SUPERPOSITION FOR LAMBDA-FREE HIGHER-ORDER LOGIC

ALEXANDER BENTKAMP, JASMIN BLANCHETTE, SIMON CRUANES, AND UWE WALDMANN

Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
e-mail address: a.bentkamp@vu.nl

Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
e-mail address: j.c.blanchette@vu.nl

Aesthetic Integration, 600 Congress Ave., Austin, Texas, 78701, USA
e-mail address: simon@imandra.ai

Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
e-mail address: uwe@mpi-inf.mpg.de

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 and Isabelle benchmarks. They appear promising as a stepping stone towards complete, efficient automatic theorem provers for full higher-order logic.

1. Introduction

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

As a stepping stone towards full higher-order logic, in this article we restrict our attention to a λ-free clausal fragment of polymorphic 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 $\mathsf{pow}_\tau : nat \to (\tau \to \tau) \to \tau \to \tau$ (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

Key words and phrases: superposition calculus, lambda-free higher-order logic, refutational completeness. Extended version of Bentkamp et al.: Superposition for lambda-free higher-order logic [10].

Preprint submitted to
Logical Methods in Computer Science

expressiveness of full higher-order logic can be recovered by introducing SK-style combinators to represent λ-abstractions and proxies for the logical symbols [40, 53].

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 [40] and is employed fruitfully in tools such as Sledgehammer [16, 45], 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 [41, 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 [15, 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 [45]. 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 [11]. 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) [18] and the Knuth–Bendix order (KBO) [6] 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 into 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 [27] (Section 5). We evaluate them on first- and higher-order TPTP benchmarks [62, 63] 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.

An earlier version of this work was presented at IJCAR 2018 [10]. This article extends the conference paper with a detailed completeness proof and more explanations. We have extended the logic with polymorphism, leading to minor modifications of the calculus. We have also simplified the presentation of the clausal fragment of the logic that interests us. Moreover, we have updated the empirical evaluation based on recent improvements of the Zipperposition prover.

2. Logic

Refutational completeness of calculi for higher-order logic (also called simple type theory) [25,34] is usually stated with respect to Henkin semantics [11,36], 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 [43, Section 5.4]. Unlike other higher-order logics [32], 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 [46]. We achieve this using a *hybrid logic* that supports both mandatory (uncurried) and optional (curried) arguments, inspired by higher-order term rewriting [41]. 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 [6,18], 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 $\mathsf{f}\ x\ y$.

2.1. **Syntax.** We fix a set Σ_{ty} of type constructors with arities and a set $\mathcal{V}_{\mathsf{ty}}$ of type variables. We require at least one nullary type constructor $\iota \in \Sigma_{\mathsf{ty}}$ and a binary type constructor $\to \in \Sigma_{\mathsf{ty}}$ to be present. A type τ, υ of λ-free higher-order logic is either a type variable $\alpha \in \mathcal{V}_{\mathsf{ty}}$ or has the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. 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 $n \geq 0$. In our hybrid logic, a type declaration is an expression of the form $\Pi\bar{\alpha}_m.\ \bar{\tau}_n \Rightarrow \tau$ (or simply $\bar{\tau}_n \Rightarrow \tau$, $\Pi\bar{\alpha}_m.\ \tau$, or τ if if m or n is 0), where all type variables occurring in $\bar{\tau}_n \Rightarrow \tau$ belong to $\bar{\alpha}_m$.

We fix a nonempty set Σ of symbols with type declarations, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\ \bar{\tau} \Rightarrow \tau$ or f, and a set $\mathcal{V}$ of typed variables, written as $x : \tau$ or x. The sets $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$ form the logic's signature. 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 is defined inductively as follows. Every variable in X is a term. If $\mathsf{f} : \Pi\bar{\alpha}_m.\ \bar{\tau}_n \Rightarrow \tau$ is a symbol, $\bar{\upsilon}_m$ are types, and for $i \in \{1, \dots n\}$ $u_i : \tau_i$ is a term, then $\mathsf{f}\langle\bar{\upsilon}_m\rangle(\bar{u}_n) : \tau$ 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 [24], terms can be decomposed in a unique way as a head ζ applied to zero or more arguments: $\zeta\ s_1 \dots s_n$ 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 $\{h \mapsto \mathsf{f}\}$.

An equation $s \approx t$ is formally an unordered pair of terms s and t. A literal is an equation or a negated equation, written $\neg\ s \approx t$ or $s \not\approx t$. A clause $L_1 \vee \dots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

2.2. **Semantics.** A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The set $\mathcal{U}$ is a nonempty collection of nonempty sets, called *universes*. The function $\mathcal{J}_{\mathsf{ty}}$ associates a function $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \to \mathcal{U}$ with each n-ary type constructor κ. A *type valuation* ξ is a function that maps every type variable to a universe. The *denotation* of a type for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a type valuation ξ is defined by $[\![\alpha]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \xi(\alpha)$ and $[\![\kappa(\bar{\tau})]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\tau}]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}})$. Here and elsewhere, we abuse notation by applying an operation on a tuple when it must be applied elementwise, such as $[\![\bar{\tau}_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$ standing for $[\![\tau_1]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}, \dots, [\![\tau_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$.

A type valuation ξ can be extended to be a *valuation* by additionally assigning an element $\xi(x) \in [\![\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$ to each variable $x : \tau$. An *interpretation function* $\mathcal{J}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ associates with each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\ \bar{\tau}_n \Rightarrow \tau$ and universe tuple $\bar{U}_m \in \mathcal{U}^m$ a function $\mathcal{J}(\mathsf{f}, \bar{U}_m) : [\![\bar{\tau}_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} \to [\![\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to U_i. Loosely following Fitting [33, Section 2.5], an *extension function* $\mathcal{E}$ associates to any pair of universes $U_1, U_2 \in \mathcal{U}$ a function $\mathcal{E}_{U_1,U_2} : \mathcal{J}_{\mathsf{ty}}(\to)(U_1, U_2) \to (U_1 \to U_2)$. Together, a type interpretation, an interpretation function, and an extension function form an *interpretation* $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$.

An interpretation is *extensional* if $\mathcal{E}_{U_1,U_2}$ is injective for all U_1, U_2. Both intensional and extensional logics are widely used for interactive theorem proving; for example, Coq's calculus of inductive constructions is intensional [13], whereas Isabelle/HOL is extensional [49]. The semantics is *standard* if $\mathcal{E}_{U_1,U_2}$ is bijective for all U_1, U_2.

