Superposition for Lambda-Free Higher-Order Logic (Technical Report)

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 [3] 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 report 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 $\mathsf{pow}_\tau : \mathit{nat} \to (\tau \to \tau) \to \tau \to \tau$ (intended to denote the iterated application $h^n\ x$):

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

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 [34, 47].

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 [34] and is employed fruitfully in tools such as Sledgehammer [11, 39], 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 [35, Section 2.3.1]. 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 [10, Section 3.3] in conjunction with the applicative encoding, the term $\mathsf{S}\;x$ becomes $\mathsf{t}(\mathsf{nat}, \mathsf{app}(\mathsf{t}(\mathsf{fun}(\mathsf{nat}, \mathsf{nat}), \mathsf{S}), \mathsf{t}(\mathsf{nat}, x)))$. The term's simple structure is lost in translation.

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 [39]. 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 [7]. 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) [13] and the Knuth–Bendix order (KBO) [5] 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 superposition 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.*

The calculi are implemented in the Zipperposition prover [21] (Section 5). We evaluate them on first- and higher-order TPTP benchmarks [55, 56] and compare them with the applicative encoding (Section 6). We find that there is a substantial cost associated with the applicative encoding, that the nonmonotonicity is not particularly expensive, and that the nonpurifying calculi outperform the purifying variants.

2 Logic

Refutational completeness of calculi for higher-order logic (also called simple type theory) [19, 28] is usually stated with respect to Henkin semantics [7, 30], 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. In that sense, our semantics resemble Henkin prestructures [37, Section 5.4]. Unlike other higher-order logics [26], 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 [40]. We achieve this using a *hybrid logic* that supports both mandatory (uncurried) and optional (curried) arguments, inspired by higher-order term rewriting [35]. Thus, if symbol sk takes two mandatory and one optional arguments, $\mathsf{sk}(x,y)$ and $\mathsf{sk}(x,y)\ z$ are valid terms, whereas sk and $\mathsf{sk}(x)$ are invalid. Nevertheless, as in our earlier work [5,13], 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 $n = 0$). Here and elsewhere, 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, w 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 as follows. 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*. Using the spine notation [18], 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 of the same type 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 finite multisets of such (dis)equations.

Loosely following Fitting [27], an *interpretation* $\mathcal{J} = (\mathcal{U}, \mathcal{E}, \mathcal{J})$ 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 for interactive theorem proving; for example, Coq's calculus of inductive constructions is intensional [9], whereas Isabelle/HOL is extensional [43]. 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: $[\![x]\!]^\xi_\mathcal{J} = \xi(x)$; $[\![\mathsf{f}(\bar{t})]\!]^\xi_\mathcal{J} = \mathcal{J}(\mathsf{f})([\![\bar{t}]\!]^\xi_\mathcal{J})$; $[\![s\ t]\!]^\xi_\mathcal{J} = \mathcal{E}([\![s]\!]^\xi_\mathcal{J})([\![t]\!]^\xi_\mathcal{J})$. The truth value $[\![\varphi]\!]^\xi_\mathcal{J} \in \{0,1\}$ of a formula φ is defined as in first-order logic:

$$[\![\forall(x : \tau)\ \psi]\!]^\xi_\mathcal{J} = \min_{a \in \mathcal{U}_\tau} \{[\![\psi]\!]^{\xi[x\mapsto a]}_\mathcal{J}\} \qquad [\![\exists(x : \tau)\ \psi]\!]^\xi_\mathcal{J} = \max_{a \in \mathcal{U}_\tau} \{[\![\psi]\!]^{\xi[x\mapsto a]}_\mathcal{J}\}$$

A formula φ is true in $\mathcal{J} = (\mathcal{U}, \mathcal{E}, \mathcal{J})$ under valuation ξ and we write $\mathcal{J}, \xi \models \varphi$ if $[\![\varphi]\!]^\xi_\mathcal{J} = 1$. The interpretation $\mathcal{J}$ is a model of φ, written $\mathcal{J} \models \varphi$, if $\mathcal{J}, \xi \models \varphi$ for all valuations ξ into $\{\mathcal{U}_\tau\}_\tau$.

For example, given the signature $\Sigma = \{\mathsf{a} : \iota\}$, the formula $\forall(h : \iota \to \iota).\ h\ \mathsf{a} \not\approx \mathsf{a}$ has an extensional model given by $\mathcal{U}_\iota = \{a, b\}$, $\mathcal{U}_{\iota\to\iota} = \{f\}$, $\mathcal{E}_{\iota,\iota}(f)(a) = \mathcal{E}_{\iota,\iota}(f)(b) = b$, and $\mathcal{J}(\mathsf{a}) = a$, where $a \neq b$.

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 "has argument" 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 or, equivalently, that $s[\,]$ is a function context. 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 from the term order 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 the five rules and the extensionality axiom given on page 5. We view positive and negative superposition as two cases of one rule called SUP. We have two rules and one axiom in addition to the standard first-order rules and their usual order conditions.

Positive superposition:

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

- $\sigma = \text{mgu}(t,u)$
- $t\sigma \not\preceq t'\sigma$ and $s\langle u\rangle\sigma \not\preceq s'\sigma$
- $(t \approx t')\sigma$ is strictly eligible in $D\sigma$
- $(s\langle u\rangle \approx s')\sigma$ is strictly eligible in $C\sigma$
- $C\sigma \not\preceq D\sigma$
- the variable conditions holds

Negative superposition:

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

- $\sigma = \text{mgu}(t,u)$
- $t\sigma \not\preceq t'\sigma$ and $s\langle u\rangle\sigma \not\preceq s'\sigma$
- $(t \approx t')\sigma$ is strictly eligible in $D\sigma$
- $(s\langle u\rangle \not\approx s')\sigma$ is eligible in $C\sigma$
- $C\sigma \not\preceq D\sigma$
- the variable condition holds

Equality resolution:

$$\frac{C' \vee s \not\approx s'}{C'\sigma}\,\textsc{EqRes}$$

- $\sigma = \text{mgu}(s,s')$
- $(s \not\approx s')\sigma$ is eligible in the premise

Equality factoring:

$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\,\textsc{EqFact}$$

- $\sigma = \text{mgu}(s,s')$
- $s'\sigma \not\preceq t'\sigma$ and $s\sigma \not\preceq t\sigma$
- $(s \approx t)\sigma$ is eligible in the premise

Argument congruence:

$$\frac{C' \vee s \approx s'}{C' \vee s\,\bar{x} \approx s'\,\bar{x}}\,\textsc{ArgCong}$$

- $\bar{x}$ contains fresh variables
- $s \approx s'$ is strictly eligible in the premise

Positive extensionality:

$$\frac{C' \vee s\,\bar{x} \approx s'\,\bar{x}}{C' \vee s \approx s'}\,\textsc{PosExt}$$

- $\bar{x}$ is a tuple of variables that occur nowhere else in the premise
- $s\,\bar{x} \approx s'\,\bar{x}$ is strictly eligible in the premise

Extensionality axiom: For every function type $\tau \to \upsilon$, we introduce a Skolem symbol $\mathsf{diff}_{\tau,\upsilon} : (\tau \to \upsilon)^2 \Rightarrow \tau$ characterized by the axiom

$$x\,(\mathsf{diff}(x,y)) \not\approx y\,(\mathsf{diff}(x,y)) \vee x \approx y$$

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 terms $\tilde{t}$, $\tilde{t}'$ and distinct variables $\bar{y}_n$ that do not occur elsewhere in D.

We add the following *variable condition* as a side condition to the SUP rules, 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 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 monotonic 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 [16, 23, 48, 51] (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 ensure that all clauses are purified, by processing the initial clause set and the conclusion of every inference or simplification.

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 $x_i \not\approx x$. For example, $\mathit{pure}(x\;\mathsf{a} \approx x\;\mathsf{b} \vee \mathsf{f}\;x \approx \mathsf{g}\;x) = (x\;\mathsf{a} \approx x_1\;\mathsf{b} \vee \mathsf{f}\;x_2 \approx \mathsf{g}\;x_2 \vee x_1 \not\approx x \vee x_2 \not\approx x)$. 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 $x_i \not\approx x$. For example, $\mathit{pure}(x\;\mathsf{a} \approx x\;\mathsf{b} \vee \mathsf{f}\;x \approx \mathsf{g}\;x) = (x\;\mathsf{a} \approx x\;\mathsf{b} \vee \mathsf{f}\;x_1 \approx \mathsf{g}\;x_1 \vee x \not\approx x_1)$. 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. These restrictions are needed for our completeness proof, but it might be possible to avoid them at the cost of a more elaborate argument.

Remark 2. In descriptions of first-order logic with equality, the property $y \approx y' \longrightarrow \mathsf{f}(\bar{x}, y, \bar{z}) \approx \mathsf{f}(\bar{x}, y', \bar{z})$ is often referred to as "function congruence." It seems natural to use the same label for the higher-order version $t \approx t' \longrightarrow s\ t \approx s\ t'$ and to call the symmetric property $s \approx s' \longrightarrow s\ t \approx s'\ t$ "argument congruence," whence the name ArgCong for our inference rule. Confusingly, the corresponding Isabelle/HOL theorems are called *arg_cong* and *fun_cong*, respectively.

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 rules, 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 because the variable conditions are weaker. The following example shows that the intensional nonpurifying calculus would be incomplete if we used the variable condition of the extensional nonpurifying calculus. Consider a left-to-right LPO [13] instance with precedence $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, and consider the following unsatisfiable clause set:

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

The only possible inference is a Sup inference of the first into the second clause, but the variable condition of the extensional nonpurifying calculus is not met. It is unclear whether the variable condition of the intensional purifying calculus cannot be strengthened either, but our completeness proof suggests that it cannot.

