoding, the term $\mathsf{S}\;x$ becomes $\mathsf{t}(\mathsf{nat}, \mathsf{app}(\mathsf{t}(\mathsf{fun}(\mathsf{nat}, \mathsf{nat}), \mathsf{S}), \mathsf{t}(\mathsf{nat}, x)))$.

Hybrid schemes have been proposed to strengthen the applicative encoding: If a given symbol always occurs with at least k arguments, these can be passed directly [27]. However, this relies on a closed-world assumption: that all terms that will ever be compared arise in the input problem. This noncompositionality conflicts with the need for complete higher-order calculi to synthesize arbitrary terms during proof search [6]. As a result, hybrid encodings are not an ideal basis for higher-order automated reasoning. Instead, we propose to generalize the superposition calculus to *intensional* and *extensional* λ-free higher-order logic. In the extensional version of the logic, the property $(\forall x.\; h\,x \approx k\,x) \longrightarrow h \approx k$ holds for all functions h, k of the same type. For each logic, we present two calculi (Section 3). The intentional calculi perfectly coincide with standard superposition on first-order clauses; the extensional calculi depend on an extra axiom.

Superposition is parameterized by a term order, which prunes the search space. If we assume that the term order is a simplification order enjoying totality on ground terms, the standard calculus rules and completeness proof can be lifted verbatim. The only necessary changes concern the basic definitions of terms and substitutions. However, there is one monotonicity property that is hard to obtain unconditionally: *compatibility with arguments*. It states that $s' \succ s$ implies $s'\;t \succ s\;t$ for all terms s, s', t such that $s\;t$ and $s'\;t$ are well typed. We recently introduced graceful generalizations of the lexicographic path order (LPO) [11] and the Knuth–Bendix order (KBO) [3] with argument coefficients, but they both lack this property. For example, given a KBO with $\mathsf{g} \succ \mathsf{f}$, it may well be that $\mathsf{g}\;\mathsf{a} \prec \mathsf{f}\;\mathsf{a}$ if f has a large enough multiplier on its argument.

Our calculi are designed to be refutationally complete for such nonmonotonic orders (Section 4). To achieve this, they include an inference rule for argument congruence, which derives $C \vee s\,x \approx t\,x$ from $C \vee s \approx t$. The redundancy criterion is defined in such a way that the larger, derived clause is not subsumed by the premise. In the completeness proof, the most difficult case is the one that normally excludes superposition at or below variables using the induction hypothesis. With nonmonotonicity, this approach no longer works, and we propose two alternatives: Perform some superposition inferences onto higher-order variables, or "purify" the clauses to circumvent the issue. We refer to the corresponding calculi as *nonpurifying* and *purifying*. Detailed proofs are included in a technical report [5], together with more explanations and examples.

The calculi are implemented in the Zipperposition prover [17] (Section 5). We evaluate them on TPTP benchmarks [39,40] and compare them with the applicative encoding (Section 6). We find that there is a substantial cost associated with the applicative encoding and that the nonmonotonicity is not particularly expensive.

2 Logic

Refutational completeness of calculi for higher-order logic (also called simple type theory) is usually stated with respect to Henkin semantics [6, 22], in which the universes used to interpret functions need only contain the functions that can be expressed as terms. Since the terms of λ-free higher-order logic exclude λ-abstractions, in "λ-free Henkin semantics" the universes interpreting functions can be even smaller. Unlike other higher-order logics, there are no comprehension principles, and we disallow nesting of Boolean formulas inside terms, as a convenient intermediate step on our way towards full higher-order logic.

Problematically, in a logic with applied variables but without Hilbert choice, skolemization is unsound, unless we make sure that Skolem symbols are suitably applied [28]. We achieve this using a *hybrid logic* that supports both mandatory (uncurried) and optional (curried) arguments. Thus, if symbol sk takes two mandatory and one optional arguments, $\mathsf{sk}(x,y)$ and $\mathsf{sk}(x,y)\;z$ are valid terms. Nevertheless, as in our earlier work [3, 11], we use the adjective "graceful" in the strong sense that we can exploit optional arguments, identifying the first-order term $\mathsf{f}(x,y)$ with the curried higher-order term $\mathsf{f}\;x\;y$.

A type τ, υ of λ-free higher-order logic is either an element ι of a fixed set of atomic types or a function type $\tau \to \upsilon$ of functions from type τ to type υ. In our hybrid logic, a type declaration for a symbol is an expression of the form $\bar{\tau}_n \Rightarrow \tau$ (or simply τ if $n = 0$). We write $\bar{a}_n$ or $\bar{a}$ to abbreviate the tuple $(a_1, \ldots, a_n)$ or product $a_1 \times \cdots \times a_n$, for $n \geq 0$.

We fix a set $\mathcal{V}$ of typed variables, denoted by $x : \tau$ or x. A signature consists of a nonempty set Σ of symbols with type declarations, written as $\mathsf{f} : \bar{\tau} \Rightarrow \tau$ or f. We reserve the letters s, t, u, v for terms and x, y, z for variables and write $: \tau$ to indicate their type. The set of λ-free higher-order terms $\mathcal{T}_\Sigma^X$ over X is defined inductively. Every variable in $X \subseteq \mathcal{V}$ is a term. If $\mathsf{f} : \bar{\tau}_n \Rightarrow \tau$ and $u_i : \tau_i$ for all $i \in \{1, \ldots, n\}$, then $\mathsf{f}(\bar{u}_n) : \tau$ is a term. If $t : \tau \to \upsilon$ and $u : \tau$, then $t\;u : \upsilon$ is a term, called an *application*. Non-application terms ζ are called *heads*. Terms can be decomposed in a unique way as a head ζ applied to zero or more arguments: $\zeta\;s_1 \ldots s_n$ or $\zeta\;\bar{s}_n$ (abusing notation). Substitution and unification are generalized in the obvious way, without the complexities associated with λ-abstractions; for example, the most general unifier of $x\;\mathsf{b}\;z$ and $\mathsf{f}\;\mathsf{a}\;y\;\mathsf{c}$ is $\{x \mapsto \mathsf{f}\;\mathsf{a}, y \mapsto \mathsf{b}, z \mapsto \mathsf{c}\}$, and that of $h\;(\mathsf{f}\;\mathsf{a})$ and $\mathsf{f}\;(h\;\mathsf{a})$ is $\{h \mapsto \mathsf{f}\}$.

Formulas φ, ψ are of the form $\bot$, $\top$, $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \longrightarrow \psi$, $t \approx_\tau s$, $\forall x.\,\varphi$, or $\exists x.\,\varphi$, where t, s are terms and x is a variable. We let $s \not\approx t$ abbreviate $\neg\; s \approx t$. We normally view equations $s \approx t$ as unordered pairs and clauses as multisets of such (dis)equations.