For an interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}}, \mathcal{J}, \mathcal{E})$ and a valuation ξ, the denotation of a term is defined as follows: For variables x, let $[\![x]\!]^\xi_{\mathcal{I}} = \xi(x)$. For symbols f, let $[\![\mathsf{f}\langle\bar{\tau}\rangle(\bar{t})]\!]^\xi_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}})([\![\bar{t}]\!]^\xi_{\mathcal{I}})$. For applications $s\ t$ of a term $s : \tau \to \upsilon$ to a term $t : \tau$, let $U_1 = [\![\bar{\tau}]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$, $U_2 = [\![\bar{\upsilon}]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$, and $[\![s\ t]\!]^\xi_{\mathcal{I}} = \mathcal{E}_{U_1,U_2}([\![s]\!]^\xi_{\mathcal{I}})([\![t]\!]^\xi_{\mathcal{I}})$. If t is a ground term, we also write $[\![t]\!]_{\mathcal{I}}$ for the denotation of t because it does not depend on the valuation.

An equation $s \approx t$ is true in $\mathcal{I}$ for ξ if $[\![s]\!]^\xi_{\mathcal{I}} = [\![t]\!]^\xi_{\mathcal{I}}$; otherwise, it is false. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if at least one of its literals is true. The interpretation $\mathcal{I}$ is a model of a clause C, written $\mathcal{I} \models C$, if C is true in $\mathcal{I}$ for all valuations ξ. It is a model of a set of clauses if it is a model of all contained clauses.

For example, given the signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V}) = (\{\iota, \to\}, \{\}, \{\mathsf{a} : \iota\}, \{h : \iota \to \iota\})$, the clause $h\ \mathsf{a} \not\approx \mathsf{a}$ has an extensional model with $\mathcal{U} = \{U_1, U_2\}$, $U_1 = \{a, b\}$ $(a \neq b)$, $U_2 = \{f\}$, $\mathcal{J}_{\mathsf{ty}}(\iota) = U_1$, $\mathcal{J}_{\mathsf{ty}}(\to)(U_1, U_1) = U_2$, $\mathcal{J}(\mathsf{a}) = a$, $\mathcal{E}_{U_1,U_1}(f)(a) = \mathcal{E}_{U_1,U_1}(f)(b) = b$.

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 $\mathsf{f}\langle\bar{\tau}\rangle(\bar{s}, t', \bar{u})\ \bar{v} \succ \mathsf{f}\langle\bar{\tau}\rangle(\bar{s}, t, \bar{u})\ \bar{v}$ 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 inductively as follows. A term s is an argument subterm of t if either $s = t$; or $t = \mathsf{f}\langle\bar{\tau}\rangle(\bar{u})\ \bar{v}$ and s is an argument subterm of u_i or v_i for some i; or $t = x\ \bar{v}$ and s is an argument subterm of v_i for some 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 $\mathsf{f}\ \mathsf{a}$ are subterms of $\mathsf{f}\ \mathsf{a}\ \mathsf{b}$, but not argument subterms; correspondingly, $[\,]\ \mathsf{a}\ \mathsf{b}$ and $[\,]\ \mathsf{b}$ are not function contexts. The literal and clause orders are defined from the term order as multiset extensions in the standard way [3].

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.

The following four rules are common to all four calculi. We regard positive and negative superposition as two cases of the same rule

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

where $\sigma = \mathrm{mgu}(t, u)$; $t\sigma \not\preceq t'\sigma$; $s\langle u\rangle\sigma \not\preceq s'\sigma$; $(t \approx t')\sigma$ is strictly eligible in $D\sigma$; $(s\langle u\rangle \approx s')\sigma$ is eligible in $C\sigma$ and, if positive, even strictly eligible; and $C\sigma \not\preceq D\sigma$. Moreover, the *variable condition* must hold, which varies from one calculus to another and is specified below.

The equality resolution and equality factoring rules are almost identical to their standard counterparts:

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

Side conditions for EqRes: $\sigma = \mathrm{mgu}(s, s')$ and $(s \not\approx s')\sigma$ is eligible in the premise. Side conditions for EqFact: $\sigma = \mathrm{mgu}(s, s')$; $s'\sigma \not\preceq t'\sigma$; $s\sigma \not\preceq t\sigma$; and $(s \approx t)\sigma$ is eligible in the premise.

The argument congruence rule compensates for the limitation that the superposition rule applies only to argument subterms:

$$\frac{C' \vee s \approx s'}{C'\sigma \vee (s\sigma)\ x \approx (s'\sigma)\ x}\ \textsc{ArgCong}$$

Here, the literal $s\sigma \approx s'\sigma$ is strictly eligible in $(C' \vee s \approx s')\sigma$, and x is a fresh variable. The substitution σ is the most general type substitution that ensures well-typedness of the conclusion. In particular, if s is of function type, σ is the identity, and if the type of s is a

type variable α, $\sigma = \{\alpha \mapsto (\beta \to \gamma)\}$ for fresh type variables β and γ. If s has some other type, no ArgCong inference is possible for that literal.

For the **intensional nonpurifying** variant, the variable condition of the Sup rule is as follows: "If $u \in \mathcal{V}$, there exists a grounding substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C\{u \mapsto t'\}\sigma\theta$." This condition generalizes the standard condition that $u \notin \mathcal{V}$. The two coincide if C is first-order or if the term order is monotonic. In some cases involving nonmonotonicity, the variable condition effectively mandates Sup inferences at variable positions of the right premise, but never below. We will call theses inferences *at variables*.

For the **extensional nonpurifying** calculus, the variable condition uses the following definition.

Definition 3.1. 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.

Using the naming convention from Definition 3.1 for $\tilde{t}'$, the variable condition can be stated as follows: "If u has a variable head x and jells with the literal $t \approx t' \in D$, there must exist a grounding substitution θ with $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{x \mapsto \tilde{t}'\}$." If C is first-order, this amounts to $u \notin \mathcal{V}$. Since the order is compatible with function contexts, the substitution θ can only exist if x occurs applied in C.

Moreover, the extensional nonpurifying calculus has one additional rule, the positive extensionality rule, and one axiom, the extensionality axiom. The rule is

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

where $\bar{x}$ is a tuple of distinct variables that do not occur in C', s, or s', and $s\ \bar{x} \approx s'\ \bar{x}$ is strictly eligible in the premise. The extensionality axiom uses a polymorphic Skolem symbol $\mathsf{diff} : \Pi\alpha, \beta.\ (\alpha \to \beta)^2 \Rightarrow \alpha$ characterized by the axiom

$$x\ (\mathsf{diff}\langle\alpha, \beta\rangle(x, y)) \not\approx y\ (\mathsf{diff}\langle\alpha, \beta\rangle(x, y)) \vee x \approx y$$

Unlike the nonpurifying variants, the purifying calculi never perform superposition at variables. Instead, they rely on purification [22, 29, 54, 58] (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 to x. 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 **intensional purifying** calculus, the purification $\mathit{pure}(C)$ of clause C is defined as the result of the following procedure. Choose a variable x that occurs applied in C and also unapplied in a literal of C that is not of the form $x \not\approx y$. If no such variable exists, terminate. Otherwise, replace all unapplied occurrences of x in C by a fresh variable x' and add the purification literal $x' \not\approx x$. Then repeat the procedure with another variable. For example,

$$\mathit{pure}(x\ \mathsf{a} \approx x\ \mathsf{b} \vee \mathsf{f}\ x \approx \mathsf{g}\ x) = x\ \mathsf{a} \approx x\ \mathsf{b} \vee \mathsf{f}\ x' \approx \mathsf{g}\ x' \vee x \not\approx x'$$

The variable condition is "$u \notin \mathcal{V}$." The conclusion C of ArgCong is changed to $\mathit{pure}(C)$; the other rules preserve purity.