The variable condition in the extensional calculi is designed to prevent the argument congruence explosion 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. This is why we need the PosExt rule. In the following unsatisfiable clause set, the only possible inference from these clauses in the extensional nonpurifying calculus is PosExt, showing its necessity.

$$\mathsf{g}\,x \approx \mathsf{f}\,x \qquad \mathsf{g} \not\approx \mathsf{f} \qquad x\,(\mathsf{diff}(x,y)) \not\approx y\,(\mathsf{diff}(x,y)) \lor x \approx y$$

The same argument applies for the purifying variant with the difference that the last clause in this example must be purified.

Due to nonmonotonicity, for refutational completeness we need either to purify the clauses or to allow some superposition at variable positions, as mandated by the respective variable conditions. Without either of these measures, at least the extensional calculi and presumably also the intensional calculi would be incomplete, as the next example demonstrates. Consider the following clause set:

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

Using a left-to-right LPO [13] instance with precedence $\mathsf{k} \succ \mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, this clause set is saturated by the extensional purification variant when omitting purification. It is also saturated by the extensional nonpurifying variant when omitting Sup inferences at variables. By contrast, the intensional variants derive $\bot$, even without purification and without Sup inferences at variables, because of the less restrictive variable conditions. This raises the question as to whether the intensional variants actually need to purify or to perform Sup inferences at variables. Omitting purification and Sup inferences at variables in the intensional calculi is complete when redundant clauses are not deleted, but we conjecture that it is incomplete in general.

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:

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

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.

We initially considered inference rules instead of the extensionality axiom. However, we did not find a set of inference that is complete and leads to fewer inferences than the extensionality axiom. We considered the two inference rules

$$\frac{C \lor s\,x \approx t\,x}{C \lor s \approx t}\,\textsc{PosExt} \qquad \frac{C \lor s \not\approx t}{C \lor s\,\mathsf{sk} \not\approx t\,\mathsf{sk}}\,\textsc{NegExt}$$

where x is a fresh variable and sk is a fresh Skolem term. However, these two rules do not suffice for a refutationally complete calculus, as the following example demonstrates:

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

Assuming that all four equations are oriented from left to right, this set is saturated with respect to the superposition inference rules extended with PosExt and NegExt, yet it is unsatisfiable in an extensional logic.

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 $m \in \{k, \ldots, n\}$. Here and elsewhere, we write $\lfloor \bar{a} \rfloor$ for the componentwise application of $\lfloor\ \rfloor$ to the tuple $\bar{a}$. The translation of ground terms is given by $\lfloor \mathsf{f}(\bar{u}_k)\, u_{k+1} \ldots u_m \rfloor = \mathsf{f}_m(\lfloor \bar{u}_m \rfloor)$.

For example, let $\Sigma = \{\mathsf{f} : \iota^2 \Rightarrow \iota \to \iota, \mathsf{a} : \iota\}$. The corresponding floor logic signature is $\Sigma^{\downarrow} = \{\mathsf{f}_2 : \lfloor \iota \rfloor^2 \Rightarrow \lfloor \iota \to \iota \rfloor, \mathsf{f}_3 : \lfloor \iota \rfloor^3 \Rightarrow \lfloor \iota \rfloor, \mathsf{a}_0 : \lfloor \iota \rfloor\}$ where $\lfloor \iota \rfloor$ and $\lfloor \iota \to \iota \rfloor$ are atomic types. The term $\lfloor \mathsf{f}(\mathsf{a},\mathsf{a})\ \mathsf{a} \rfloor = \mathsf{f}_3(\mathsf{a}_0,\mathsf{a}_0,\mathsf{a}_0)$ is of type $\lfloor \iota \rfloor$, and $\lfloor \mathsf{f}(\mathsf{a},\mathsf{a}) \rfloor = \mathsf{f}_2(\mathsf{a}_0,\mathsf{a}_0)$ is of type $\lfloor \iota \to \iota \rfloor$.

The $\lfloor\ \rfloor$ mapping can be extended to ground literals and ground clauses:

$$\begin{aligned} \lfloor s \approx t \rfloor &= \lfloor s \rfloor \approx \lfloor t \rfloor \\ \lfloor s \not\approx t \rfloor &= \lfloor s \rfloor \not\approx \lfloor t \rfloor \\ \lfloor L_1 \vee \cdots \vee L_n \rfloor &= \lfloor L_1 \rfloor \vee \cdots \vee \lfloor L_n \rfloor \end{aligned}$$

The $\lfloor\ \rfloor$ mapping is bijective with $\lceil\ \rceil$:

$$\begin{aligned} \lceil \mathsf{f}_{k+i}(t_1, \ldots, t_{k+i}) \rceil &= \mathsf{f}(\lceil t_1 \rceil, \ldots, \lceil t_k \rceil)\ \lceil t_{k+1} \rceil \ldots \lceil t_{k+i} \rceil \\ \lceil s \approx t \rceil &= \lceil s \rceil \approx \lceil t \rceil \\ \lceil s \not\approx t \rceil &= \lceil s \rceil \not\approx \lceil t \rceil \\ \lceil L_1 \vee \cdots \vee L_n \rceil &= \lceil L_1 \rceil \vee \cdots \vee \lceil L_n \rceil \end{aligned}$$

(In the first equation, k is determined by the type declaration of f.) 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 (Lemma 3), whereas non-argument subterms in the ceiling logic are not subterms at all in the floor logic. In the floor logic, the notions of subterms and argument subterms coincide because the signature does not contain optional arguments.

Lemma 3. *For all terms s and t in the floor logic,* $\lceil t[s]_p \rceil = \lceil t \rceil \langle \lceil s \rceil \rangle_p$.

Proof. By induction on p.

If $p = \varepsilon$, then $s = t[s]_p$. Hence $\lceil t[s]_p \rceil = \lceil s \rceil = \lceil t \rceil \langle \lceil s \rceil \rangle_p$.

If $p = i.p'$, then $t[s]_p = \mathsf{f}_n(u_1, \ldots, u_n)$ with $u_i = u_i[s]_{p'}$. Applying $\lceil\ \rceil$, we obtain by the induction hypothesis that $\lceil t[s]_p \rceil$ equals

$$\mathsf{f}(\lceil u_1 \rceil, \ldots, \lceil u_{i-1} \rceil, \lceil u_i \rceil \langle \lceil s \rceil \rangle_{p'}, \lceil u_{i+1} \rceil, \ldots, \lceil u_k \rceil)\ \lceil u_{k+1} \rceil \ldots \lceil u_n \rceil$$

or

$$\mathsf{f}(\lceil u_1 \rceil, \ldots, \lceil u_k \rceil)\ \lceil u_{k+1} \rceil \ldots \lceil u_{i-1} \rceil\ \lceil u_i \rceil \langle \lceil s \rceil \rangle_{p'}\ \lceil u_{i+1} \rceil \ldots \lceil u_n \rceil$$

depending on whether the ith argument of f is mandatory or optional. In both cases, it follows that $\lceil t[s]_p \rceil = \lceil t \rceil \langle \lceil s \rceil \rangle_p$. $\square$

Lemma 4. *Well-foundedness, totality on ground terms, compatibility with* all *contexts, and the subterm property hold for* $\succ$ *in the floor logic.*

Proof. COMPATIBILITY WITH CONTEXTS: We want to show that $s \succ s'$ implies $t[s]_p \succ t[s']_p$ for floor terms t, s and s'. Assuming $s \succ s'$, we have $\lceil s \rceil \succ \lceil s' \rceil$. By compatibility with function contexts in the ceiling logic, we have $\lceil t \rceil \langle \lceil s \rceil \rangle_p \succ \lceil t \rceil \langle \lceil s' \rceil \rangle_p$. By Lemma 3, we have $t[s]_p \succ t[s']_p$.

WELL-FOUNDEDNESS: Assume that there exists an infinite descending chain $t_1 \succ t_2 \succ \cdots$ of floor terms. By applying $\lceil\ \rceil$, we then obtain an infinite descending chain of ceiling terms $\lceil t_1 \rceil \succ \lceil t_2 \rceil \succ \cdots$, contradicting well-foundedness in the ceiling logic.

TOTALITY ON GROUND TERMS: Let s, t be ground terms of the floor logic. Then $\lceil t \rceil$ and $\lceil s \rceil$ must be comparable by totality on ground ceiling terms. Hence, t and s are comparable.

SUBTERM PROPERTY: By Lemma 3 and the subterm property in the ceiling logic, $\lceil t[s]_p \rceil = \lceil t \rceil \langle \lceil s \rceil \rangle_p \succ \lceil s \rceil$. Hence, $t[s]_p \succ s$. $\square$

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. 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.

More precisely, we define redundancy as follows: 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 $\lfloor C \rfloor$. A possibly nonground ceiling clause C is *redundant with respect to a set of ceiling clauses* N if all its ground instances are redundant with respect to $\mathcal{G}_\Sigma(N)$, the set of ground instances of clauses in N. Let $\mathrm{Red}(N)$ be the set of all clauses that are redundant with respect to N.

For all inference rules except ArgCong and PosExt, a ground inference with conclusion E and right (or only) premise C is *redundant with respect to a set of ground clauses* N if one of its premises is redundant with respect to N, or if $\lfloor E \rfloor$ is entailed by clauses in $\lfloor N \rfloor$ that are smaller than $\lfloor C \rfloor$. A nonground inference is *redundant with respect to a clause set* N if all its ground instances are redundant with respect to $\mathcal{G}_\Sigma(N)$.