Loosely following Fitting [20], an *interpretation* $\mathcal{I} = (\mathcal{U}, \mathcal{E}, \mathcal{J})$ consists of a type-indexed family of nonempty sets $\mathcal{U}_\tau$, called *universes*; a family of functions $\mathcal{E}_{\tau,\upsilon} : \mathcal{U}_{\tau\to\upsilon} \to (\mathcal{U}_\tau \to \mathcal{U}_\upsilon)$, one for each pair of types τ, υ; and a function $\mathcal{J}$ that maps each symbol with type declaration $\bar{\tau}_n \Rightarrow \tau$ to an element of $\overline{\mathcal{U}}_{\tau_n} \to \mathcal{U}_\tau$. An interpretation is *extensional* if $\mathcal{E}_{\tau,\upsilon}$ is injective for all τ, υ. Both intensional and extensional logics are widely used. The semantics is *standard* if $\mathcal{E}_{\tau,\upsilon}$ is bijective. A *valuation* ξ is a function that maps variables $x : \tau$ to elements of $\mathcal{U}_\tau$.

For an interpretation $(\mathcal{U}, \mathcal{E}, \mathcal{J})$ and a valuation ξ, the denotation of a term is defined as follows: $[\![x]\!]_{\mathcal{I}}^{\xi} = \xi(x)$; $[\![\mathsf{f}(\bar{t})]\!]_{\mathcal{I}}^{\xi} = \mathcal{J}(\mathsf{f})([\![\bar{t}]\!]_{\mathcal{I}}^{\xi})$; $[\![s\;t]\!]_{\mathcal{I}}^{\xi} = \mathcal{E}([\![s]\!]_{\mathcal{I}}^{\xi})([\![t]\!]_{\mathcal{I}}^{\xi})$. The truth value $[\![\varphi]\!]_{\mathcal{I}}^{\xi} \in \{0, 1\}$ of a formula φ is defined as in first-order logic. The interpretation $\mathcal{I}$ is a model of φ, written $\mathcal{I} \models \varphi$, if $[\![\varphi]\!]_{\mathcal{I}}^{\xi} = 1$ for all valuations ξ.

3 The Inference Systems

We introduce four versions of the superposition calculus, varying along two axes: intentional versus extensional, and nonpurifying versus purifying. To avoid repetitions, our presentation unifies them into a single framework.

3.1 The Inference Rules

The calculi are parameterized by a partial order $\succ$ on terms that is well founded, total on ground terms, and stable under substitutions and that has the subterm property. It must also be *compatible with function contexts*, meaning that $t' \succ t$ implies both $\mathsf{f}(\bar{s}, t', \bar{u})\ \bar{v} \succ \mathsf{f}(\bar{s}, t, \bar{u})\ \bar{v}$ and $s\ t'\ \bar{u} \succ s\ t\ \bar{u}$. On the other hand, it need not be *compatible with optional arguments*: $s' \succ s$ need not imply $s'\ t \succ s\ t$. Function contexts are built around *argument subterms*, defined as the reflexive transitive closure of the relation inductively specified by $\mathsf{f}(\bar{s})\ \bar{t} \rhd s_i$ and $\zeta\ \bar{t} \rhd t_i$ for all i. We write $s\langle u\rangle$ to indicate that the subterm u of $s[u]$ is an argument subterm. For example, f and $\mathsf{f\ a}$ are subterms of $\mathsf{f\ a\ b}$, but not argument subterms. The literal and clause orders are defined as multiset extensions in the usual way.

Literal selection is supported. The selection function maps each clause C to a subclause of C consisting of negative literals. A literal L is (*strictly*) *eligible* in C if it is selected in C or there are no selected literals in C and L is (strictly) maximal in C.

We start with the **extensional nonpurifying** calculus, which consists of five rules:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee [\neg]\, s\langle u\rangle \approx s'}^{C}}{(D' \vee C' \vee [\neg]\, s\langle t'\rangle \approx s')\sigma}\,\textsc{Sup} \qquad \frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\,\textsc{EqFact}$$

$$\frac{C' \vee s \not\approx s'}{C'\sigma}\,\textsc{EqRes} \qquad \frac{C' \vee s \approx s'}{C' \vee s\ \bar{x} \approx s'\ \bar{x}}\,\textsc{ArgCong} \qquad \frac{C' \vee s\ \bar{x} \approx s'\ \bar{x}}{C' \vee s \approx s'}\,\textsc{PosExt}$$

In the first three rules, σ denotes the most general unifier of the two grayed terms. For Sup, we assume that D's and C's variables have been standardized apart. For Sup, EqFact, and EqRes, the following standard order conditions apply on the premises after the application of σ: The last literal in each premise is eligible and even strictly eligible for positive literals of Sup. For the last literal of each premise of Sup and the last two literals of the premise of EqFact, the left-hand sides are not smaller than or equal to ($\not\preceq$) the respective right-hand sides. For Sup, $C\sigma \not\preceq D\sigma$.

Definition 1. A term of the form $x\ \bar{s}_n$, for $n \geq 0$, *jells* with a literal $t \approx t' \in D$ if $t = \tilde{t}\ \bar{y}_n$ and $t' = \tilde{t}'\ \bar{y}_n$ for some $\tilde{t}$, $\tilde{t}'$ and distinct variables $\bar{y}_n$ that do not occur elsewhere in D.

We add the following *variable condition* as a side condition to Sup, to further prune the search space, using the naming convention from Definition 1 for $\tilde{t}'$:

> If u has a variable head x and jells with the literal $t \approx t' \in D$, there must exist a ground substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C[x \mapsto \tilde{t}']$.

This condition generalizes the standard condition that $u \notin \mathcal{V}$. The two coincide if C is first-order. In some cases involving nonmonotonicity, the variable condition effectively mandates Sup inferences at variable positions, but never below.

The last two rules are nonstandard. For ArgCong, $s \approx s'$ must be strictly eligible in the premise, and $\bar{x}$ is a tuple of fresh variables. For PosExt, $s\ \bar{x} \approx s'\ \bar{x}$ must be strictly eligible in the premise, and $\bar{x}$ is a tuple of distinct variables that occur nowhere else in the premise. Furthermore, for every function type $\tau \to \upsilon$ occurring in the input problem, we introduce a Skolem symbol $\mathsf{diff}_{\tau,\upsilon} : (\tau \to \upsilon)^2 \Rightarrow \tau$ characterized by the following *extensionality axiom*: $h\ (\mathsf{diff}(h,k)) \not\approx k\ (\mathsf{diff}(h,k)) \lor h \approx k$.

The second calculus is the **intensional nonpurifying** variant. We obtain it by removing the PosExt rule and the extensionality axiom and by replacing the variable condition with "if $u \in \mathcal{V}$, there exists a ground substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C[u \mapsto t']\sigma\theta$." For monotone term orders, this condition amounts to $u \notin \mathcal{V}$.

By contrast, the purifying calculi never perform superposition at variables. Instead, they rely on purification [14, 35] (also called abstraction) to circumvent nonmonotonicity. The idea is to rename apart problematic occurrences of a variable x in a clause to $x_1, \ldots, x_n$ and to add *purification literals* $x_1 \not\approx x, \ldots, x_n \not\approx x$ to connect the new variables. We must then purify the initial clauses and all derived clauses.