In the **extensional purifying** calculus, $\mathit{pure}(C)$ is defined as follows. Choose a variable x that occurs in distinct argument subterms $x\ \bar{u}$ and $x\ \bar{v}$ in literals of C not of the form $x \not\approx y$. If no such variable exists, terminate. Otherwise, replace all argument subterms $x\ \bar{v}$ with $x'\ \bar{v}$,

where x' is fresh, and add the purification literal $x' \not\approx x$. Then repeat the procedure with another variable. For example,

$$\mathit{pure}(x\ \mathsf{a} \approx x\ \mathsf{b} \vee \mathsf{f}\ x \approx \mathsf{g}\ x) = x\ \mathsf{a} \approx x'\ \mathsf{b} \vee \mathsf{f}\ x'' \approx \mathsf{g}\ x'' \vee x' \not\approx x \vee x'' \not\approx x$$

Like the extensional nonpurifying calculus, this calculus variant also contains the PosExt rule and the extensionality axiom defined above. The variable condition is "either u has a non-variable head or u does not jell with the literal $t \approx t' \in D$." The conclusion E of each rule is changed to $\mathit{pure}(E)$, except for PosExt, which preserves 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 3.2. 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 companion 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 at argument subterms, mirroring the requirement that the term order enjoy compatibility with function contexts. The ArgCong rule then makes it possible to simulate superposition at non-argument subterms. However, in conjunction with the Sup rules, ArgCong can exhibit an unpleasant behavior, which we call *argument congruence explosion*:

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

In both derivation trees, 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.

Example 3.3. Consider a left-to-right LPO [18] instance with precedence $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, and consider the following unsatisfiable clause set:

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

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.

Example 3.4. In the following unsatisfiable clause set, the only possible inference from these clauses in the extensional nonpurifying calculus is PosExt, showing its necessity.

$$\mathsf{g}\ x \approx \mathsf{f}\ x \qquad \mathsf{g} \not\approx \mathsf{f} \qquad x\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \not\approx y\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \vee x \approx y$$

The same argument applies for the purifying variant with the difference that the third clause 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.

Example 3.5. Consider the following clause set:

$$\mathsf{k}\ (\mathsf{g}\ x) \approx \mathsf{k}\ (x\ \mathsf{b}) \quad \mathsf{k}\ (\mathsf{f}\ (\mathsf{h}\ \mathsf{a})\ \mathsf{b}) \not\approx \mathsf{k}\ (\mathsf{g}\ \mathsf{h}) \quad \mathsf{f}\ (\mathsf{h}\ \mathsf{a}) \approx \mathsf{h} \quad \mathsf{f}\ (\mathsf{h}\ \mathsf{a})\ x \approx \mathsf{h}\ x$$
$$x\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \not\approx y\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \vee x \approx y$$

Using a left-to-right LPO [18] 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 with respect to the extensional purifying calculus when omitting purification. It also quickly saturates using the extensional nonpurifying calculus 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 kept, but we conjecture that it is incomplete in general.

We initially considered inference rules instead of the extensionality axiom. However, we did not find a set of inference rules that is complete and leads to fewer inferences than the extensionality axiom. We considered the PosExt rule described above in combination with the following rule:

$$\frac{C \vee s \not\approx t}{C \vee s\ (\mathsf{sk}(\bar{x}_n)) \not\approx t\ (\mathsf{sk}(\bar{x}_n))}\ \textsc{NegExt}$$

where sk is a fresh n-ary Skolem symbol and $\bar{x}_n$ are the variables occurring free in the the literal $s \not\approx t$. However, these two rules do not suffice for a refutationally complete calculus, as the following example demonstrates:

Example 3.6. Consider the clause set

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

Assuming that all four equations are oriented from left to right, this set is saturated with respect to the extensional calculi if the extensionality axiom is replaced by NegExt; yet it is unsatisfiable in an extensional logic.

Example 3.7. 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 equations provide two definitions of addition on Peano numbers:

$$\mathsf{add_L}\ 0\ y \approx y \qquad \mathsf{add_R}\ x\ 0 \approx x$$
$$\mathsf{add_L}\ (\mathsf{S}\ x)\ y \approx \mathsf{add_L}\ x\ (\mathsf{S}\ y) \qquad \mathsf{add_R}\ x\ (\mathsf{S}\ y) \approx \mathsf{add_R}\ (\mathsf{S}\ x)\ y$$

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.

More precisely, suppose we can simplify one side of the negated conjecture by rewriting, e.g. to $\mathsf{S}^{100}\ y \not\approx \mathsf{add_R}\ y\ (\mathsf{S}^{100}\ 0)$. After that, we must perform 100 Sup inferences to derive $\mathsf{add_R}\ y\ (\mathsf{S}\ 0) \approx \mathsf{S}\ y$, $\mathsf{add_R}\ y\ (\mathsf{S}\ (\mathsf{S}\ 0)) \approx \mathsf{S}\ (\mathsf{S}\ y)$, …, $\mathsf{add_R}\ y\ (\mathsf{S}^{100}\ 0) \approx \mathsf{S}^{100}\ y$. Using this last clause, we can easily derive $\bot$.

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 λ-free higher-order terms into ground monomorphic first-order terms, with $\lceil\ \rceil$ as its inverse. Accordingly, we refer to the source logic as the *ceiling logic* and to the target logic as the *floor logic*. Essentially, the encoding indexes each symbol occurrence with its type arguments and argument count. Thus, $\lfloor \mathsf{f} \rfloor = \mathsf{f}_0$, $\lfloor \mathsf{f}\ \mathsf{a} \rfloor = \mathsf{f}_1(\mathsf{a}_0)$, and $\lfloor \mathsf{g}\langle\iota\rangle \rfloor = \mathsf{g}_0^\iota$. This is enough to disable argument congruence; for example, $\{\mathsf{f} \approx \mathsf{h}, \mathsf{f}\ \mathsf{a} \not\approx \mathsf{h}\ \mathsf{a}\}$ is unsatisfiable, whereas its encoding $\{\mathsf{f}_0 \approx \mathsf{h}_0, \mathsf{f}_1(\mathsf{a}_0) \not\approx \mathsf{h}_1(\mathsf{a}_0)\}$ 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 ground signature $(\Sigma_{\mathsf{ty}}, \{\}, \Sigma, \{\})$ in the ceiling logic, we define a signature $(\Sigma_{\mathsf{ty}}, \{\}, \Sigma^{\downarrow}, \{\})$ in the floor logic as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is uninterpreted in the floor logic. For each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\ \bar{\tau}_k \Rightarrow \tau_{k+1} \to \cdots \to \tau_n \to \tau$ in Σ, where τ is atomic, we introduce symbols $\mathsf{f}_l^{\bar{\upsilon}_m} \in \Sigma^{\downarrow}$ with argument types $\bar{\tau}_l\sigma$ and return type $(\tau_{l+1} \to \cdots \to \tau_n \to \tau)\sigma$, where $\sigma = \{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$, for each tuple

of ground types $\bar{\upsilon}_m$ and each $l \in \{k, \ldots, n\}$. The translation of ground terms is defined recursively as $\lfloor \mathsf{f}\langle\bar{\upsilon}_m\rangle(\bar{u}_k)\ u_{k+1}\ \ldots\ u_l \rfloor = \mathsf{f}_l^{\bar{\upsilon}_m}(\lfloor \bar{u}_l \rfloor)$.