An ArgCong or PosExt inference is *redundant with respect to a clause set* N if its premise is redundant with respect to N or if its conclusion is contained in N or redundant with respect to N.

We call N *saturated up to redundancy* if every inference from clauses in N is redundant with respect to N.

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 the following model. Let $\mathcal{U}_\tau = \mathcal{U}_\upsilon = \{a_1, a_2\}$ and $\mathcal{J}(\mathsf{a}) = a_1$. Let $\mathcal{U}_{\tau\to\upsilon} = \{f\}$ and $\mathcal{J}(\mathsf{f}) = f$. We interpret f as the identity function by setting $\mathcal{E}_{\tau,\upsilon}(f)(a_i) = a_i$ for $i = 1,2$. Let $\mathcal{U}_{\upsilon\to\tau} = \{g\}$ and $\mathcal{E}_{\upsilon,\tau}(g)(a_i) = a_2$ for $i = 1,2$. As a consequence, z cannot be interpreted as a function mapping $\mathcal{J}(\mathsf{a})$ to $\mathcal{J}(\mathsf{a})$ and hence the formula is true in this interpretation. 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 [40, 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 [3, 42]. We use the structure and notation of Waldmann's version [59], which is essentially the completeness proof for superposition without constraints [3] presented in the style of the proof for superposition with constraints by Nieuwenhuis and Rubio [41]. 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 $\mathsf{g}_0 \approx \mathsf{f}_0$ cannot be used directly to rewrite the clause $\mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0)$; instead, we first need to apply ARGCONG to derive $\mathsf{g}_1(x) \approx \mathsf{f}_1(x)$ 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 on the clause order. 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.

Then C is *productive*. 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—that is, for every element a of a universe of this interpretation, there exists a ground term t such that $[\![t]\!]^\xi_{\mathcal{T}_\Sigma^\emptyset/R} = a$. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_\Sigma^\emptyset/R$.

The following properties of the candidate interpretations can be shown exactly as in Waldmann's version of the first-order proof [59].

Lemma 5. *The rewrite systems R_C and R_∞ are confluent and terminating.*

Lemma 6. *If $D \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ is true in R_D, then D is true in R_∞ and R_C for all $C \succ D$.*

Lemma 7. *If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ D$.*

4.2 Lifting Lemmas

Following Waldmann's proof [59], 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*.

It is essential to the lifting lemmas that the selected literals of a clause correpond to the selected literals in its ground instances. However, there is no need to impose this as a restriction to the selection function. Instead, following Bachmair and Ganzinger [4, p. 45], let S be the selection function with respect to which N is saturated up to redundancy. We introduce another selection function S_N, such that each clause $C \in \mathcal{G}_\Sigma(N)$ is a ground instance of a clause $D \in N$ such that the selections $S(D)$ and $S_N(C)$ coincide. In the remainder of the proof, we adhere to the following convention:

Convention 8. When we speak about selected literals of clauses in N, it is with respect to the selection function S. When we speak about selected literals of clauses in $\mathcal{G}_\Sigma(N)$, it is with respect to the selection function S_N.

This auxiliary lemma is useful in the lifting lemma proofs:

Lemma 9. *Let σ be the most general unifier of s and s' (which can be assumed idempotent [58, Theorem 3.27]). Let θ be an arbitrary unifier of s and s'. Then $\sigma\theta = \theta$.*

Proof. Since σ is most general, there exists a substitution ρ such that $\sigma\rho = \theta$. Therefore, by idempotence, $\sigma\theta = \sigma\sigma\rho = \sigma\rho = \theta$. □

Lemma 10 (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 $C\theta$ (using S for C and S_N for $C\theta$ as mentioned in Convention 8). 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.*

Proof. EqRes: We assume that there is a EqRes inference from $C\theta$. This means that $C\theta$ is of the form $C\theta = C'\theta \vee s\theta \not\approx s'\theta$ where $C = C' \vee s \not\approx s'$, and $s\theta \not\approx s'\theta$ is selected or no literal of $C\theta$ is selected and $s\theta \not\approx s'\theta$ is maximal. Then the ground inference is

$$\frac{C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\,\text{EqRes}$$

where $s\theta$ and $s'\theta$ are unifiable and ground; hence $s\theta = s'\theta$. Since $s\theta \not\approx s'\theta$ is maximal (and nothing is selected) or selected in $C\theta$, $s \not\approx s'$ is maximal (and nothing is selected) or selected in C. Hence we have the inference

$$\frac{C' \vee s \not\approx s'}{C'\sigma \vee C_P}\ \textsc{EqRes}$$

where $\sigma = \mathrm{mgu}(s, s')$ and C_P are purification literals. By Lemma 9, we have $C'\sigma\theta = C'\theta$. Thus, the ground inference is the θ-ground instance of the nonground inference up to purification literals.

EQFACT: We assume that there is a EQFACT inference from $C\theta$. This means that $C\theta$ is of the form $C\theta = C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta$ where $s\theta \approx t\theta$ is maximal, no literal is selected in $C\theta$, $s\theta \not\preceq t\theta$, and $C = C' \vee s' \approx t' \vee s \approx t$. Then the ground inference is

$$\frac{C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta}{C'\theta \vee t\theta \not\approx t'\theta \vee s\theta \approx t'\theta}\ \textsc{EqFact}$$

where $s\theta$ and $s'\theta$ are unifiable and ground; hence $s\theta = s'\theta$. Since $s\theta \approx t\theta$ is maximal in $C\theta$, nothing is selected in $C\theta$, and $s\theta \not\preceq t\theta$, $s \approx t$ is maximal in C, nothing is selected in C, and $s \not\preceq t$. Hence we have the inference

$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma \vee C_P}\ \textsc{EqFact}$$

where $\sigma = \mathrm{mgu}(s, s')$ and C_P are purification literals. By Lemma 9, we have $(C' \vee t \not\approx t' \vee s \approx t')\sigma\theta = C'\theta \vee t\theta \not\approx t'\theta \vee s\theta \approx t'\theta$. Thus, the ground inference is the θ-ground instance of the nonground inference up to purification literals.

ARGCONG: We assume that there is an ARGCONG inference from $C\theta$. This means that $C\theta$ is of the form $C\theta = C'\theta \vee s\theta \approx s'\theta$, where $s\theta \approx s'\theta$ is strictly maximal, no literal is selected in $C\theta$, and $C = C' \vee s \approx s'$. Then the inference from $C\theta$ is

$$\frac{C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{x} \approx s'\theta\,\bar{x}}\ \textsc{ArgCong}$$

There cannot be purification literals because the premise is ground. Every ground instance of this inference has the form

$$\frac{C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{v} \approx s'\theta\,\bar{v}}\ \textsc{ArgCong}$$

Since $s\theta \not\approx s'\theta$ is strictly maximal in $C\theta$, $s \approx s'$ is strictly maximal in C. Since nothing is selected in $C\theta$, nothing is selected in C. Hence we have the inference

$$\frac{C' \vee s \approx s'}{C' \vee s\,\bar{x} \approx s'\,\bar{x} \vee C_P}\ \textsc{ArgCong}$$

where C_P are purification literals. Thus, the ground inference is the $\theta[x_1 \mapsto v_1, \ldots x_n \mapsto v_n]$-ground instance of this inference from C up to purification literals. □

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 11. 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 12 (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.*

Proof. We assume that there is a ground SUP inference of $D\theta$ in $C\theta$. Let $t \approx t' \in D$ and $[\neg]\, s \approx s' \in C$ be the literals involved in this inference. This means that $t\theta \approx t'\theta$ is strictly maximal and nothing is selected in $D\theta$. For positive superposition, $s\theta \approx s'\theta$ is strictly maximal and nothing is selected in $C\theta$. For negative superposition, either $s\theta \not\approx s'\theta$ is maximal and nothing is selected or $s\theta \not\approx s'\theta$ is selected in $C\theta$. Moreover, $D\theta \not\succeq C\theta$, $t\theta \not\prec t'\theta$, and $s\theta \not\prec s'\theta$. The ground inference is

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee [\neg]\, s\theta\langle t\theta\rangle_p \approx s'\theta}{D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_p \approx s'\theta}\ \textsc{Sup}$$

The inference conditions can be lifted: That $t\theta \approx t'\theta$ is strictly maximal in $D\theta$ implies that $t \approx t'$ is strictly maximal in D. That nothing is selected in $D\theta$ implies that nothing is selected in D. If $[\neg]\, s\theta \approx s'\theta$ is (strictly) maximal and nothing is selected in $C\theta$, then $[\neg]\, s \approx s'$ is (strictly) maximal and nothing is selected in C. If $s\theta \not\approx s'\theta$ is selected in $C\theta$, then $s \not\approx s'$ is selected in C. $D\theta \not\succeq C\theta$ implies $D \not\succeq C$. $t\theta \not\prec t'\theta$ implies $t \not\prec t'$. $s\theta \not\prec s'\theta$ implies $s \not\prec s'$.