In the **extensional purifying** calculus, the purification $\mathit{pure}(C)$ of clause C is defined as the result of the following iterative procedure. Consider the literals of C excluding those of the form $y \not\approx z$. If these literals contain both $x\ \bar{u}$ and $x\ \bar{v}$ as distinct argument subterms, replace all argument subterms $x\ \bar{v}$ with $x_i\ \bar{v}$, where x_i is fresh, and add the purification literal $x_i \not\approx x$. This calculus variant contains the PosExt rule and the extensionality axiom. The conclusion E of each rule is changed to $\mathit{pure}(E)$, except for PosExt, which preserves purity. Moreover, the variable condition is replaced by "either u has a non-variable head or u does not jell with the literal $t \approx t' \in D$."

In the **intensional purifying** calculus, we define $\mathit{pure}(C)$ iteratively as follows. Consider the literals of C excluding those of the form $y \not\approx z$. If these literals contain a variable x both applied and unapplied, replace all unapplied occurrences of x in C by a fresh variable x_i and add the purification literal $x_i \not\approx x$. We remove the PosExt rule and the extensionality axiom. The variable condition is replaced by "$u \notin \mathcal{V}$." The conclusion C of ArgCong is changed to $\mathit{pure}(C)$; the other rules preserve purity.

Finally, we impose some additional restrictions on literal selection. In the nonpurifying variants, a literal may not be selected if $x\ \bar{u}$ is a maximal term of the clause and the literal contains an argument subterm $x\ \bar{v}$ with $\bar{v} \neq \bar{u}$. In the extensional purifying calculus, a literal may not be selected if it contains a variable that is applied to different arguments in the clause. In the intensional purifying calculus, a literal may not be selected if the literal contains an unapplied variable that also appears applied in the clause.

3.2 Rationale for the Inference Rules

A key restriction of all four calculi is that they superpose only onto argument subterms, mirroring the requirement that the term order enjoy compatibility with function contexts. The ArgCong rule then makes it possible to simulate superposition onto non-argument subterms. However, in conjunction with the Sup rule, ArgCong can exhibit an unpleasant behavior, which we call *argument congruence explosion*:

$$\text{Sup}\ \dfrac{\text{ArgCong}\ \dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\ x \approx \mathsf{f}\ x} \quad h\ \mathsf{a} \not\approx \mathsf{b}}{\mathsf{f}\ \mathsf{a} \not\approx \mathsf{b}} \qquad \text{Sup}\ \dfrac{\text{ArgCong}\ \dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\ x\ y\ z \approx \mathsf{f}\ x\ y\ z} \quad h\ \mathsf{a} \not\approx \mathsf{b}}{\mathsf{f}\ x\ y\ \mathsf{a} \not\approx \mathsf{b}}$$

In both cases, the higher-order variable h is effectively the target of a SUP inference. Such derivations essentially amount to superposition at variable positions (as shown on the left) or even superposition below variable positions (as shown on the right), both of which can be extremely prolific. In standard superposition, the explosion is averted by the condition on the SUP rule that $u \notin \mathcal{V}$. In the extensional purifying calculus, the variable condition tests that either u has a non-variable head or u does not jell with the literal $t \approx t' \in D$, which prevents derivations such as the above. In the corresponding nonpurifying variant, some such derivations may need to be performed when the term order exhibits nonmonotonicity for the terms of interest.

In the intensional calculi, the explosion can arise even for monotonic orders, and it must be tamed by heuristics. The reason is connected to the absence of the POSEXT rule (which would be unsound). The variable condition in the extensional calculi is designed to prevent derivations such as those shown above, but since it only considers the shape of the clauses, it might also block SUP inferences whose side premises do not originate from ARGCONG. Consider a left-to-right LPO [11] instance with precedence $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, and consider the following unsatisfiable clause set:

$$\mathsf{g}\ (x\ \mathsf{b})\ x \approx \mathsf{a} \qquad\qquad \mathsf{g}\ (\mathsf{f}\ \mathsf{b})\ \mathsf{h} \not\approx \mathsf{a} \qquad\qquad \mathsf{h}\ x \approx \mathsf{f}\ x$$

The only possible inference from these clauses is POSEXT, showing its necessity. It is unclear whether POSEXT is necessary for the extensional purifying variant as well, but our completeness proof suggests that it is. Our proof also suggests that to achieve refutational completeness, due to nonmonotonicity, we need either to purify the clauses or to allow some superposition at variable positions, as mandated by the respective variable conditions. However, we have yet to find an example that demonstrates the necessity of these measures.

A significant advantage of our calculi over the use of standard superposition on applicatively encoded problems is the flexibility they offer in orienting equations. The following example gives two definitions of addition on Peano numbers:

$$\begin{array}{rr} \mathsf{add_L}\ 0\ y \approx y & \qquad\qquad \mathsf{add_R}\ x\ 0 \approx x \\ \mathsf{add_L}\ (\mathsf{S}\ x)\ y \approx \mathsf{add_L}\ x\ (\mathsf{S}\ y) & \qquad\qquad \mathsf{add_R}\ x\ (\mathsf{S}\ y) \approx \mathsf{add_R}\ (\mathsf{S}\ x)\ y \end{array}$$

Let $\mathsf{add_L}\ (\mathsf{S}^{100}\ 0)\ \mathsf{n} \not\approx \mathsf{add_R}\ \mathsf{n}\ (\mathsf{S}^{100}\ 0)$ be the negated conjecture. With LPO, we can use a left-to-right comparison for $\mathsf{add_L}$'s arguments and a right-to-left comparison for $\mathsf{add_R}$'s arguments to orient all four equations from left to right. Then the negated conjecture can be simplified to $\mathsf{S}^{100}\ \mathsf{n} \not\approx \mathsf{S}^{100}\ \mathsf{n}$ by rewriting (demodulation), and $\bot$ can be derived with a single inference. If we use the applicative encoding instead, there is no instance of LPO or KBO that can orient both recursive equations from left to right. For at least one of the two sides of the negated conjecture, the rewriting is replaced by 100 SUP inferences, which is much less efficient, especially in the presence of additional axioms.

3.3 Redundancy Criterion

For our calculi, a redundant (or composite) clause cannot simply be defined as a clause whose ground instances are entailed by smaller ($\prec$) ground instances of existing clauses, because this would make all ARGCONG inferences redundant. Our solution is to base the redundancy criterion on a weaker ground logic in which argument congruence does not hold. This logic also plays a central role in our completeness proof, to reason about the nonmonotonicity emerging from the lack of compatibility with optional arguments.