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

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 $\lceil\ \rceil$:

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

(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 subterms in the floor logic (Lemma 3.8), 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 first-order logic does not have optional arguments.

To state the correspondence between argument subterms in the ceiling logic and subterms in the floor logic explicitly, we define positions of argument subterms as follows. A term s is an argument subterm at position ϵ of s. If a term s is an argument subterm at position p of u_i for some $1 \leq i \leq n$, then s is an argument subterm at position $i.p$ of $\mathsf{f}\langle\bar{\tau}\rangle(\bar{u}_k)\ u_{k+1}\ \ldots\ u_n$ and of $x\ \bar{u}_n$. On the floor logic, positions are defined as usual in first-order logic.

Lemma 3.8. *For all ground 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 $t[s]_p = \mathsf{f}_n^{\bar{\upsilon}_m}(u_1, \ldots, u_n)$ 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 $\lceil t[s]_p \rceil = \lceil t \rceil\langle\lceil s \rceil\rangle_p$. □

Lemma 3.9. *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 3.8, 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 3.8 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, $t[s]_p \succ s$. □

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.

A ground instance of an inference is usually defined as an inference obtained by applying a substitution to all premises and to the conclusion. In the purifying calculi however, we cannot employ this notion because nonground conclusions may contain purification literals (corresponding to applied variables) not present in the ground conclusions. Instead, given an inference I of the form $\bar{C} \vdash \mathit{pure}(E)$, we refer to the inferences $\bar{C}\theta \vdash E\theta$ as ground instances of I *up to purification*. For nonpurifying inferences, any ground instance is a ground instance up to purification.

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 use redundancy of clauses to avoid the complication of defining what ground instances of these inferences are.

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 up to purification 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 nonclausal problem 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 $\mathsf{f} : \iota \to \kappa$, 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 $\mathcal{U} = \{U_\iota, U_\kappa, U_f\}$ with $U_\iota = \{i_1, i_2\}$, $U_\kappa = \{k_1, k_2\}$ and $U_f = \{f\}$. Let $\mathcal{J}_{\mathsf{ty}}(\iota) = U_\iota$, $\mathcal{J}_{\mathsf{ty}}(\kappa) = U_\kappa$ and $\mathcal{J}_{\mathsf{ty}}(\to)(U_\iota, U_\kappa) = \mathcal{J}_{\mathsf{ty}}(\to)(U_\kappa, U_\iota) = U_f$. Let $\mathcal{J}(\mathsf{a}) = k_1$ and $\mathcal{J}(\mathsf{f}) = f$. We interpret f as a bijection between U_ι and U_κ by defining $\mathcal{E}_{U_\iota, U_\kappa}(f)(i_n) = k_n$ for $n \in \{1, 2\}$. For functions from κ to ι on the other hand, we deny the existence of functions mapping k_1 to i_1 by defining $\mathcal{E}_{U_\kappa, U_\iota}(f)(k_n) = i_2$ for $n \in \{1, 2\}$. 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 [46, 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 with 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 $\mathcal{E}_{U_\iota, U_\kappa}$ 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 [3,48]. We use the structure and notation of Waldmann's version [66], 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 [47]. 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 argument subterms. This is reflected in our proof by the reliance on the floor logic from Section 3.3. In that logic, the equation $\mathsf{g}_0 \approx \mathsf{f}_0$, which corresponds to the ceiling equation $\mathsf{g} \approx \mathsf{f}$, cannot be used directly to rewrite the clause $\mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0)$, which corresponds to the ceiling equation $\mathsf{g}\ \mathsf{a} \not\approx \mathsf{f}\ \mathsf{a}$. Instead, we first need to apply ArgCong to derive $\mathsf{g}\ x \approx \mathsf{f}\ x$, which after grounding and transfer to the floor logic yields $\mathsf{g}_1(\mathsf{a}_0) \approx \mathsf{f}_1(\mathsf{a}_0)$. 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 completeness 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 $\mathsf{f}_k^{\bar{\upsilon}}, \ldots, \mathsf{f}_n^{\bar{\upsilon}}$ 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 a symbol with type declaration $\Pi\alpha.\ \alpha$.

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 $\mathcal{T}_{\Sigma\downarrow}^{\emptyset}/R$ such that for every *ground* equation $s \approx t$, we have $\mathcal{T}_{\Sigma\downarrow}^{\emptyset}/R \models s \approx t$ if and only if $s \leftrightarrow_R^* t$. Moreover, $\mathcal{T}_{\Sigma\downarrow}^{\emptyset}/R$ is term-generated—that is, for every element a of a universe of this interpretation, there exists a ground term t such that $[\![t]\!]^{\xi}_{\mathcal{T}_{\Sigma\downarrow}^{\emptyset}/R} = a$. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_{\Sigma\downarrow}^{\emptyset}/R$.

The following properties of the candidate interpretations can be shown exactly as in Waldmann's version of the first-order proof [66].

Lemma 4.1. *The rewrite systems R_C and R_∞ are confluent and terminating.*

Lemma 4.2. *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 4.3. *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 [66], we proceed by lifting inferences from the ground to the nonground level.

It is essential to the lifting lemmas that the selected literals of a clause correspond 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 $D \in N$ for which the selections $S(D)$ and $S_N(C)$ coincide. In the remainder of the proof, we adhere to the following convention:

Convention 4.4. When we refer to selected literals of clauses in N, it is with respect to the selection function S. When we refer to 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 4.5. *Let σ be the most general unifier of s and s' (which can be assumed idempotent [65, 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 4.6 (Lifting of non-SUP inferences)**.** *Let $C\theta \in \mathcal{G}_\Sigma(N)$, where $C \in N$ and θ is a substitution. (As selection functions, we use S for C and S_N for $C\theta$ as mentioned in Convention 4.4.) Then every* EQRES *or* EQFACT *inference from $C\theta$ is a ground instance of an inference from C up to purification.*

Proof. EQRES: We assume that there is an 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 eligible in $C\theta$. 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 eligible in $C\theta$, $s \not\approx s'$ is eligible C. Hence we have the inference

$$\frac{C' \vee s \not\approx s'}{C'\sigma \vee C_\mathrm{p}}\,\text{EQRES}$$

where $\sigma = \mathrm{mgu}(s, s')$ and C_p are purification literals. By Lemma 4.5, we have $C'\sigma\theta = C'\theta$. Thus, the ground inference is the θ-ground instance of the nonground inference up to purification.