The variable condition holds and that p is a position of s, because the ground inference is liftable. The argument subterm u of s at position p is unifiable with t, because θ is a unifier. So we have the nonground inference

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\, s\langle u\rangle_p \approx s'}{(D' \vee C' \vee [\neg]\, s\langle t'\rangle_p \approx s')\sigma \vee C_P}\ \textsc{Sup}$$

where $\sigma = \mathrm{mgu}(t,u)$ and C_P are purification literals. By Lemma 9, we have $(D' \vee C' \vee [\neg]\, s\langle t'\rangle_p \approx s')\sigma\theta = D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_p \approx s'\theta$. Thus, the ground inference is the θ-ground instance of the nonground inference up to purification literals. $\square$

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 13, about liftable inferences, is essentially as in the first-order case. The proof of Lemma 14, 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 13. *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 $\lfloor C\theta \rfloor$.*

Proof. We have a liftable SUP inference from $D\theta$ and $C\theta$ or an EQRES or EQFACT inference from $C\theta$. As shown in the lifting lemmas (Lemmas 10 and 12), up to purification literals in the conclusion, this inference is an instance of an inference from C (or from D and C for SUP inferences). Let $\tilde{E}$ be its conclusion. Since N is saturated up to redundancy, this inference is redundant with respect to N and hence the θ-ground instance of this inference is redundant with respect to $\mathcal{G}_\Sigma(N)$. By definition of inference redundancy, since $C\theta$ is not redundant with respect to $\mathcal{G}_\Sigma(N)$, $\lfloor \tilde{E}\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$.

By Lemma 9, we have $\tilde{E}\theta = E$ for the nonpurifying variants. In the purifying variants, we extend θ to the purification variables by copying the values of the original variable. Then the literals of $\lfloor \tilde{E}\theta \rfloor$ corresponding to purification literals are trivially false and hence $\lfloor E \rfloor$ is equivalent to $\lfloor \tilde{E}\theta \rfloor$. In all variants, it follows that $\lfloor E \rfloor$ is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. □

Lemma 14. *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 $\lfloor C\theta \rfloor$.*

Proof. Let $C\theta = C'\theta \vee [\neg]\, s\theta \approx s'\theta$ and $D\theta = D'\theta \vee t\theta \approx t'\theta$, where $[\neg]\, s\theta \approx s'\theta$ and $t\theta \approx t'\theta$ are the literals involved in the inference, $s\theta \succ s'\theta$, $t\theta \succ t'\theta$, and C', s, s', t, t' are the respective subclauses and terms in C and D.

Let R be an interpretation such that $\lfloor D'\theta \rfloor$ is false and the clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$ are true. Since $C\theta \succ D\theta$ by the SUP order conditions, it follows that $R \models \lfloor t\theta \approx t'\theta \rfloor$. We must show that $R \models \lfloor C\theta \rfloor$.

The inference from $D\theta$ and $C\theta$ can be nonliftable either because it is a superposition below a variable or because the variable condition does not hold for the corresponding inference between D and C.

CASE 1: We assume that it is a superposition below a variable, say, x. Let $t\theta \approx t'\theta$ be the strictly maximal literal of $D\theta$, where $t\theta \succ t'\theta$. Then $t\theta$ is an argument subterm of $x\theta$ and hence an argument subterm of $x\theta\, \bar{w}$ for any arguments $\bar{w}$. Let v be the term that we obtain by replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. It follows from the definition of R that $R \models \lfloor t\theta \approx t'\theta \rfloor$ and by congruence, $R \models \lfloor x\theta\,\bar{w} \approx v\,\bar{w} \rfloor$ for any arguments $\bar{w}$. Hence, $R \models \lfloor C\theta \rfloor$ if and only if $R \models \lfloor C[x \mapsto v]\theta \rfloor$. By the inference conditions we have $t\theta \succ t'\theta$, which implies $\lfloor C\theta \rfloor \succ \lfloor C[x \mapsto v]\theta \rfloor$ by compatibility with function contexts. Therefore, by the assumption about R, we have $R \models \lfloor C[x \mapsto v]\theta \rfloor$ and hence $R \models \lfloor C\theta \rfloor$.

CASE 2: The variable condition does not hold in the corresponding inference between D and C. Let u denote the superposed subterm of s.

Since the variable condition does not hold, u has a variable head and jells with $t \approx t'$. For the intensional variants, we even have $u \in \mathcal{V}$. For the nonpurifying variants, we additionally have $C\theta \succeq C''\theta$, where $C'' = C[x \mapsto \tilde{t}']$. By Definition 1, in all variants, u, t, and t' have the following form: $u = x\,\bar{v}$ for some variable x and n terms $\bar{v}$, for $n \geq 0$; $t = \tilde{t}\,\bar{x}_n$ and $t' = \tilde{t}'\,\bar{x}_n$, where $\bar{x}_n$ are variables that do not occur elsewhere in D. For the intensional variants, we have $n = 0$.

CASE 2.1 (PURIFYING CALCULI): First, we assume that x occurs only with arguments $\bar{v}$ in C. For the intensional variant, this must be the case because $n = 0$ and hence x can only occur without arguments by the definition of *pure* due to the selection restriction. Define a substitution θ' by $x\theta' = \tilde{t}'\theta$ and $y\theta' = y\theta$ for other variables y. Since $t\theta \succ t'\theta$, we have $C\theta \succ C\theta'$. Moreover, $C\theta' \in \mathcal{G}_\Sigma(N)$. Then $R \models \lfloor C\theta \rfloor$ by congruence, because $R \models \lfloor C\theta' \rfloor$ and $R \models \lfloor t\theta \approx t'\theta \rfloor$.

Now we assume that x occurs with arguments other than $\bar{v}$ in C. This can only happen in the extensional variant and by the selection restrictions, $[\neg]\, s\theta \approx s'\theta$ may not be selected in $C\theta$. Therefore, $s\theta$ is the maximal term in $C\theta$. Then $s \neq x$ and hence $\bar{v} \neq \varepsilon$ because otherwise $s\theta = x\theta$ would be smaller than the applied occurrence of $x\theta$ in $C\theta$.

Define a substitution θ'' such that $x\theta'' = \tilde{t}'\theta$, $y\theta'' = \tilde{t}'\theta$ for other variables y if $y\theta = s\theta$ and C contains the literal $x \not\approx y$, and $y\theta'' = y\theta$ otherwise.

We show that $C\theta \succ C\theta''$ by proving that no literal of $C\theta''$ is larger than the maximal literal $[\neg]\, s\theta \approx s'\theta$ of $C\theta$ and that $[\neg]\, s\theta \approx s'\theta$ appears more often in $C\theta$ than in $C\theta''$:

For a literal of the form $x \not\approx y$, we have $x\theta'' \prec s\theta$ and $y\theta'' \prec s\theta$. For literals that are not of this form, by the definition of *pure* in the extensional variant, x occurs always with arguments $\bar{v}$. Hence these literals are equal or smaller in $C\theta''$ than in $C\theta$, because $x\theta''\,\bar{v} \prec x\theta\,\bar{v}$ and $y\theta'' \preceq y\theta$. Therefore, no literal of $C\theta''$ is larger than the maximal literal $[\neg]\, s\theta \approx s'\theta$ of $C\theta$. Moreover, these inequalities show that every occurrence of $[\neg]\, s\theta \approx s'\theta$ in $C\theta''$ corresponds to an occurrence of $[\neg]\, s\theta \approx s'\theta$ in $C\theta$ that corresponds to a literal in C without the variable x. Since at least one occurrence of $[\neg]\, s\theta \approx s'\theta$ in $C\theta$ corresponds to a literal in C containing x, $[\neg]\, s\theta \approx s'\theta$ appears more often in $C\theta$ than in $C\theta''$. This concludes the argument that $C\theta \succ C\theta''$.

A POSEXT inference from D to $D' \vee \tilde{t} \approx \tilde{t}'$ is possible. Therefore, $D' \vee \tilde{t} \approx \tilde{t}'$ is in N or redundant with respect to to N because N is saturated up to redundancy. In either case, $R \models \lfloor (D' \vee \tilde{t} \approx \tilde{t}')\theta \rfloor$ because this clause is smaller than $C\theta$. Since $D'\theta$ is false in R, we have $R \models \lfloor \tilde{t}\theta \approx \tilde{t}'\theta \rfloor$.

For every literal of the form $x \not\approx y$, where $y\theta = s\theta$, the variable y can only occur without arguments in C because of the maximality of $s\theta$. Since $C\theta \succ C\theta''$, we have $R \models \lfloor C\theta'' \rfloor$. If for every literal of the form $x \not\approx y$, where $y\theta = s\theta$ we have $R \models \lfloor y\theta'' \approx y\theta \rfloor$, then $R \models \lfloor C\theta \rfloor$ by congruence. If for some literal of the form $x \not\approx y$ where $y\theta = s\theta$ we have $R \models \lfloor y\theta'' \not\approx y\theta \rfloor$, then $R \models \lfloor y\theta \not\approx x\theta \rfloor$ because $y\theta'' = \tilde{t}'\theta$, $R \models \lfloor \tilde{t}'\theta \approx \tilde{t}\theta \rfloor$, and $\tilde{t}\theta = x\theta$. Hence a literal of $C\theta$ is true in R and therefore $C\theta$ is true in R.

CASE 2.2 (NONPURIFYING CALCULI): Since the variable condition does not hold, we have $C\theta \succeq C''\theta$. We cannot have $C\theta = C''\theta$ because $x\theta = \tilde{t}\theta \neq \tilde{t}'\theta$ and x occurs in C. Hence, we have $C\theta \succ C''\theta$.

By the definition of R, $C\theta \succ C''\theta$ implies $R \models \lfloor C''\theta \rfloor$. We will use equalities that are true in R to rewrite $\lfloor C\theta \rfloor$ into $\lfloor C''\theta \rfloor$, which implies $R \models \lfloor C\theta \rfloor$ by congruence.

By saturation up to redundancy, for any type-correct tuple of fresh variables $\bar{z}$, we can use a POSEXT inference with premise D (if $n <$ length($\bar{z}$)) or ARGCONG inference with premise D (if $n >$ length($\bar{z}$)) or using D itself (if $n =$ length($\bar{z}$)) to show that up to variable renaming $D' \vee \tilde{t}\,\bar{z} \approx \tilde{t}'\,\bar{z}$ is in $\mathcal{G}_\Sigma(N \cup \text{Red}(N))$. Hence, $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u}$ is in $\mathcal{G}_\Sigma(N \cup \text{Red}(N))$ for any type-correct ground arguments $\bar{u}$.