The weaker logic is defined via an encoding $\lfloor\ \rfloor$ of ground hybrid λ-free higher-order terms into uncurried terms, with $\lceil\ \rceil$ as its inverse. Accordingly, we refer to clausal λ-free higher-order logic as the *ceiling logic* and to its weaker relative as the *floor logic*. Essentially, the encoding indexes each symbol occurrence with its argument count. Thus, $\lfloor \mathsf{f} \rfloor = \mathsf{f}_0$ and $\lfloor \mathsf{f}\ \mathsf{a} \rfloor = \mathsf{f}_1(\mathsf{a}_0)$. This is enough to disable argument congruence; for example, $\{\mathsf{f} \approx \mathsf{g},\ \mathsf{f}\ \mathsf{a} \not\approx \mathsf{g}\ \mathsf{a}\}$ is unsatisfiable, whereas its encoding $\{\mathsf{f}_0 \approx \mathsf{g}_0,\ \mathsf{f}_1(\mathsf{a}_0) \not\approx \mathsf{g}_1(\mathsf{a}_0)\}$ is satisfiable. For clauses built from fully applied ground terms, the two logics are isomorphic, as we would expect from a graceful generalization.

Given a signature Σ in the ceiling logic, we define a signature $\Sigma^{\downarrow}$ in the floor logic as follows. For each higher-order type τ, we introduce an atomic type $\lfloor\tau\rfloor$ in the floor logic. For each symbol $\mathsf{f} : \bar{\tau}_k \Rightarrow \tau_{k+1} \to \cdots \to \tau_n \to \upsilon$ in Σ, where υ is atomic, we introduce symbols $\mathsf{f}_m : \lfloor\bar{\tau}_m\rfloor \Rightarrow \lfloor\tau_{m+1} \to \cdots \to \tau_n \to \upsilon\rfloor$ for $m \in \{k,\ldots,n\}$. The translation of ground terms is given by $\lfloor \mathsf{f}(\bar{u}_k)\ u_{k+1} \ldots u_m \rfloor = \mathsf{f}_m(\lfloor\bar{u}_m\rfloor)$. We extend this mapping to literals and clauses by applying it to each side of a literal and to each literal of a clause. Using $\lceil\ \rceil$, the clause order $\succ$ can be transferred to the floor logic by defining $t \succ s$ as equivalent to $\lceil t\rceil \succ \lceil s\rceil$. The property that $\succ$ on clauses is the multiset extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, is maintained because $\lceil\ \rceil$ maps the multiset representations elementwise.

Crucially, argument subterms in the ceiling logic correspond to argument subterms in the floor logic, whereas non-argument subterms in the ceiling logic are not subterms at all in the floor logic. Well-foundedness, totality on ground terms, compatibility with *all* contexts, and the subterm property hold for $\succ$ in the floor logic.

In standard superposition, redundancy relies on the entailment relation $\models$ on ground clauses. We define redundancy of ceiling clauses in the same way, but using the floor logic's entailment relation: A ground ceiling clause C is *redundant* with respect to a set of ceiling ground clauses N if $\lfloor C\rfloor$ is entailed by clauses from $\lfloor N\rfloor$ that are smaller than $\lfloor C\rfloor$. This notion of redundancy gracefully generalizes the first-order notion without making all ArgCong inferences redundant.

For Sup, EqFact, and EqRes, we can use the more precise notion of redundancy of inferences instead of redundancy of clauses, a ground inference being redundant if the conclusion follows from existing clauses that are smaller than the largest premise. For ArgCong and PosExt, we must use redundancy of clauses.

3.4 Skolemization

A problem expressed in λ-free higher-order logic must be transformed into clausal normal form before the calculi can be applied. This process works as in the first-order case, except for skolemization. The issue is that skolemization, when performed naively, is unsound for λ-free higher-order logic with a Henkin semantics. For example, given $\mathsf{f} : \tau \to \upsilon$, the formula $(\forall y.\, \exists x.\, \mathsf{f}\ x \approx y) \wedge (\forall z.\, \mathsf{f}\ (z\ \mathsf{a}) \not\approx \mathsf{a})$ has a model with $\mathcal{U}_\tau = \mathcal{U}_\upsilon$ that interprets f as the identity function and ensures that none of the functions in the image of $\mathcal{E}_{\upsilon,\tau}$ map $\mathcal{J}(\mathsf{a})$ to $\mathcal{J}(\mathsf{a})$.Yet, naive skolemization would yield the clause set $\{\mathsf{f}\ (\mathsf{sk}\ y) \approx y,\ \mathsf{f}\ (z\ \mathsf{a}) \not\approx \mathsf{a}\}$, whose unsatisfiability can be shown by taking $y := \mathsf{a}$ and $z := \mathsf{sk}$. The crux of the issue is that sk denotes a new function that can be used to instantiate z.

Inspired by Miller [28, Section 6], we adapt skolemization as follows. An existentially quantified variable $x : \tau$ in a context with universally quantified variables $\bar{x}_n$ of types $\bar{\tau}_n$ is replaced by a fresh symbol $\mathsf{sk} : \bar{\tau}_n \Rightarrow \tau$ applied to the tuple $\bar{x}_n$. For the example above, we obtain $\{\mathsf{f}\ (\mathsf{sk}(y)) \approx y,\ \mathsf{f}\ (z\ \mathsf{a}) \not\approx \mathsf{a}\}$. Syntactically, z cannot be instantiated by sk, which is not even a term. Semantically, the clause set is satisfiable because we can have $\mathcal{J}(\mathsf{sk})(\mathcal{J}(\mathsf{a})) = \mathcal{J}(\mathsf{a})$ even if the image of $\mathcal{E}_{\tau,\tau}$ contains no such function.

4 Refutational Completeness

The proof of refutational completeness of the four calculi introduced in Section 3.1 follows the same general idea as for standard superposition [2, 42]. Given a clause set $N \not\ni \bot$ saturated up to redundancy, we construct a term rewriting system R based on the set of ground instances $\mathcal{G}_\Sigma(N)$. From R, we define an interpretation. We show, by induction on the clause order, that this interpretation is a model of $\mathcal{G}_\Sigma(N)$ and hence of N.

To circumvent the term order's potential nonmonotonicity, our SUP inference rule only considers the argument subterms u of a maximal term $s\langle u\rangle$. This is reflected in our proof by the reliance of the floor logic from Section 3.3. In that logic, the equation $\mathrm{g}_0 \approx \mathrm{f}_0$ cannot be used directly to rewrite the clause $\mathrm{g}_1(\mathsf{a}_0) \not\approx \mathrm{f}_1(\mathsf{a}_0)$; instead, we first need to apply ARGCONG to derive $\mathrm{g}_1(x) \approx \mathrm{f}_1(x)$ and then use that equation. The floor logic is a device that enables us to reuse the traditional model construction almost verbatim, including its reliance on a first-order term rewriting system.

Following the traditional proof, we obtain a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Since N is saturated up to redundancy with respect to ARGCONG, the model $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ can easily be turned into a model of $\mathcal{G}_\Sigma(N)$ by conflating the interpretations of the members $\mathsf{f}_k, \ldots, \mathsf{f}_n$ of a same symbol family. For this section, we fix a set $N \not\ni \bot$ of λ-free higher-order clauses that is saturated up to redundancy. For the purifying calculi, we additionally require that all clauses in N are purified. To avoid empty Herbrand universes, we assume that the signature Σ contains, for each type τ, a symbol of type τ.