EQFACT: We assume that there is an 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 eligible in $C\theta$, $s\theta \not\prec t\theta$, and $C = C' \vee s' \approx t' \vee s \approx t$. Then the ground inference is

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

where $s\theta$ and $s'\theta$ are unifiable and ground; hence $s\theta = s'\theta$. Since $s\theta \approx t\theta$ is eligible in $C\theta$ and $s\theta \not\prec t\theta$, $s \approx t$ is eligible in C and $s \not\prec t$. Hence we have the inference

$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma \vee C_\mathrm{p}}\,\text{EQFACT}$$

where $\sigma = \mathrm{mgu}(s, s')$ and C_p are purification literals. By Lemma 4.5, 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. □

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 below a variable. This condition is replaced by the following criterion.

Definition 4.7. We call a ground SUP inference from $D\theta$ and $C\theta$ *liftable* if the superposed subterm in $C\theta$ is not below a variable in C and the corresponding variable condition holds for D and C.

Lemma 4.8 (Lifting of SUP inferences)**.** *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$ where $D, C \in N$ and θ is a substitution. 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 eligible in $D\theta$. For positive superposition, $s\theta \approx s'\theta$ is strictly eligible in $C\theta$. For negative

superposition, $s\theta \not\approx s'\theta$ is eligible 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 eligible in $D\theta$ implies that $t \approx t'$ is strictly eligible in D. If $[\neg]\, s\theta \approx s'\theta$ is (strictly) eligible in $C\theta$, then $[\neg]\, s \approx s'$ is (strictly) eligible in C. Moreover, $D\theta \not\succeq C\theta$ 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 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. Thus we have a nonground inference

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

where $\sigma = \mathrm{mgu}(t, u)$ and C_{p} are purification literals. By Lemma 4.5, we have $(D' \vee C' \vee [\neg]\, s\langle t'\rangle_p \approx s')\sigma\theta = D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_p \approx s'\theta$. Therefore, the ground inference is the θ-ground instance of the nonground inference up to purification. □

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

In the following, we first prove those parts of the induction that crucially differ from the first-order proof, stated as separate lemmas. The justification for Lemma 4.9, about liftable inferences, is essentially as in the first-order case, but we need to argue why purification literals do not matter. The proof of Lemma 4.11, 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, the PosExt rule, and the ArgCong rule come into play. Lemma 4.10 explains how the the ArgCong rule can be used to justify the addition of more than one argument, which is nontrivial because deletion of redundant clauses might interrupt a chain of ArgCong applications if an intermediate clause is redundant.

Lemma 4.9. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$ be nonredundant with respect to $\mathcal{G}_\Sigma(N)$. 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. 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 stated by the lifting lemmas (Lemmas 4.6 and 4.8), this inference is an instance of an inference from C (or from D and C for Sup inferences) up to purification. 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 up to purification is redundant with respect to $\mathcal{G}_\Sigma(N)$.

By definition of inference redundancy, since neither $D\theta$ nor $C\theta$ is 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 4.5, 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 4.10. *Let $C\theta \in \mathcal{G}_\Sigma(N)$. Let t and t' be ground terms such that $R_{\lfloor C\theta \rfloor} \models \lfloor t \approx t' \rfloor$. Let $\bar{u}$ be a tuple of ground terms such that $t\ \bar{u}$ and $t'\ \bar{u}$ are smaller than the maximal term in $C\theta$. We assume that all clauses in $\mathcal{G}_\Sigma(N)$ smaller than $C\theta$ are true in $R_{\lfloor C\theta \rfloor}$. Moreover, the productive clauses of $R_{\lfloor C\theta \rfloor}$ contain no selected literals and are not redundant. Then $R_{\lfloor C\theta \rfloor} \models \lfloor t\ \bar{u} \approx t'\ \bar{u} \rfloor$.*

Proof. By induction on the length of $\bar{u}$. If $\bar{u}$ is empty, the claim is trivial. As the induction step, we want to show that $R_{\lfloor C\theta \rfloor} \models \lfloor t\ \bar{u} \approx t'\ \bar{u} \rfloor$ implies $R_{\lfloor C\theta \rfloor} \models \lfloor t\ \bar{u}\ u \approx t'\ \bar{u}\ u \rfloor$. This is equivalent to

$$\lfloor t\ \bar{u} \rfloor \downarrow_{R_{\lfloor C\theta \rfloor}} \lfloor t'\ \bar{u} \rfloor \text{ implies } \lfloor t\ \bar{u}\ u \rfloor \leftrightarrow^*_{R_{\lfloor C\theta \rfloor}} \lfloor t'\ \bar{u}\ u \rfloor$$

where $\downarrow_{R_{\lfloor C\theta \rfloor}}$ denotes that the two terms have the same normal form. For every rewrite step rewriting a proper subterm, there is obviously an analogous rewrite step if the term u is appended at the top level. Therefore, we can focus on the rewrite steps at the top level and hence it suffices to prove that

$$\mathsf{f}^{\bar{\tau}}_m(\bar{v}) \to \mathsf{g}^{\bar{\upsilon}}_n(\bar{w}) \in R_{\lfloor C\theta \rfloor} \text{ implies } \mathsf{f}^{\bar{\tau}}_{m+1}(\bar{v}, \lfloor u \rfloor) \leftrightarrow^*_{R_{\lfloor C\theta \rfloor}} \mathsf{g}^{\bar{\upsilon}}_{n+1}(\bar{w}, \lfloor u \rfloor)$$

for all function symbols f, g and all $m, n, \bar{\tau}, \bar{\upsilon}, \bar{v}, \bar{w}$. Since $\lfloor t\ \bar{u}\ u \rfloor$ and $\lfloor t'\ \bar{u}\ u \rfloor$ are smaller than the maximal term of $C\theta$, we may moreover assume that $\mathsf{f}^{\bar{\tau}}_{m+1}(\bar{v}, \lfloor u \rfloor)$ is smaller than the maximal term of $C\theta$.

Since $\mathsf{f}^{\bar{\tau}}_m(\bar{v}) \to \mathsf{g}^{\bar{\upsilon}}_n(\bar{w}) \in R_{\lfloor C\theta \rfloor}$, it must come from a productive clause of the form $\lfloor D\theta \rfloor = \lfloor D'\theta \rfloor \vee \mathsf{f}^{\bar{\tau}}_m(\bar{v}) \approx \mathsf{g}^{\bar{\upsilon}}_n(\bar{w})$, originating from a clause $D = D' \vee s \approx s'$ in N. By this lemma's assumption on productive literals, $\mathsf{f}^{\bar{\tau}}_m(\bar{v}) \approx \mathsf{g}^{\bar{\upsilon}}_n(\bar{w})$ must be strictly maximal in $\lfloor D\theta \rfloor$. By Convention 4.4, the corresponding literal $s \approx s'$ in D must also be strictly maximal. Hence we have the inference

$$\frac{D}{D' \vee s\ x \approx s'\ x \vee D_\mathrm{p}}\ \textsc{ArgCong}$$

where D_p are purification literals (or $\bot$).