First, we observe that whenever $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$ for some arguments $\bar{u}$, we have

$$R \models \lfloor \tilde{t}\theta\,\bar{u} \rfloor \approx \lfloor \tilde{t}'\theta\,\bar{u} \rfloor \qquad (\dagger)$$

To show this, we assume that $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$ and we distinguish two cases: If $t\theta$ is smaller than the maximal term of $C\theta$, all terms in $D'\theta$ are smaller than the maximal term of $C\theta$ and hence $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec C\theta$. If, on the other hand, $t\theta$ is equal to the maximal term of $C\theta$, $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than $t\theta$. Hence $\tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec t\theta \approx t'\theta$ and $D'\theta \vee \tilde{t}\theta\,\bar{u} \approx \tilde{t}'\theta\,\bar{u} \prec D\theta \prec C\theta$. In both cases, since $D'\theta$ is false in R, by the definition of R, $R \models \lfloor \tilde{t}\theta\,\bar{u} \rfloor \approx \lfloor \tilde{t}'\theta\,\bar{u} \rfloor$.

We proceed by a case distinction on whether $s\theta$ appears in a selected or in a maximal literal of $C\theta$. In both cases we provide an algorithm that establishes the equivalence of $C\theta$ and $C''\theta$ via rewriting using (†). This might seem trivial at first sight, but we can only use the equations (†) if $\tilde{t}\theta\,\bar{u}$ and $\tilde{t}'\theta\,\bar{u}$ are smaller than the maximal term of $C\theta$. Moreover, $\bar{u}$ might itself contain positions where we want to rewrite, such that the order of rewriting matters.

CASE 2.2.1: $s\theta$ is the maximal side of a maximal literal of $C\theta$. Then, since $C\theta \succ C''\theta$, every term in $C\theta$ and in $C''\theta$ is smaller or equal to $s\theta$. Let C_0 and $\tilde{C}_0$ be the clauses resulting from rewriting $\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor$ wherever possible in $\lfloor C\theta \rfloor$ and $\lfloor C''\theta \rfloor$, respectively. Since $\lfloor t\theta \rfloor$ is a subterm of $\lfloor s\theta \rfloor$, now every term in C_0 and $\tilde{C}_0$ is strictly smaller than $\lfloor s\theta \rfloor$.

We define $C_1, C_2, \ldots$ inductively as follows: Given C_i, choose a subterm of the form $\lfloor \tilde{t}\theta\,\bar{u} \rfloor$ where $\tilde{t}\theta\,\bar{u} \succ \tilde{t}'\theta\,\bar{u}$ or of the form $\lfloor \tilde{t}'\theta\,\bar{u} \rfloor$ where $\tilde{t}'\theta\,\bar{u} \succ \tilde{t}\theta\,\bar{u}$. Let C_{i+1} be the clause resulting from rewriting that subterm $\lfloor \tilde{t}\theta\,\bar{u} \rfloor$ to $\lfloor \tilde{t}'\theta\,\bar{u} \rfloor$ or that subterm $\lfloor \tilde{t}'\theta\,\bar{u} \rfloor$ to $\lfloor \tilde{t}\theta\,\bar{u} \rfloor$ in C_i, depending on which term was chosen.

Analogously, we define $\tilde{C}_1, \tilde{C}_2, \ldots$ by applying the same algorithm to $\tilde{C}_0$. In both cases, the process terminates because $\succ$ is of compatible with function contexts and well founded. Let C_* and $\tilde{C}_*$ be the respective final clauses.

The algorithm preserves the invariant that every term in C_i and $\tilde{C}_i$ is strictly smaller than $s\theta$. By congruence via (†), applied at every step of the algorithm, we know that C_* and $\lfloor C\theta \rfloor$ are equivalent in R and that $\tilde{C}_*$ and $\lfloor C''\theta \rfloor$ are equivalent in R as well.

We show that $C_* = \tilde{C}_*$. Assume that $C_* \neq \tilde{C}_*$. The algorithm preserves a second invariant, namely that $\lceil C_i \rceil$ and $\lceil \tilde{C}_j \rceil$ are equal except for positions where one contains $\tilde{t}\theta$

and the other one contains $\tilde{t}'\theta$. Consider the deepest position where $\lceil C_* \rceil$ and $\lceil \tilde{C}_* \rceil$ are different. The respective position in C_* and $\tilde{C}_*$ then contains $\lfloor \tilde{t}\theta\,\bar{u} \rfloor$ and $\lfloor \tilde{t}'\theta\,\bar{u} \rfloor$ or vice versa. The arguments $\bar{u}$ must be equal because we consider the deepest position possible. But then $\tilde{t}\theta\,\bar{u} \succ \tilde{t}'\theta\,\bar{u}$ or $\tilde{t}\theta\,\bar{u} \prec \tilde{t}'\theta\,\bar{u}$, which contradicts the fact that the algorithm terminated in C_* and $\tilde{C}_*$.

This shows that $C_* = \tilde{C}_*$. Hence $\lfloor C\theta \rfloor$ and $\lfloor C''\theta \rfloor$ are equivalent, which proves $R \models \lfloor C\theta \rfloor$.

CASE 2.2.2: $s\theta$ is the maximal side of a selected literal of $C\theta$. Then, by the selection restrictions, x cannot be the head of a maximal literal of C.

At every position where $x\,\bar{u}$ occurs in C with some (or no) arguments $\bar{u}$, we rewrite $(\tilde{t}\,\bar{u})\theta$ to $(\tilde{t}'\,\bar{u})\theta$ in $C\theta$ if $(\tilde{t}\,\bar{u})\theta \succ (\tilde{t}'\,\bar{u})\theta$. We start with the innermost occurrences of x, such that the order of the two terms at one step does not change by later rewriting.

Analogously, at every position where $x\,\bar{u}$ occurs in C with some (or no) arguments $\bar{u}$, we rewrite $(\tilde{t}'\,\bar{u})\theta$ to $(\tilde{t}\,\bar{u})\theta$ in $C''\theta$ if $(\tilde{t}'\,\bar{u})\theta \succ (\tilde{t}\,\bar{u})\theta$, again starting with the innermost occurrences.

We never rewrite at the top level of the maximal term of $C\theta$ or $C''\theta$ because x cannot be the head of a maximal literal of C. The two resulting clauses are identical because $C\theta$ and $C''\theta$ only differ at positions where x occurs in C. The rewritten terms are all smaller than the maximal term of $C\theta$. With (†), this implies that $R \models C\theta$ by congruence. □

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

Lemma 15 (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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.*

Proof. We use induction of the clause order $\succ$ on floor logic ground clauses and assume that (i)–(iv) are already satisfied for all clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. The 'if' part of (i) is obvious from the construction and that condition (iii) follows from (i) by Lemmas 6 and 7. So it remains to show (ii), (iv), and the 'only if' part of (i), i.e., we show the following: If $E_{\lfloor C\theta \rfloor} = \emptyset$ or $C\theta$ is redundant with respect to $\mathcal{G}_\Sigma(N)$ or $C\theta$ has selected literals, then $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$. Without loss of generality, we assume that the selected literals in $C \in N$ correspond to those in $C\theta$.

CASE 1: $C\theta$ is redundant with respect to $\mathcal{G}_\Sigma(N)$. Then $\lfloor C\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$. Hence $\lfloor C\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$.

CASE 2: $C\theta$ is not redundant with respect to $\mathcal{G}_\Sigma(N)$ and $C\theta$ contains an eligible negative literal. Let $s\theta \not\approx s'\theta$ with $s\theta \succeq s'\theta$ be one of these literals and $C'\theta$ the rest of the clause.

CASE 2.1: $s\theta = s'\theta$. Then there is an EQRES inference:

$$\frac{C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\;\textsc{EqRes}$$

By Lemma 13, $\lfloor C'\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$, which implies that $\lfloor C'\theta \rfloor$ and hence $\lfloor C\theta \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$.

CASE 2.2: $s\theta \succ s'\theta$. If $R \models \lfloor s\theta \not\approx s'\theta \rfloor$, then it follows directly that $R \models \lfloor C\theta \rfloor$. So we assume that $\lfloor s\theta \rfloor \downarrow_{R_{\lfloor C\theta \rfloor}} \lfloor s'\theta \rfloor$ (i.e., $\lfloor s\theta \rfloor$ and $\lfloor s'\theta \rfloor$ have the same normal form), which means that $R \models \lfloor s\theta \approx s'\theta \rfloor$. Since $s\theta \succ s'\theta$, $\lfloor s\theta \rfloor$ must be reducible by some rule in some $E_{\lfloor D\theta \rfloor} \subseteq R_{\lfloor C\theta \rfloor}$. Without loss of generality, we assume that the selected literals in $D \in N$ correspond to those in $D\theta$ and that C and D are variable disjoint; so we can use the same substitution θ. Let $D\theta = D'\theta \vee t\theta \approx t'\theta$ with $E_{\lfloor D\theta \rfloor} = \{\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor\}$.

There is a SUP inference

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta\langle t\theta \rangle \not\approx s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta}\ \textsc{Sup}$$

If this inference is not liftable, by Lemma 14, $\neg \lfloor D'\theta \rfloor$ and the clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$ imply $\lfloor C\theta \rfloor$. Since $\lfloor D\theta \rfloor$ is productive, $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$ by Lemma 7. By part (iii) of the induction hypothesis, all clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$ are true in $R_{\lfloor C\theta \rfloor}$. Therefore, $\lfloor C\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$.