4.1 Candidate Interpretation

The construction of the candidate interpretation is as in the first-order proof, except that it is based on $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. We first define sets of rewrite rules E_C and R_C for all $C \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ by induction. Assume that E_D has already been defined for all $D \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ with $D \prec C$. Then $R_C = \bigcup_{D \prec C} E_D$. Let $E_C = \{s \to t\}$ if the following conditions are met: (a) $C = C' \vee s \approx t$; (b) $s \approx t$ is strictly maximal in C; (c) $s \succ t$; (d) C is false in R_C; (e) C' is false in $R_C \cup \{s \to t\}$; and (f) s is irreducible with respect to R_C. Otherwise, $E_C = \emptyset$. Finally, $R_\infty = \bigcup_D E_D$. A rewrite system R defines an interpretation $\mathcal{T}_\Sigma^\emptyset / R$ such that for every *ground* equation $s \approx t$, we have $\mathcal{T}_\Sigma^\emptyset / R \models s \approx t$ if and only if $s \leftrightarrow_R^* t$. Moreover, $\mathcal{T}_\Sigma^\emptyset / R$ is term-generated. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_\Sigma^\emptyset / R$.

4.2 Lifting Lemmas

Following Waldmann's version of the first-order proof [42], we proceed by lifting inferences from the ground to the nonground level. We also need to lift ARGCONG. A complication that arises when lifting purifying inferences is that the nonground conclusions may contain purification literals (corresponding to applied variables) not present in the ground conclusions. Given an inference I of the form $\bar{C} \vdash \mathit{pure}(E)$, we refer to the ground instances of $\bar{C} \vdash E$ as ground instances of *I up to purification*.

Lemma 2 (Lifting of non-SUP inferences). *Let $C\theta \in \mathcal{G}_\Sigma(N)$, where θ is a substitution and the selected literals in $C \in N$ correspond to those in $C\theta$. Then every* EQRES *or* EQFACT *inference from $C\theta$ and every ground instance of an* ARGCONG *inference from $C\theta$ is a ground instance of an inference from C up to purification.*

The conditions of the lifting lemma for SUP differ slightly from the first-order version. For standard superposition, the lemma applies if the superposed term is not at or under a variable. This condition is replaced by the following criterion.

Definition 3. We call a ground SUP inference from $D\theta$ and $C\theta$ *liftable* if the superposed subterm in $C\theta$ is not under a variable in C and the correponding variable condition holds for D and C.

Lemma 4 (Lifting of SUP inferences). *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$ where the selected literals in $D \in N$ and $C \in N$ correspond to those in $D\theta$ and $C\theta$, respectively. Then every liftable* SUP *inference between $D\theta$ and $C\theta$ is a ground instance of a* SUP *inference from D and C up to purification.*

4.3 Main Result

The candidate interpretation R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Like in the first-order proof, this is shown by induction on the clause order. For the induction step, we fix some clause $\lfloor C\theta \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ and assume that all smaller clauses are true in $R_{C\theta}$. We distinguish several cases, most of which amount to showing that $C\theta$ can be used in a certain inference. Then we deduce that $\lfloor C\theta \rfloor$ is true in $R_{C\theta}$ to complete the induction step.

The next two lemmas are slightly adapted from the first-order proof. The justification for Lemma 5, about liftable inferences, is essentially as in the first-order case. The proof of Lemma 6, about nonliftable inferences, is more problematic. The standard argument involves defining a substitution θ' such that $C\theta'$ and $C\theta$ are equivalent and $C\theta' \prec C\theta$. But due to nonmonotonicity, we might have $C\theta' \succ C\theta$, blocking the application of the induction hypothesis. This is where the variable conditions, purification, and the POSEXT rule come into play.

Lemma 5. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$, where the selected literals in $D \in N$ and in $C \in N$ correspond to those in $D\theta$ and $C\theta$, respectively. We consider a liftable* SUP *inference from $D\theta$ and $C\theta$ or an* EQRES *or* EQFACT *inference from $C\theta$. Let E be the conclusion. Assume that $C\theta$ and $D\theta$ are nonredundant with respect to $\mathcal{G}_\Sigma(N)$. Then $\lfloor E \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$.*

Lemma 6. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$, where the selected literals in $D \in N$ and in $C \in N$ correspond to those in $D\theta$ and $C\theta$, respectively. We consider a nonliftable* SUP *inference from $D\theta$ and $C\theta$. Assume that $C\theta$ and $D\theta$ are nonredundant with respect to $\mathcal{G}_\Sigma(N)$. Let $D'\theta$ be the clause $D\theta$ without the literal involved in the inference. Then $\lfloor C\theta \rfloor$ is entailed by $\neg \lfloor D'\theta \rfloor$ and the clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$.*

Using these two lemmas, the induction argument works as in the first-order case.

Lemma 7 (Model construction). *Let $\lfloor C\theta\rfloor \in \lfloor \mathcal{G}_\Sigma(N)\rfloor$. We have*

(i) $E_{\lfloor C\theta\rfloor} = \emptyset$ *if and only if* $R_{\lfloor C\theta\rfloor} \models \lfloor C\theta\rfloor$;
(ii) *if* $C\theta$ *is redundant with respect to* $\mathcal{G}_\Sigma(N)$, *then* $R_{\lfloor C\theta\rfloor} \models \lfloor C\theta\rfloor$;
(iii) $\lfloor C\theta\rfloor$ *is true in* R_∞ *and in* R_D *for every* $D \in \lfloor \mathcal{G}_\Sigma(N)\rfloor$ *with* $D \succ \lfloor C\theta\rfloor$; *and*
(iv) *if* $C\theta$ *has selected literals, then* $R_{\lfloor C\theta\rfloor} \models \lfloor C\theta\rfloor$.

Given a model R_∞ of $\lfloor \mathcal{G}_\Sigma(N)\rfloor$, we construct a model $R^{\uparrow}_{\infty}$ of $\mathcal{G}_\Sigma(N)$. The key properties are that R_∞ is term-generated and that the interpretations of the members $\mathsf{f}_k,\ldots,\mathsf{f}_n$ of a same symbol family behave in the same way.

Lemma 8 (Argument congruence). *For all ground terms* $\mathsf{f}_m(\bar{s})$ *and* $\mathsf{g}_n(\bar{t})$, *if* $[\![\mathsf{f}_m(\bar{s})]\!]^{\xi}_{R_\infty} = [\![\mathsf{g}_n(\bar{t})]\!]^{\xi}_{R_\infty}$, *then* $[\![\mathsf{f}_{m+1}(\bar{s},u)]\!]^{\xi}_{R_\infty} = [\![\mathsf{g}_{n+1}(\bar{t},u)]\!]^{\xi}_{R_\infty}$ *for all u.*

The proof relies on the saturation of N up to redundancy with respect to ArgCong.