By this lemma's assumption on productive clauses, $D\theta$ is not redundant and therefore D is not redundant. Hence, the conclusion of the above inference is in $N \cup \mathrm{Red}(N)$. Let θ' be an extension of θ that copies the corresponding values of θ to the freshly introduced purification variables and that maps x to $\lfloor u \rfloor$. The θ'-ground instance $\lfloor D'\theta \rfloor \vee \mathsf{f}^{\bar{\tau}}_m(\bar{v}, \lfloor u \rfloor) \approx \mathsf{g}^{\bar{\upsilon}}_n(\bar{w}, \lfloor u \rfloor) \vee \lfloor C_\mathrm{p}\theta' \rfloor$ of the conclusion of the above inference 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 $R_{\lfloor C\theta \rfloor}$, because $\mathsf{f}^{\bar{\tau}}_{m+1}(\bar{v}, \lfloor u \rfloor)$ is smaller than the maximal term of $C\theta$ and hence this θ'-ground instance is smaller than $C\theta$. By Lemma 4.3, $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. The instances of the purification literals $D_P\theta'$ are trivially false because θ' duplicates the value of corresponding variables. So $\mathsf{f}^{\bar{\tau}}_m(\bar{v}, \lfloor u \rfloor) \approx \mathsf{g}^{\bar{\upsilon}}_n(\bar{w}, \lfloor u \rfloor)$ must be true in $R_{\lfloor C\theta \rfloor}$ and hence $\mathsf{f}^{\bar{\tau}}_m(\bar{v}, \lfloor u \rfloor) \leftrightarrow^*_{R_{\lfloor C\theta \rfloor}} \mathsf{g}^{\bar{\upsilon}}_n(\bar{w}, \lfloor u \rfloor)$. □

Lemma 4.11. *Let $D\theta, C\theta \in \mathcal{G}_\Sigma(N)$ be nonredundant with respect to $\mathcal{G}_\Sigma(N)$. We consider a nonliftable* Sup *inference from $D\theta$ and $C\theta$. Let $D'\theta$ be the clause $D\theta$ without the literal involved in the inference. We assume that all clauses in $\mathcal{G}_\Sigma(N)$ smaller than $C\theta$ are true and that $\lfloor D'\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. Moreover, we assume that the productive clauses of $R_{\lfloor C\theta \rfloor}$ contain no selected literals and are nonredundant with respect to $\mathcal{G}_\Sigma(N)$. Then $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.*

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. Since $R_{\lfloor C\theta \rfloor} \not\models \lfloor D'\theta \rfloor$, we have $R_{\lfloor C\theta \rfloor} \models \lfloor t\theta \approx t'\theta \rfloor$.

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 a proper 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. Since $R_{\lfloor C\theta \rfloor} \models \lfloor t\theta \approx t'\theta \rfloor$, by congruence, $R_{\lfloor C\theta \rfloor} \models \lfloor x\theta\ \bar{w} \approx v\ \bar{w} \rfloor$ for any arguments $\bar{w}$. Hence, $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$ if and only if $R_{\lfloor C\theta \rfloor} \models \lfloor C\{x \mapsto v\}\theta \rfloor$. 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 $R_{\lfloor C\theta \rfloor}$, we have $R_{\lfloor C\theta \rfloor} \models \lfloor C\{x \mapsto v\}\theta \rfloor$ and hence $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

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 3.1, in all variants, u, t, and t' have the following form: $u = x\ \bar{v}_n$ for some variable x and a tuple of terms $\bar{v}_n$ of length $n \geq 0$; $t = \tilde{t}\ \bar{x}_n$ 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 $\bar{v}_n$ 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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$ by congruence, because $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta' \rfloor$ and $R_{\lfloor C\theta \rfloor} \models \lfloor t\theta \approx t'\theta \rfloor$.

Now we assume that x occurs with arguments other than $\bar{v}_n$ 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 $\bar{v}_n \neq \varepsilon$ 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 $\bar{v}_n$. Hence these literals are equal or smaller in $C\theta''$ than in $C\theta$, because $x\theta''\ \bar{v}_n \prec x\theta\ \bar{v}_n$ 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 $C\theta \succ C\theta''$.

We need to show that $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$. There is a PosExt inference from D to $D' \vee \tilde{t} \approx \tilde{t}'$. 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, $R_{\lfloor C\theta \rfloor} \models \lfloor (D' \vee \tilde{t} \approx \tilde{t}')\theta \rfloor$ because this clause is smaller than $C\theta$. Since $D'\theta$ is false in $R_{\lfloor C\theta \rfloor}$, we have $R_{\lfloor C\theta \rfloor} \models \lfloor \tilde{t}\theta \approx \tilde{t}'\theta \rfloor$. 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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta'' \rfloor$.

We distinguish two cases. If for every literal of the form $x \not\approx y$, where $y\theta = s\theta$, we have $R_{\lfloor C\theta \rfloor} \models \lfloor y\theta'' \approx y\theta \rfloor$, then $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$ by congruence. If for some literal of the form $x \not\approx y$, where $y\theta = s\theta$, we have $R_{\lfloor C\theta \rfloor} \models \lfloor y\theta'' \not\approx y\theta \rfloor$, then $R_{\lfloor C\theta \rfloor} \models \lfloor y\theta \not\approx x\theta \rfloor$ because $y\theta'' = \tilde{t}'\theta$, $R_{\lfloor C\theta \rfloor} \models \lfloor \tilde{t}'\theta \approx \tilde{t}\theta \rfloor$, and $\tilde{t}\theta = x\theta$. Hence a literal of $C\theta$ is true in $R_{\lfloor C\theta \rfloor}$ and therefore $C\theta$ is true in $R_{\lfloor C\theta \rfloor}$.

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 our assumption on $R_{\lfloor C\theta \rfloor}$, $C\theta \succ C''\theta$ implies $R_{\lfloor C\theta \rfloor} \models \lfloor C''\theta \rfloor$. We will use equalities that are true in $R_{\lfloor C\theta \rfloor}$ to rewrite $\lfloor C\theta \rfloor$ into $\lfloor C''\theta \rfloor$, which implies $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$ by congruence.