If this inference is liftable, by Lemma 13, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ is then true in $R_{\lfloor C\theta \rfloor}$. Since $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, it follows that $R_{\lfloor C\theta \rfloor} \models C'\theta$ or $R_{\lfloor C\theta \rfloor} \models s\theta\langle t'\theta \rangle \not\approx s'\theta$. In the latter case, we have $R_{\lfloor C\theta \rfloor} \models s\theta\langle t\theta \rangle \not\approx s'\theta$ because $t\theta \to t\theta' \in R_{\lfloor C\theta \rfloor}$. Hence, in both cases, $R_{\lfloor C\theta \rfloor} \models C\theta$.

CASE 3: $C\theta$ is not redundant and contains no eligible negative literal. Then nothing is selected in $C\theta$ and $C\theta$ can be written as $C'\theta \vee s\theta \approx s'\theta$, where $s\theta \approx s'\theta$ is a maximal literal. If $E_{\lfloor C\theta \rfloor} = \{\lfloor s\theta \rfloor \to \lfloor s'\theta \rfloor\}$ or $R_{\lfloor C\theta \rfloor} \models \lfloor C'\theta \rfloor$ or $s\theta = s'\theta$, there is nothing to show, so assume that $E_{\lfloor C\theta \rfloor} = \emptyset$ and that $\lfloor C'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. Without loss of generality, $s\theta \succ s'\theta$.

CASE 3.1: $\lfloor s\theta \approx s'\theta \rfloor$ is maximal in $\lfloor C\theta \rfloor$, but not strictly maximal. Then $C\theta$ can be written as $C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta$, where $t\theta = s\theta$ and $t'\theta = s'\theta$. In this case, there is a EQFACT inference

$$\frac{C\theta \ = \ C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}\ \textsc{EqFact}$$

By Lemma 13, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta \rfloor}$, which implies that $\lfloor C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta \rfloor$ is true in $R_{\lfloor C\theta \rfloor}$. Since $t'\theta = s'\theta$ and hence $\lfloor t'\theta \not\approx s'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$, this implies that $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

CASE 3.2: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $\lfloor s\theta \rfloor$ is reducible by $R_{\lfloor C\theta \rfloor}$. Let $\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor \in R_{\lfloor C\theta \rfloor}$ be a rule that reduces $\lfloor s\theta \rfloor$. This rule stems from a productive clause $\lfloor D\theta \rfloor = \lfloor D'\theta \vee t\theta \approx t'\theta \rfloor$. Without loss of generality, we assume that the selected literals in $D \in N$ correspond to those in $D\theta$ and that C and D are variable disjoint; so we can use the same substitution θ.

We can now proceed in essentially the same way as in Case 2.2: There is a SUP inference

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta\langle t\theta\rangle \approx s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \approx s'\theta}\,\textsc{Sup}$$

If this inference is not liftable, by Lemma 14, $\neg\lfloor D'\theta\rfloor$ and the clauses in $\lfloor \mathcal{G}_\Sigma(N)\rfloor$ that are smaller than $\lfloor C\theta\rfloor$ imply $\lfloor C\theta\rfloor$. Since $\lfloor D\theta\rfloor$ is productive, $\lfloor D'\theta\rfloor$ is false in $R_{\lfloor C\theta\rfloor}$ by Lemma 7. By part (iii) of the induction hypothesis, all clauses in $\lfloor \mathcal{G}_\Sigma(N)\rfloor$ that are smaller than $\lfloor C\theta\rfloor$ are true in $R_{\lfloor C\theta\rfloor}$. Therefore, $\lfloor C\theta\rfloor$ is true in $R_{\lfloor C\theta\rfloor}$.

If this inference is liftable, by Lemma 13, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \approx s'\theta\rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N)\rfloor$ that are smaller than $\lfloor C\theta\rfloor$. By part (iii) of the induction hypothesis, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \approx s'\theta\rfloor$ is then true in $R_{\lfloor C\theta\rfloor}$. Since $\lfloor D'\theta\rfloor$ is false in $R_{\lfloor C\theta\rfloor}$, it follows that $R_{\lfloor C\theta\rfloor} \models C'\theta$ or $R_{\lfloor C\theta\rfloor} \models s\theta\langle t'\theta\rangle \approx s'\theta$. In the latter case, we have $R_{\lfloor C\theta\rfloor} \models s\theta\langle t\theta\rangle \approx s'\theta$ because $t\theta \to t\theta' \in R_{\lfloor C\theta\rfloor}$. Hence, in both cases, $R_{\lfloor C\theta\rfloor} \models C\theta$.

CASE 3.3: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $\lfloor s\theta\rfloor$ is irreducible with respect to $R_{\lfloor C\theta\rfloor}$. Since $C\theta$ is not redundant, we have $E_{\lfloor C\theta\rfloor} = \emptyset$. We assume that $C\theta$ is false. By the definition of $E_{\lfloor C\theta\rfloor}$, $\lfloor C'\theta\rfloor$ must be true in $R_{\lfloor C\theta\rfloor} \cup \{\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor\}$. Then $C'\theta = C''\theta \vee t\theta \approx t'\theta$, where the literal $\lfloor t\theta \approx t'\theta\rfloor$ is true in $R_{\lfloor C\theta\rfloor} \cup \{\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor\}$ and false in $R_{\lfloor C\theta\rfloor}$. In other words, $\lfloor t\theta\rfloor \downarrow_{R_{\lfloor C\theta\rfloor}\cup\{\lfloor s\theta\rfloor\to\lfloor s'\theta\rfloor\}} \lfloor t'\theta\rfloor$, but not $\lfloor t\theta\rfloor \downarrow_{R_{\lfloor C\theta\rfloor}} \lfloor t'\theta\rfloor$. Consequently, there is a rewrite proof of $\lfloor t\theta\rfloor \to^* \lfloor u\rfloor \leftarrow^* \lfloor t'\theta\rfloor$ by $R_{\lfloor C\theta\rfloor} \cup \{\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor\}$ in which the rule $\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor$ is used at least once. Without loss of generality, we assume that $t\theta \succeq t'\theta$. Since $s\theta \approx s'\theta \succ t\theta \approx t'\theta$ and $s\theta \succ s'\theta$ we can conclude that $s\theta \succeq t\theta \succ t'\theta$. But then there is only one possibility how the rule $\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor$ can be used in the rewrite proof: We must have $s\theta = t\theta$ and the rewrite proof must have the form $\lfloor t\theta\rfloor \to \lfloor s'\theta\rfloor \to^* \lfloor u\rfloor \leftarrow^* \lfloor t'\theta\rfloor$, where the first step uses $\lfloor s\theta\rfloor \to \lfloor s'\theta\rfloor$ and all other steps use rules from $R_{\lfloor C\theta\rfloor}$. Consequently, $\lfloor s'\theta \approx t'\theta\rfloor$ is true in $R_{\lfloor C\theta\rfloor}$. Now observe that there is an EQFACT inference

$$\frac{C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}\,\textsc{EqFact}$$

By Lemma 13, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N)\rfloor$ that are smaller than $\lfloor C\theta\rfloor$. By part (iii) of the induction hypothesis, these clauses are true in $R_{\lfloor C\theta\rfloor}$, which implies that $\lfloor C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta\rfloor$ is true in $R_{\lfloor C\theta\rfloor}$. Since the literal $\lfloor t'\theta \not\approx s'\theta\rfloor$ must be false in $R_{\lfloor C\theta\rfloor}$, this implies that $R_{\lfloor C\theta\rfloor} \models \lfloor C\theta\rfloor$, contradicting our assumption. □

Given a model R_∞ of $\lfloor \mathcal{G}_\Sigma(N)\rfloor$, we construct a model $R_\infty^\uparrow$ 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 16 (Argument congruence). *For all ground terms* $\mathsf{f}_m(\bar{s})$, $\mathsf{g}_n(\bar{t})$, *and* u, *if* $[\![\mathsf{f}_m(\bar{s})]\!]_{R_\infty}^\xi = [\![\mathsf{g}_n(\bar{t})]\!]_{R_\infty}^\xi$, *then* $[\![\mathsf{f}_{m+1}(\bar{s},u)]\!]_{R_\infty}^\xi = [\![\mathsf{g}_{n+1}(\bar{t},u)]\!]_{R_\infty}^\xi$.

Proof. What we want to show is equivalent to

$$R_\infty \models \mathsf{f}_m(\bar{s}) \approx \mathsf{g}_n(\bar{t}) \text{ implies } R_\infty \models \mathsf{f}_{m+1}(\bar{s},u) \approx \mathsf{g}_{n+1}(\bar{t},u)$$

which is equivalent to

$$\mathsf{f}_m(\bar{s}) \downarrow_{R_\infty} \mathsf{g}_n(\bar{t}) \text{ implies } \mathsf{f}_{m+1}(\bar{s},u) \leftrightarrow^*_{R_\infty} \mathsf{g}_{n+1}(\bar{t},u)$$

For every rewrite step rewriting a subterm, there is obviously an analogous rewrite step if the term u is appended at the top level. Therefore, it suffices to prove that

$$\mathsf{h}_k(\bar{v}) \rightarrow \mathsf{h}'_{k'}(\bar{v}') \in R_\infty \text{ implies } \mathsf{h}_{k+1}(\bar{v},u) \leftrightarrow^*_{R_\infty} \mathsf{h}'_{k'+1}(\bar{v}',u)$$

for all function symbols h,h' and all k,k'.