Definition 9. Define an interpretation $R^{\uparrow}_{\infty} = (\mathcal{U}^{\uparrow}, \mathcal{E}^{\uparrow}, \mathcal{J}^{\uparrow})$ in the ceiling logic as follows. Let $(\mathcal{U}, \mathcal{E}, \mathcal{J}) = R_\infty$. Let $\mathcal{U}^{\uparrow}_{\tau} = \mathcal{U}_{\lfloor\tau\rfloor}$ and $\mathcal{J}^{\uparrow}(\mathsf{f}) = \mathcal{J}(\mathsf{f}_k)$, where k is the number of mandatory arguments of f. Since R_∞ is term-generated, for every $a \in \mathcal{U}_{\lfloor\tau\to\upsilon\rfloor}$, there exists a ground term $s : \tau \to \upsilon$ such that $[\![\lfloor s\rfloor]\!]^{\xi}_{R_\infty} = a$. Without loss of generality, we write $s = \mathsf{f}(\bar{s}_k)\ s_{k+1}\ \ldots\ s_m$. Then we have $a = [\![\mathsf{f}_m(\lfloor\bar{s}_m\rfloor)]\!]^{\xi}_{R_\infty}$ and define $\mathcal{E}^{\uparrow}$ by

$$\mathcal{E}^{\uparrow}_{\tau,\upsilon}(a)(b) = \mathcal{J}(\mathsf{f}_{m+1})([\![\lfloor\bar{s}_m\rfloor]\!]^{\xi}_{R_\infty}, b) \quad \text{for all } b \in \mathcal{U}_\tau$$

It follows that $\mathcal{E}^{\uparrow}_{\tau,\upsilon}(a)\big([\![u]\!]^{\xi}_{R_\infty}\big) = [\![\mathsf{f}_{m+1}(\lfloor\bar{s}_m\rfloor, u)]\!]^{\xi}_{R_\infty}$ for any term u. This interpretation is well defined if the definition of $\mathcal{E}^{\uparrow}$ does not depend on the choice of the ground term s. To show this, we assume that there exists another ground term $t = \mathsf{g}(\bar{t}_l)\ t_{l+1}\ \ldots\ t_n$ such that $[\![\lfloor t\rfloor]\!]^{\xi}_{R_\infty} = a$. By Lemma 8, it follows from $[\![\lfloor s\rfloor]\!]^{\xi}_{R_\infty} = [\![\lfloor t\rfloor]\!]^{\xi}_{R_\infty}$ that

$$[\![\mathsf{f}_{m+1}(\lfloor\bar{s}_m\rfloor, u)]\!]^{\xi}_{R_\infty} = [\![\mathsf{g}_{n+1}(\lfloor\bar{t}_n\rfloor, u)]\!]^{\xi}_{R_\infty}$$

indicating that the definition of $\mathcal{E}^{\uparrow}$ is independent of the choice of s.

Since R_∞ is a term-generated model of $\lfloor \mathcal{G}_\Sigma(N)\rfloor$, we can show that $R^{\uparrow}_{\infty}$ is also term-generated. And using the same argument as in the first-order proof, we can lift this result to nonground clauses. For the extensional variants, we also need to show that $R^{\uparrow}_{\infty}$ is an extensional interpretation.

Lemma 10 (Model transfer to ceiling logic). $R^{\uparrow}_{\infty}$ *is a term-generated model of* $\mathcal{G}_\Sigma(N)$.

Lemma 11 (Model transfer to nonground clauses). $R^{\uparrow}_{\infty}$ *is a model of N.*

Lemma 12 (Completeness of the extensionality axioms). *If N contains the extensionality axioms,* $R^{\uparrow}_{\infty}$ *is extensional.*

We summarize the results of this section in the following theorem.

Theorem 13 (Refutational completeness). *Let N be a clause set that is saturated by any of the four calculi, up to redundancy. For the purifying calculi, we additionally assume that all clauses in N are purified. Then N has a model if and only if* $\bot \notin N$. *Such a model is extensional if N contains the extensionality axioms.*

5 Implementation

Zipperposition [16, 17] is an open source superposition-based theorem prover written in OCaml.[1] It was initially designed for polymorphic first-order logic with equality, as embodied by TPTP TFF [10]. We will refer to this implementation as Zipperposition's first-order mode. Recently, we extended the prover with a pragmatic higher-order mode with support for λ-abstractions and extensionality, without any completeness guarantees. Using this mode, Zipperposition entered the 2017 edition of the CADE ATP System Competition [38]. We have now also implemented a complete λ-free higher-order mode based on the four calculi described in this paper, extended with polymorphism.

The pragmatic higher-order mode provided a convenient basis to implement our calculi. It includes higher-order term and type representations and orders. Its ad hoc calculus extensions are similar to our calculi. Notably, they include an ArgCong rule and a PosExt-like rule, and Sup inferences are performed only at argument subterms. In the term indexes, which are imperfect (overapproximating), terms whose heads are applied variables and λ-abstractions are treated as fresh variables. This could be further optimized to reduce the number of unification candidates.

To implement the λ-free mode, we restricted the unification algorithm to non-λ-terms, and we added support for mandatory arguments to make skolemization sound, by associating the number of mandatory arguments to each symbol and incorporating this number in the unification algorithm. To satisfy the requirements on selection, we avoid selecting literals that contain higher-order variables. Finally, we disabled rewriting of non-argument subterms to comply with our redundancy notion.

For the purifying calculi, we implemented purification as a simplification rule. This ensures that it is applied aggressively on all clauses, whether initial clauses from the problem or clauses produced during saturation, before any inferences are performed.

For the nonpurifying calculi, we added the possibility to perform Sup inferences at variable positions. This means that variables must be indexed as well. In addition, we modified the variable condition. However, it is in general impossible to decide whether there exists a ground substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$. We overapproximate the condition as follows: (1) check whether x appears with different arguments in the clause C; (2) use an order-specific algorithm (for LPO and KBO) to determine whether there might exist a ground substitution θ and terms $\bar{u}$ such that $t\sigma\theta \succ t'\sigma\theta$ and $t\sigma\theta\ \bar{u} \prec t'\sigma\theta\ \bar{u}$; and (3) check whether $C\sigma \not\succeq C''\sigma$. If these three conditions apply, we conclude that there might exist a ground substitution θ witnessing nonmonotonicity.

For the extensional calculi, we added a single extensionality axiom based on a polymorphic symbol $\mathsf{diff} : \forall\alpha\beta.\ (\alpha \to \beta)^2 \Rightarrow \alpha$. To curb the explosion associated with extensionality, this axiom and all clauses derived from it are penalized by the clause selection heuristic. We also added a negative extensionality rule that resembles Vampire's [21].

Using Zipperposition, we can quantify the disadvantage of the applicative encoding on the problem given at the end of Section 3.2. Well-chosen LPO and KBO instances allow Zipperposition to derive $\bot$ in 4 iterations of the prover's main loop and 0.04 s. KBO or LPO with default settings needs 203 iterations and 0.5 s, whereas KBO or LPO on the applicatively encoded problem needs 203 iterations and almost 2 s.