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

$$R_{\lfloor C\theta \rfloor} \models \lfloor \tilde{t}\theta\ \bar{u} \rfloor \approx \lfloor \tilde{t}'\theta\ \bar{u} \rfloor \tag{\dagger}$$

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 three cases:

If $\bar{u}$ has length n, then $D'\theta \vee \tilde{t}\theta\ \bar{u} \approx \tilde{t}'\theta\ \bar{u}$ is a ground instance of D and hence in $\mathcal{G}_\Sigma(N)$. Since $\tilde{t}\theta\ \bar{u}$ and $\tilde{t}'\theta\ \bar{u}$ are smaller than the maximal term of $C\theta$, this ground instance must be true in $R_{\lfloor C\theta \rfloor}$ and hence (†) because $R_{\lfloor C\theta \rfloor} \not\models \lfloor D'\theta \rfloor$.

If $\bar{u}$ is shorter than length n, there is a PosExt inference from D to a clause with ground instance $D'\theta \vee \tilde{t}\theta\ \bar{u} \approx \tilde{t}'\theta\ \bar{u}$. By saturation up to redundancy, this ground instance is in $\mathcal{G}_\Sigma(N)$ or redundant with respect to $\mathcal{G}_\Sigma(N)$. Again, since $\tilde{t}\theta\ \bar{u}$ and $\tilde{t}'\theta\ \bar{u}$ are smaller than the maximal term of $C\theta$, this ground instance must be true in $R_{\lfloor C\theta \rfloor}$ and hence (†) because $R_{\lfloor C\theta \rfloor} \not\models \lfloor D'\theta \rfloor$.

If $\bar{u}$ is longer than length n, we apply Lemma 4.10 to the terms $\tilde{t}$ and $\tilde{t}'$, using $R_{\lfloor C\theta \rfloor} \models \lfloor \tilde{t}\theta \approx \tilde{t}'\theta \rfloor$, which we have just proved. This lemma lets us derive (†).

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, so 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 $R_{\lfloor C\theta \rfloor}$ and that $\tilde{C}_*$ and $\lfloor C''\theta \rfloor$ are equivalent in $R_{\lfloor C\theta \rfloor}$ 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 a 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 a deepest position. 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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

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, so 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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$ by congruence. □

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

Lemma 4.12 (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. Part (iii) follows from (i) by Lemmas 4.2 and 4.3. 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 $R_{\lfloor C\theta \rfloor} \models \lfloor C\theta \rfloor$.

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 4.9, $\lfloor C'\theta \rfloor$ 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 $\lfloor s\theta \rfloor \downarrow_{R_{\lfloor C\theta \rfloor}} \lfloor s'\theta \rfloor$, 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 4.11, $\neg\lfloor D'\theta \rfloor$ 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 4.3. 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 4.9, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. It follows from part (iii) of the induction hypothesis that $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \not\approx s'\theta \rfloor$ 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 4.9, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. It follows from part (iii) of the induction hypothesis that 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 $R_{\lfloor C\theta \rfloor}$. Then there must be a rule $\lfloor t\theta \rfloor \to \lfloor t'\theta \rfloor \in R_{\lfloor C\theta \rfloor}$ that reduces $\lfloor s\theta \rfloor$. This rule must stem from a productive clause $\lfloor D\theta \rfloor = \lfloor D'\theta \vee t\theta \approx t'\theta \rfloor$.

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 4.11, $\neg\lfloor D'\theta \rfloor$ 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 4.3. 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 4.9, $\lfloor D'\theta \vee C'\theta \vee s\theta\langle t'\theta \rangle \approx s'\theta \rfloor$ 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 $R_{\lfloor C\theta \rfloor}$. By assumption, $C\theta$ is not redundant and we have $E_{\lfloor C\theta \rfloor} = \emptyset$. We assume that $\lfloor C\theta \rfloor$ is false in $R_{\lfloor C\theta \rfloor}$. 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 4.9, its conclusion is entailed by clauses from $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ that are smaller than $\lfloor C\theta \rfloor$. It follows from part (iii) of the induction hypothesis that 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 enabling us to do this construction are that R_∞ is term-generated and that the interpretations of the members $\mathsf{f}_k^{\bar{\tau}_m}, \ldots, \mathsf{f}_n^{\bar{\tau}_m}$ of a same symbol family behave in the same way.

Lemma 4.13 (Argument congruence)**.** *For all floor-logic ground terms $\mathsf{f}_m^{\bar{\tau}}(\bar{s})$, $\mathsf{g}_n^{\bar{\tau}'}(\bar{t})$, and u, if $[\![\mathsf{f}_m^{\bar{\tau}}(\bar{s})]\!]_{R_\infty} = [\![\mathsf{g}_n^{\bar{\tau}'}(\bar{t})]\!]_{R_\infty}$, then $[\![\mathsf{f}_{m+1}^{\bar{\tau}}(\bar{s}, u)]\!]_{R_\infty} = [\![\mathsf{g}_{n+1}^{\bar{\tau}'}(\bar{t}, u)]\!]_{R_\infty}$.*

Proof. The proof of this lemma is analogous to the proof of Lemma 4.10, but for R_∞. What we want to show is equivalent to

$$R_\infty \models \mathsf{f}_m^{\bar{\tau}}(\bar{s}) \approx \mathsf{g}_n^{\bar{\tau}'}(\bar{t}) \;\text{ implies }\; R_\infty \models \mathsf{f}_{m+1}^{\bar{\tau}}(\bar{s}, u) \approx \mathsf{g}_{n+1}^{\bar{\tau}'}(\bar{t}, u)$$

which is equivalent to

$$\mathsf{f}^{\bar\tau}_m(\bar s)\downarrow_{R_\infty}\mathsf{g}^{\bar\tau'}_n(\bar t)\ \text{ implies }\ \mathsf{f}^{\bar\tau}_{m+1}(\bar s,u)\leftrightarrow^*_{R_\infty}\mathsf{g}^{\bar\tau'}_{n+1}(\bar t,u)$$

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

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

for all function symbols h,h' and all $k,k',\bar\upsilon,\bar\upsilon'$.

Since $\mathsf{h}^{\bar\upsilon}_k(\bar v)\to\mathsf{h}'^{\bar\upsilon'}_{k'}(\bar v')\in R_\infty$, it must come from a productive clause of the form $\lfloor C\theta\rfloor=\lfloor C'\theta\rfloor\vee\mathsf{h}^{\bar\upsilon}_k(\bar v)\approx\mathsf{h}'^{\bar\upsilon'}_{k'}(\bar v')$, originating from a clause $C=C'\vee w\approx w'$ in N. By Lemma 4.12 (iv) and condition (b) of the construction of R_C, the literal $\mathsf{h}^{\bar\upsilon}_k(\bar v)\approx\mathsf{h}'^{\bar\upsilon'}_{k'}(\bar v')$ must be strictly eligible. By Convention 4.4, the corresponding literal $w\approx w'$ in C must also be strictly eligible. Hence we have the inference

$$\frac{C}{C'\vee w\ x\approx w'\ x\vee C_{\mathrm{p}}}\ \textsc{ArgCong}$$

where C_{p} are purification literals (or $\bot$).