Since $\mathsf{h}_k(\bar{v}) \rightarrow \mathsf{h}'_{k'}(\bar{v}') \in R_\infty$, it must come from a productive clause of the form $\lfloor C\theta \rfloor = \lfloor C'\theta \rfloor \vee \mathsf{h}_k(\bar{v}) \approx \mathsf{h}'_{k'}(\bar{v}')$. Without loss of generality, we assume that the selected literals in $C \in N$ correspond to those in $C\theta$. We have an ArgCong inference from $C\theta$ with the following ground instance:

$$\frac{C'\theta \vee \lceil \mathsf{h}_k(\bar{v}) \approx \mathsf{h}'_{k'}(\bar{v}') \rceil}{C'\theta \vee \lceil \mathsf{h}_k(\bar{v},u) \approx \mathsf{h}'_{k'}(\bar{v}',u) \rceil \vee u_1 \not\approx u_1 \vee \cdots \vee u_l \not\approx u_l}\ \textsc{ArgCong}$$

(The additional literals $u_1 \not\approx u_1 \vee \cdots \vee u_l \not\approx u_l$ are due to purification. For the nonpurifying variants, $l = 0$.) By the lifting lemma (Lemma 10), this is a ground instance of an inference from C. By part (ii) of Lemma 15, a productive clause is never redundant; hence $C\theta$ is not redundant and therefore C is not redundant. Hence, the conclusion E of the inference from C is in $N \cup \mathrm{Red}(N)$. Therefore, the ground instance $\lfloor C'\theta \rfloor \vee \mathsf{h}_k(\bar{v},u) \approx \mathsf{h}'_{k'}(\bar{v}',u) \vee \lfloor u_1 \not\approx u_1 \vee \cdots \vee u_l \not\approx u_l \rfloor$ of $\lfloor E \rfloor$ is either contained in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ or it is entailed by clauses in $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Thus, it is true in R_∞, because R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. By Lemma 7, $\lfloor C'\theta \rfloor$ is false in R_∞. The literals $\lfloor u_1 \not\approx u_1 \vee \cdots \vee u_l \not\approx u_l \rfloor$ are obviously false. So, $\mathsf{h}_k(\bar{v},u) \approx \mathsf{h}'_{k'}(\bar{v}',u)$ must be true in R_∞ and $\mathsf{h}_{k+1}(\bar{v},u) \leftrightarrow^*_{R_\infty} \mathsf{h}'_{k'+1}(\bar{v}',u)$. □

Definition 17. Define an interpretation $R_\infty^\uparrow = (\mathcal{U}^\uparrow, \mathcal{E}^\uparrow, \mathcal{J}^\uparrow)$ in the ceiling logic as follows. Let $(\mathcal{U}, \mathcal{E}, \mathcal{J}) = R_\infty$. Let $\mathcal{U}_\tau^\uparrow = \mathcal{U}_{\lfloor \tau \rfloor}$ 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 $a \in \mathcal{U}_{\lfloor \tau \rightarrow \upsilon \rfloor}$, there exists a ground term $s : \tau \rightarrow \upsilon$ such that $[\![\lfloor s \rfloor]\!]_{R_\infty}^\xi = a$. Without loss of generality, we write $s = \mathsf{f}(\bar{s}_k)\ s_{k+1} \ldots s_m$. Then we have $a = [\![\mathsf{f}_m(\lfloor \bar{s}_m \rfloor)]\!]_{R_\infty}^\xi$ and define $\mathcal{E}^\uparrow$ by

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

It follows that $\mathcal{E}_{\tau,\upsilon}^\uparrow(a)\big([\![u]\!]_{R_\infty}^\xi\big) = [\![\mathsf{f}_{m+1}(\lfloor \bar{s}_m \rfloor, u)]\!]_{R_\infty}^\xi$ 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 $[\![\lfloor t \rfloor]\!]_{R_\infty}^\xi = a$. By Lemma 16, it follows from $[\![\lfloor s \rfloor]\!]_{R_\infty}^\xi = [\![\lfloor t \rfloor]\!]_{R_\infty}^\xi$ that

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

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 $R_\infty^\uparrow$ 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 $R_\infty^\uparrow$ is an extensional interpretation.

Lemma 18 (Model transfer to ceiling logic). *$R_\infty^\uparrow$ is a term-generated model of $\mathcal{G}_\Sigma(N)$.*

Proof. By Lemma 15, R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$, i.e., for all clauses $\lfloor C \rfloor \in \lfloor \mathcal{G}_\Sigma(N) \rfloor$ we have $[\![\lfloor C \rfloor]\!]^\xi_{R_\infty} = 1$.

We prove by induction on ground terms t and ground formulas φ of the ceiling logic that $[\![t]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor t \rfloor]\!]^\xi_{R_\infty}$ and $[\![\varphi]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor \varphi \rfloor]\!]^\xi_{R_\infty}$. It follows for all $C \in \mathcal{G}_\Sigma(N)$ that $[\![C]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor C \rfloor]\!]^\xi_{R_\infty} = 1$, and hence $(\mathcal{U}^\uparrow, \mathcal{E}^\uparrow, \mathcal{J}^\uparrow)$ is a model of $\mathcal{G}_\Sigma(N)$.

Let t be a ground ceiling term, and assume that $[\![t]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor t \rfloor]\!]^\xi_{R_\infty}$ for all subterms of t. If t is of the form $\mathsf{f}(\bar{t}_k)$, then

$$\begin{aligned}
[\![t]\!]^\xi_{R_\infty^\uparrow} &= \mathcal{J}^\uparrow(\mathsf{f})([\![\bar{t}_k]\!]^\xi_{R_\infty^\uparrow}) \\
&= \mathcal{J}(\mathsf{f}_k)([\![\bar{t}_k]\!]^\xi_{R_\infty^\uparrow}) \\
&\overset{\text{IH}}{=} \mathcal{J}(\mathsf{f}_k)([\![\lfloor \bar{t}_k \rfloor]\!]^\xi_{R_\infty}) \\
&= [\![\mathsf{f}_k(\lfloor \bar{t}_k \rfloor)]\!]^\xi_{R_\infty} \\
&= [\![\lfloor \mathsf{f}(\bar{t}_k) \rfloor]\!]^\xi_{R_\infty} = [\![\lfloor t \rfloor]\!]^\xi_{R_\infty}
\end{aligned}$$

If t is an application $t = t_1\ t_2$, where t_1 is of type $\tau \to \upsilon$, then writing t_1 as $t_1 = \mathsf{f}(\bar{s}_k)\ s_{k+1} \ldots s_m$ lets us derive

$$\begin{aligned}
[\![t_1\ t_2]\!]^\xi_{R_\infty^\uparrow} &= \mathcal{E}^\uparrow_{\tau,\upsilon}([\![t_1]\!]^\xi_{R_\infty^\uparrow})([\![t_2]\!]^\xi_{R_\infty^\uparrow}) \\
&\overset{\text{IH}}{=} \mathcal{E}^\uparrow_{\tau,\upsilon}([\![\lfloor t_1 \rfloor]\!]^\xi_{R_\infty})([\![\lfloor t_2 \rfloor]\!]^\xi_{R_\infty}) \\
&= \mathcal{E}^\uparrow_{\tau,\upsilon}([\![\mathsf{f}_m(\lfloor \bar{s}_m \rfloor]\!]^\xi_{R_\infty})([\![\lfloor t_2 \rfloor]\!]^\xi_{R_\infty}) \\
&\overset{\text{Def } \mathcal{E}^\uparrow}{=} [\![\mathsf{f}_{m+1}(\lfloor \bar{s}_m \rfloor, \lfloor t_2 \rfloor)]\!]^\xi_{R_\infty} \\
&= [\![\lfloor t_1\ t_2 \rfloor]\!]^\xi_{R_\infty}
\end{aligned}$$

So we have shown that $[\![t]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor t \rfloor]\!]^\xi_{R_\infty}$ for all terms t. Given that, the induction on formulas φ to show that $[\![\lfloor \varphi \rfloor]\!]^\xi_{R_\infty^\uparrow} = [\![\lfloor \varphi \rfloor]\!]^\xi_{R_\infty}$ is trivial.

It remains to show that $R_\infty^\uparrow$ is term-generated. Let $a \in \mathcal{U}_\tau^\uparrow$. Since $\mathcal{U}_\tau^\uparrow = \mathcal{U}_{\lfloor \tau \rfloor}$ and R_∞ is term-generated, we have a ground term t of the floor logic with $[\![t]\!]^\xi_{R_\infty} = a$. Using what we showed above, we have $[\![\lceil t \rceil]\!]^\xi_{R_\infty^\uparrow} = [\![t]\!]^\xi_{R_\infty} = a$. Hence, $R_\infty^\uparrow$ is term-generated. ◻

Lemma 19 (Model transfer to nonground clauses). *$R_\infty^\uparrow$ is a model of N.*

Proof. Let $(\forall x.\, C) \in N$. Then $R_\infty^\uparrow \models \forall x.\, C$ iff $[\![C]\!]^{\xi[x_i \mapsto a_i]}_{R_\infty^\uparrow} = 1$ for all ξ and a_i. Choose ground terms t_i such that $[\![t_i]\!]^\xi_{R_\infty^\uparrow} = a_i$; define θ such that $x_i\theta = t_i$, then $[\![C]\!]^{\xi[x_i \mapsto a_i]}_{R_\infty^\uparrow} = [\![C]\!]^{\xi \circ \theta}_{R_\infty^\uparrow} = [\![C\theta]\!]^\xi_{R_\infty^\uparrow} = 1$ since $C\theta \in \mathcal{G}_\Sigma(N)$ and $R_\infty^\uparrow \models \mathcal{G}_\Sigma(N)$ by Lemma 18. ◻

Lemma 20 (Completeness of the extensionality axioms). *If N contains the extensionality axioms, $R_\infty^\uparrow$ is extensional.*

Proof. Assume that the clause set N contains the extensionality axioms. By Lemma 19, the extensionality axioms are hence true in $R_\infty^\uparrow$.