[1] `https://github.com/c-cube/zipperposition`

6 Evaluation

We evaluated Zipperposition's implementation of our four calculi on TPTP benchmarks. We compare them with Zipperposition's first-order mode on the applicative encoding with and without the extensionality axiom. Our experimental data is available online.[2] Since the present work is only a stepping stone towards a prover for full higher-order logic, it is too early to compare this prototype to state-of-the-art higher-order provers that support a stronger logic.

We instantiated all variants with LPO [11] (which is nonmonotonic) and KBO [3] without argument coefficients (which is monotonic). This gives us a rough indication of the cost of nonmonotonicity. However, when using a monotonic order, it may be more efficient (and also refutationally complete) to superpose at non-argument subterms directly instead of relying on the ArgCong rule.

We collected 671 first-order problems in TFF format and 1114 higher-order problems in THF, both groups containing monomorphic and polymorphic problems. We excluded all problems containing λ-expressions, the quantifier constants !! ($\forall$) and ?? ($\exists$), arithmetic types, or the $distinct predicate, as well as problems that mix Booleans and terms. Figures 1 and 2 summarize, for various configurations, the number of solved satisfiable and unsatisfiable problems within 300 s. The average time and number of main loop iterations are computed over the problems that all configurations for the respective logic and term order found to be unsatisfiable within the timeout. The evaluation was carried out on StarExec [37] using Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz.

Our approach targets large, mildly higher-order problems—a practically relevant class of problems that is underrepresented in the TPTP library. The experimental results confirm that our calculi handle first-order problems gracefully. Even the extensional calculi, which include (graceless) extensionality axioms, are almost as effective as the first-order mode. This indicates that our calculi will perform well on mildly higher-order problems, too, where the proving effort is dominated by first-order reasoning. In contrast, the applicative encoding is comparatively inefficient on problems that are already first-order. For LPO, the success rate drops by 16%–18%; for both orders, the average time to show unsatisfiability roughly quadruples.

Many of the higher-order problems in the TPTP library are satisfiable for our λ-free logic, even though they may be unsatisfiable for full higher-order logic and labeled as such in the TPTP. This is a reason why we postpone a comparison with state-of-the-art higher-order provers until we have developed a prover for full higher-order logic. On higher-order problems, the nonpurifying calculi outperform their purifying relatives. The comparison of the applicative enconding and the nonpurifying calculi, however, is not entirely conclusive. In the light of the results of this evaluation, in future work, we would like to collect benchmarks for large, mildly higher-order problems and to investigate whether we can weaken the selection restrictions of our calculi.

The nonpurifying calculi perform slightly better with KBO than with LPO. This confirms our expectations, given that KBO is generally considered the more robust default option for superposition and that the nonmonotonic LPO triggers Sup inferences at variable positions—which is the price to pay for nonmonotonicity.

[2] http://matryoshka.gforge.inria.fr/pubs/lfhosup_data/

		# sat		# unsat		∅ time (s)		∅ iterations	
		LPO	KBO	LPO	KBO	LPO	KBO	LPO	KBO
TFF	first-order mode	0	0	**181**	**220**	**4.0**	**4.4**	**1497**	**1473**
	applicative encoding	0	0	150	203	19.0	16.0	1698	1916
	nonpurifying calculus	0	0	**181**	219	4.2	4.6	**1497**	**1473**
	purifying calculus	0	0	**181**	218	4.3	4.8	**1497**	**1473**
THF	applicative encoding	**444**	**438**	**676**	671	0.8	**0.2**	**72**	81
	nonpurifying calculus	353	360	675	**676**	**0.6**	0.3	83	**63**
	purifying calculus	338	343	664	666	0.8	1.0	116	231

Fig. 1: Evaluation of the intensional calculi

		# sat		# unsat		∅ time (s)		∅ iterations	
		LPO	KBO	LPO	KBO	LPO	KBO	LPO	KBO
TFF	first-order mode	0	0	**181**	**220**	**2.8**	**4.3**	**1219**	**1420**
	applicative encoding	0	0	151	201	19.0	17.6	1837	1792
	nonpurifying calculus	0	0	179	215	6.2	6.8	1610	1524
	purifying calculus	0	0	180	215	5.0	7.4	1291	1464
THF	applicative encoding	**426**	**421**	**677**	671	0.7	0.8	**78**	89
	nonpurifying calculus	310	327	669	**675**	**0.6**	**0.4**	83	**66**
	purifying calculus	227	261	647	650	1.0	1.0	114	108

Fig. 2: Evaluation of the extensional calculi

7 Discussion and Related Work

Our calculi join a long list of extensions and refinements of superposition. Among the most closely related is Peltier's [30] Isabelle formalization of the refutational completeness of a superposition calculus that operates on λ-free higher-order terms and that is parameterized by a monotonic term order. Extensions with polymorphism and induction, developed by Cruanes [16, 17] and Wand [43], contribute to increasing the power of automatic provers. Detection of inconsistencies in axioms, as suggested by Schulz et al. [34], is important for large axiomatizations. Also of interest is Bofill and Rubio's [13] integration of nonmonotonic orders in ordered paramodulation, a precursor of superposition. Their work is a veritable tour de force, but it is also highly complicated and restricted to ordered paramodulation. Lack of compatibility with arguments being a mild form of nonmonotonicity, it seemed preferable to start with superposition, enrich it with an ArgCong rule, and tune the side conditions until we obtained a complete calculus.

Most complications can be avoided by using a monotonic order such as KBO without argument coefficients, but we suspect that the coefficients will play an important role to support λ-abstractions. For example, the term $\lambda x.\ x+x$ could be treated as a constant with a coefficient of 2 on its argument and a heavy weight to ensure $(\lambda x.\ x+x)\ y \succ y+y$. LPO can also be used to good effect. This technique could allow provers to perform aggressive β-reduction in the vast majority of cases, without compromising completeness.

Many researchers have proposed or used encodings of higher-order logic constructs into first-order logic, including Robinson [32], Kerber [24], Dowek et al. [19], Meng and Paulson [27], and Czajka [18]. Encodings of types, such as those by Bobot and Paskevich [12] and Blanchette et al. [8], are also crucial to obtain a sound encoding of higher-order logic. These ideas are implemented in proof assistant tools such as HOLy-Hammer and Sledgehammer [9].

Another line of research has focused on the development of automated proof procedures for higher-order logic. Robinson's [31] and Huet's [23] pioneering work stands out. Andrews [1] and Benzmüller and Miller [6] provide excellent surveys. The competitive higher-order automatic theorem provers include LEO-II [7] (based on unordered paramodulation), Satallax [15] (based on a tableau calculus and a SAT solver), Agsy-HOL [26] (based on a focused sequent calculus and a generic narrowing engine), and Leo-III [36] (based on a pragmatic extension of superposition with no completeness guarantees). The Isabelle proof assistant [29] and its Sledgehammer subsystem also participate in the higher-order division of the CADE ATP System Competition [38].