Assume that $(\mathcal{U}^\uparrow, \mathcal{E}^\uparrow)$ is not extensional. Then $\mathcal{E}^\uparrow$ is not injective, i.e., there are $a \neq b \in \mathcal{U}^\uparrow_{\tau \to \upsilon}$, such that $\mathcal{E}^\uparrow(a) = \mathcal{E}^\uparrow(b)$. Let $\xi = \{x \mapsto a, y \mapsto b\}$. Then

$$[\![x\ (\mathsf{diff}(x,y)) \not\approx y\ (\mathsf{diff}(x,y)) \lor x \approx y]\!]^\xi_{R_\infty^\uparrow} = 0$$

because

$$[\![x\,(\mathsf{diff}(x,y))]\!]^{\xi}_{R^{\uparrow}_{\infty}} = \mathcal{E}^{\uparrow}(a)([\![\mathsf{diff}(x,y)]\!]^{\xi}_{R^{\uparrow}_{\infty}}) = \mathcal{E}^{\uparrow}(b)([\![\mathsf{diff}(x,y)]\!]^{\xi}_{R^{\uparrow}_{\infty}}) = [\![y\,(\mathsf{diff}(x,y))]\!]^{\xi}_{R^{\uparrow}_{\infty}}$$

but this is impossible since the extensionality axioms are true in $R^{\uparrow}_{\infty}$. □

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

Theorem 21 (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.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathcal{T}^{\emptyset}_{\Sigma}/R_\infty$) is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ according to part (iii) of Lemma 15. By Lemma 18, $R^{\uparrow}_{\infty}$ is a term-generated model of $\mathcal{G}_\Sigma(N)$. By Lemma 19, it is a model of N. If N contains the extensionality axioms, then $R^{\uparrow}_{\infty}$ is even an extensional model by Lemma 20. □

The dynamic view of the refutational completeness theorem holds as well. It can be shown exactly as in Waldmann's first-order proof [59].

5 Implementation

Zipperposition [20, 21] 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 [12]. 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 [54]. We have now also implemented a complete λ-free higher-order mode based on the four calculi described in this report, 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. One of the bugs we found during our implementation work occurred because argument positions shift when applying substitutions to applied variables. We resolved this by numbering argument positions in terms from right to left.

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

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

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 extensionality resolution rule [29].

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 due to the larger terms.

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. The encoding is implemented as a preprocessor, which makes all function symbols nullary and replaces all applications with a binary `app` symbol. For simplicity, the encoder uses a single polymorphic `app` symbol instead of a symbol family. Our experimental data is available online.[2] We used the developer version of Zipperposition, commit number 7fe2ebeb.[3] 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 with state-of-the-art higher-order provers that support a stronger logic.

We instantiated all variants with LPO [13] (which is nonmonotonic) and KBO [5] 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

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

[3] `https://github.com/c-cube/zipperposition/tree/7fe2ebeb`

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 TPTP TFF format and 1114 higher-order problems in TPTP THF format, 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 (excluding the applicative encoder). 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 [53] 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 applicative encoding and the nonpurifying calculi are comparable on unsatisfiable problems, which is probably indicative of the small size of the higher-order TPTP problems. The nonpurifying calculi saturate less often than the encoding, probably because of the selection restrictions, but the encoding is much slower, probably due to the additional symbols in the encoding. This difference in speed is smaller for the intensional calculi, a possible consequence of the argument congruence explosion. All in all, the comparison of the applicative enconding and the nonpurifying calculiis 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 the order's nonmonotonicity.

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 [45] Isabelle formalization of the refutational com-

		# 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

pleteness 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, independently developed by Cruanes [20, 21] and Wand [60], contribute to increasing the power of automatic provers. Detection of inconsistencies in axioms, as suggested by Schulz et al. [50], is important for large axiomatizations.

Also of interest is Bofill and Rubio's [15] 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 [47], Kerber [34], Dougherty [24], Dowek et al. [25], Hurd [33], Meng and Paulson [39], Obermeyer [44], and Czajka [22]. Encodings of types, such as those by Bobot and Paskevich [14] and Blanchette et al. [10], are also crucial to obtain a sound encoding of higher-order logic. These ideas are implemented in proof assistant tools such as HOLyHammer and Sledgehammer [11].

In the term rewriting community, λ-free higher-order logic is known as applicative first-order logic. First-order rewrite techniques can be applied to this logic via the applicative encoding. However, app being the only function symbol in this encoding has similar drawbacks as in theorem proving. Hirokawa et al. [31] propose a technique that resembles our $\lfloor\ \rfloor$ mapping to avoid those drawbacks.

Another line of research has focused on the development of automated proof procedures for higher-order logic. Robinson's [46], Andrews's [1], and Huet's [32] pioneering work stands out. Andrews [2] and Benzmüller and Miller [7] provide excellent surveys. The competitive higher-order automatic theorem provers include LEO-II [8] (based on unordered paramodulation), Satallax [17] (based on a tableau calculus and a SAT solver), AgsyHOL [38] (based on a focused sequent calculus and a generic narrowing engine), and Leo-III [52] (based on a pragmatic extension of superposition with no completeness guarantees). The Isabelle proof assistant [43] (which includes a tableau reasoner and a rewriting engine) and its Sledgehammer subsystem also participate in the higher-order division of the CADE ATP System Competition [54].

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 [49], SPASS [61], and Vampire [36], and integrate these in proof assistants to provide a high level of automation. With its stratified architecture, Otter-λ [6] 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 [57] has generalized E's data structures and algorithms to λ-free higher-order logic, assuming a monotonic KBO [5].

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 Christoph Benzmüller, Alexander Steen, Mark Summerfield, Sophie Tourret, and the anonymous reviewers for suggesting several textual improvements. 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.: Resolution in type theory. J. Symb. Log. 36(3), 414–432 (1971)
[2] 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)
[3] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)
[4] Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 19–99. Elsevier and MIT Press (2001)
[5] 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)
[6] Beeson, M.: Lambda logic. In: Basin, D.A., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 460–474. Springer (2004)
[7] 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)
[8] 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)
[9] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)
[10] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. 12(4:13), 1–52 (2016)
[11] Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reas. 9(1), 101–148 (2016)
[12] 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)
[13] 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)
[14] 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)
[15] Bofill, M., Rubio, A.: Paramodulation with non-monotonic orderings and simplification. J. Autom. Reason. 50(1), 51–98 (2013)
[16] Brand, D.: Proving theorems with the modification method. SIAM J. Comput. 4, 412–430 (1975)
[17] 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)

[18] Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. 13(5), 639–688 (2003)
[19] Church, A.: A formulation of the simple theory of types. J. Symb. Log. 5(2), 56–68 (1940)
[20] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)
[21] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)
[22] 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)
[23] Digricoli, V.J., Harrison, M.C.: Equality-based binary resolution. J. ACM 33(2), 253–289 (1986)
[24] Dougherty, D.J.: Higher-order unification via combinators. Theor. Comput. Sci. 114(2), 273–298 (1993)
[25] Dowek, G., Hardin, T., Kirchner, C.: Higher-order unification via explicit substitutions (extended abstract). In: LICS '95. pp. 366–374. IEEE (1995)
[26] 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)
[27] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)
[28] Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
[29] 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)
[30] Henkin, L.: Completeness in the theory of types. J. Symb. Log. 15(2), 81–91 (1950)
[31] Hirokawa, N., Middeldorp, A., Zankl, H.: Uncurrying for termination and complexity. J. Autom. Reasoning 50(3), 279–315 (2013)
[32] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)
[33] Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Di Vito, B., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics. pp. 56–68. NASA Technical Reports (2003)
[34] 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)
[35] Kop, C.: Higher Order Termination: Automatable Techniques for Proving Termination of Higher-Order Term Rewriting Systems. Ph.D. thesis, Vrije Universiteit Amsterdam (2012)
[36] 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)
[37] Leivant, D.: Higher order logic. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A., Siekmann, J.H. (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2, Deduction Methodologies, pp. 229–322. Oxford University Press (1994)
[38] 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)
[39] Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. 40(1), 35–60 (2008)
[40] Miller, D.A.: A compact representation of proofs. Studia Logica 46(4), 347–370 (1987)
[41] Nieuwenhuis, R., Rubio, A.: Basic superposition is complete. In: Krieg-Brückner, B. (ed.) ESOP '92. LNCS, vol. 582, pp. 371–389. Springer (1992)
[42] Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 371–443. Elsevier and MIT Press (2001)
[43] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)

[44] Obermeyer, F.H.: Automated Equational Reasoning in Nondeterministic λ-Calculi Modulo Theories $\mathscr{H}^*$. Ph.D. thesis, Carnegie Mellon University (2009)

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

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

[47] 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)

[48] Schmidt-Schauß, M.: Unification in a combination of arbitrary disjoint equational theories. J. Symb. Comput. 8, 51–99 (1989)

[49] 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)

[50] 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)

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

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

[53] 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)

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

[55] 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)

[56] 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)

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

[58] Waldmann, U.: Automated reasoning I. Lecture notes, Max-Planck-Institut für Informatik (2015), `http://resources.mpi-inf.mpg.de/departments/rg1/teaching/autrea-ws15/script.pdf`

[59] 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`

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

[61] 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)