Zipperposition is a convenient vehicle for experimenting and prototyping because it is easier to understand and modify than highly-optimized C or C++ provers. Our middle-term goal is to design higher-order superposition calculi, implement them in state-of-the-art provers such as E [33], SPASS [44], and Vampire [25], and integrate these in proof assistants to provide a high level of automation. With its stratified architecture, Otter-λ [4] is perhaps the closest to what we are aiming at, but it is limited to second-order logic and offers no completeness guarantees. In preliminary work supervised by Blanchette and Schulz, Vukmirović [41] has generalized E's data structures and algorithms to λ-free higher-order logic, assuming a monotonic KBO [3].

8 Conclusion

We presented four superposition calculi for intensional and extensional λ-free higher-order logic and proved them refutationally complete. The calculi nicely generalize standard superposition and are compatible with our λ-free higher-order LPO and KBO. Our experiments partly confirm what one would naturally expect: that native support for partial application and applied variables outperforms the applicative encoding.

The new calculi reduce the gap between proof assistants based on higher-order logic and superposition provers. We can use them to reason about arbitrary higher-order problems by axiomatizing suitable combinators. But perhaps more importantly, they appear promising as a stepping stone towards complete, highly efficient automatic theorem provers for full higher-order logic.

Acknowledgment. We are grateful to the maintainers of StarExec for letting us use their service. We want to thank Christoph Benzmüller, Sander Dahmen, Johannes Hölzl, Anders Schlichtkrull, Stephan Schulz, Alexander Steen, Geoff Sutcliffe, Andrei Voronkov, Petar Vukmirović, Daniel Wand, Christoph Weidenbach, and the participants in the 2017 Dagstuhl Seminar on Deduction beyond First-Order Logic for stimulating discussions. We also want to thank Mark Summerfield, Sophie Tourret, and the anonymous reviewers for suggesting several textual improvements to this paper and to the technical report. Bentkamp and Blanchette's research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

References

[1] Andrews, P.B.: Classical type theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, pp. 965–1007. Elsevier and MIT Press (2001)

[2] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)

[3] Becker, H., Blanchette, J.C., Waldmann, U., Wand, D.: A transfinite Knuth–Bendix order for lambda-free higher-order terms. In: de Moura, L. (ed.) CADE-26. LNCS, vol. 10395, pp. 432–453. Springer (2017)

[4] Beeson, M.: Lambda logic. In: Basin, D.A., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 460–474. Springer (2004)

[5] Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic (technical report). Technical report (2018), `http://matryoshka.gforge.inria.fr/pubs/lfhosup_report.pdf`

[6] Benzmüller, C., Miller, D.: Automation of higher-order logic. In: Siekmann, J.H. (ed.) Computational Logic, Handbook of the History of Logic, vol. 9, pp. 215–254. Elsevier (2014)

[7] Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II—A cooperative automatic theorem prover for higher-order logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 162–170. Springer (2008)

[8] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. 12(4:13), 1–52 (2016)

[9] Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reas. 9(1), 101–148 (2016)

[10] Blanchette, J.C., Paskevich, A.: TFF1: The TPTP typed first-order form with rank-1 polymorphism. In: Bonacina, M.P. (ed.) CADE-24. LNCS, vol. 7898, pp. 414–420. Springer (2013)

[11] Blanchette, J.C., Waldmann, U., Wand, D.: A lambda-free higher-order recursive path order. In: Esparza, J., Murawski, A.S. (eds.) FoSSaCS 2017. LNCS, vol. 10203, pp. 461–479. Springer (2017)

[12] Bobot, F., Paskevich, A.: Expressing polymorphic types in a many-sorted language. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS, vol. 6989, pp. 87–102. Springer (2011)

[13] Bofill, M., Rubio, A.: Paramodulation with non-monotonic orderings and simplification. J. Autom. Reason. 50(1), 51–98 (2013)

[14] Brand, D.: Proving theorems with the modification method. SIAM J. Comput. 4, 412–430 (1975)

[15] Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 111–117. Springer (2012)

[16] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)

[17] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)

[18] Czajka, Ł.: Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions. In: Avigad, J., Chlipala, A. (eds.) CPP 2016. pp. 49–57. ACM (2016)

[19] Dowek, G., Hardin, T., Kirchner, C.: Higher-order unification via explicit substitutions (extended abstract). In: LICS '95. pp. 366–374. IEEE (1995)

[20] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)

[21] Gupta, A., Kovács, L., Kragl, B., Voronkov, A.: Extensional crisis and proving identity. In: Cassez, F., Raskin, J. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 185–200. Springer (2014)

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

[23] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)

[24] Kerber, M.: How to prove higher order theorems in first order logic. In: Mylopoulos, J., Reiter, R. (eds.) IJCAI-91. pp. 137–142. Morgan Kaufmann (1991)

[25] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer (2013)

[26] Lindblad, F.: A focused sequent calculus for higher-order logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 61–75. Springer (2014)

[27] Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. 40(1), 35–60 (2008)

[28] Miller, D.A.: A compact representation of proofs. Studia Logica 46(4), 347–370 (1987)

[29] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)

[30] Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs (2016), `https://www.isa-afp.org/entries/SuperCalc.shtml`

[31] Robinson, J.: Mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)

[32] Robinson, J.: A note on mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 121–135. Edinburgh University Press (1970)

[33] Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) LPAR-19. LNCS, vol. 8312, pp. 735–743. Springer (2013)

[34] Schulz, S., Sutcliffe, G., Urban, J., Pease, A.: Detecting inconsistencies in large first-order knowledge bases. In: de Moura, L. (ed.) CADE-26. LNCS, vol. 10395, pp. 310–325. Springer (2017)

[35] Snyder, W., Lynch, C.: Goal directed strategies for paramodulation. In: Book, R.V. (ed.) RTA-91. LNCS, vol. 488, pp. 150–161. Springer (1991)

[36] Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, Springer (2018)

[37] Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 367–373. Springer (2014)

[38] Sutcliffe, G.: The CADE-26 automated theorem proving system competition—CASC-26. AI Commun. 30(6), 419–432 (2017)

[39] Sutcliffe, G., Benzmüller, C., Brown, C.E., Theiss, F.: Progress in the development of automated theorem proving for higher-order logic. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 116–130. Springer (2009)

[40] Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP typed first-order form with arithmetic. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18. LNCS, vol. 7180, pp. 406–419. Springer (2012)

[41] Vukmirović, P.: Implementation of Lambda-Free Higher-Order Superposition. M.Sc. thesis, Vrije Universiteit Amsterdam (2018)

[42] Waldmann, U.: Automated reasoning II. Lecture notes, Max-Planck-Institut für Informatik (2016), `http://resources.mpi-inf.mpg.de/departments/rg1/teaching/autrea2-ss16/script-current.pdf`

[43] Wand, D.: Superposition: Types and Polymorphism. Ph.D. thesis, Universität des Saarlandes (2017)

[44] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 140–145. Springer (2009)