By part (ii) of Lemma 4.12, a productive clause is never redundant; hence $C\theta$ is not redundant and therefore C is not redundant. Hence, the conclusion of the above inference is in $N\cup\mathrm{Red}(N)$. Let θ' be an extension of θ that copies the corresponding values of θ to the freshly introduced purification variables and that maps x to u. The θ'-ground instance $\lfloor C'\theta\rfloor\vee\mathsf{h}^{\bar\upsilon}_k(\bar v,u)\approx\mathsf{h}'^{\bar\upsilon'}_{k'}(\bar v',u)\vee\lfloor C_{\mathrm{p}}\theta'\rfloor$ of the conclusion of the above inference 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 4.3, $\lfloor C'\theta\rfloor$ is false in R_∞. The instances of the purification literals $C_{\mathrm{p}}\theta'$ are trivially false because θ' duplicates the value of corresponding variables. So $\mathsf{h}^{\bar\upsilon}_k(\bar v,u)\approx\mathsf{h}'^{\bar\upsilon'}_{k'}(\bar v',u)$ must be true in R_∞ and hence $\mathsf{h}^{\bar\upsilon}_{k+1}(\bar v,u)\leftrightarrow^*_{R_\infty}\mathsf{h}'^{\bar\upsilon'}_{k'+1}(\bar v',u)$. □

Definition 4.14. Define an interpretation $R^\uparrow_\infty=(\mathcal{U}^\uparrow,\mathcal{J}^\uparrow_{\mathsf{ty}},\mathcal{J}^\uparrow,\mathcal{E}^\uparrow)$ in the ceiling logic as follows. The interpretation R_∞ defined above is an interpretation in monomorphic first-order logic. Let U_τ be its universe for type τ and and $\mathcal{J}$ its interpretation function. Let $\mathcal{U}^\uparrow=\{U_\tau\mid\tau\text{ is a ground type}\}$. Let $\mathcal{J}^\uparrow_{\mathsf{ty}}(\kappa)(U_{\bar\tau})=U_{\kappa(\bar\tau)}$ for all type constructors κ and type tuples $\bar\tau$. Let $\mathcal{J}^\uparrow(\mathsf{f},U_{\bar\tau})=\mathcal{J}(\mathsf{f}^{\bar\tau}_k)$, where k is the number of mandatory arguments of f. Since R_∞ is term-generated, for every $a\in U_{\tau\to\upsilon}$, there exists a ground term $s:\tau\to\upsilon$ such that $[\![\lfloor s\rfloor]\!]_{R_\infty}=a$. Without loss of generality, we write $s=\mathsf{f}\langle\bar\upsilon_m\rangle(\bar s_k)\ s_{k+1}\ \ldots\ s_m$. Then we have $a=[\![\mathsf{f}^{\bar\upsilon_m}_m(\lfloor\bar s_m\rfloor)]\!]_{R_\infty}$ and define $\mathcal{E}^\uparrow$ by

$$\mathcal{E}^\uparrow_{U_\tau,U_\upsilon}(a)(b)=\mathcal{J}(\mathsf{f}^{\bar\upsilon_m}_{m+1})([\![\lfloor\bar s_m\rfloor]\!]_{R_\infty},b)\quad\text{for all } b\in U_\tau$$

It follows that $\mathcal{E}^\uparrow_{U_\tau,U_\upsilon}(a)\big([\![u]\!]_{R_\infty}\big)=[\![\mathsf{f}^{\bar\upsilon_m}_{m+1}(\lfloor\bar s_m\rfloor,u)]\!]_{R_\infty}$ for any term u. This interpretation is well defined if the definition of $\mathcal{E}^\uparrow$ does not depend on the choice of the ground term s. To show this, we assume that there exists another ground term $t=\mathsf{g}\langle\bar\upsilon'\rangle(\bar t_l)\ t_{l+1}\ \ldots\ t_n$ such that $[\![\lfloor t\rfloor]\!]_{R_\infty}=a$. By Lemma 4.13, it follows from $[\![\lfloor s\rfloor]\!]_{R_\infty}=[\![\lfloor t\rfloor]\!]_{R_\infty}$ that

$$[\![\mathsf{f}^{\bar\upsilon}_{m+1}(\lfloor\bar s_m\rfloor,u)]\!]_{R_\infty}=[\![\mathsf{g}^{\bar\upsilon'}_{n+1}(\lfloor\bar t_n\rfloor,u)]\!]_{R_\infty}$$

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

Since R_∞ is a term-generated model of $\lfloor\mathcal{G}_\Sigma(N)\rfloor$, we can show that $R^\uparrow_\infty$ is a term-generated model of $\mathcal{G}_\Sigma(N)$ (Lemma 4.15). And using the same argument as in the first-order

proof, we can lift this result to nonground clauses (Lemma 4.16). For the extensional variants, we also need to show that $R_\infty^\uparrow$ is an extensional interpretation (Lemma 4.17).

Lemma 4.15 (Model transfer to ceiling logic)**.** *$R_\infty^\uparrow$ is a type- and term-generated model of $\mathcal{G}_\Sigma(N)$.*

Proof. We first prove by induction on ground terms t of the ceiling logic that $[\![t]\!]_{R_\infty^\uparrow} = [\![\lfloor t \rfloor]\!]_{R_\infty}$. Let t be a ground ceiling term, and assume as the induction hypothesis that $[\![u]\!]_{R_\infty^\uparrow} = [\![\lfloor u \rfloor]\!]_{R_\infty}$ for all subterms u of t. If t is of the form $\mathsf{f}\langle\bar{\upsilon}\rangle(\bar{t}_k)$, then

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

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

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

So we have shown that $[\![t]\!]_{R_\infty^\uparrow} = [\![\lfloor t \rfloor]\!]_{R_\infty}$ for all terms t. It follows that a ground equation $s \approx t$ is true in $R_\infty^\uparrow$ if and only if $\lfloor s \approx t \rfloor$ is true in R_∞. Hence a ground clause C is true in $R_\infty^\uparrow$ if and only if $\lfloor C \rfloor$ is true in R_∞. By Lemma 4.12, R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$. Thus, $R_\infty^\uparrow$ is a model of $\mathcal{G}_\Sigma(N)$.

It remains to show that $R_\infty^\uparrow$ is type- and term-generated. All universes in $\mathcal{U}^\uparrow$ are of the form U_τ for some ground term τ. Since $[\![\tau]\!]_{\mathcal{I}^\uparrow} = U_\tau$, the model $R_\infty^\uparrow$ is type-generated.

To show that it is also term-generated, let $U_\tau \in \mathcal{U}^\uparrow$ be a universe and $a \in U_\tau$. Since R_∞ is term-generated, we have a ground term t of the floor logic with $[\![t]\!]_{R_\infty}^\xi = a$. Using what we showed above, we have $[\![\lceil t \rceil]\!]_{R_\infty^\uparrow}^\xi = [\![t]\!]_{R_\infty}^\xi = a$. Hence, $R_\infty^\uparrow$ is term-generated. □

Lemma 4.16 (Model transfer to nonground clauses)**.** *$R_\infty^\uparrow$ is a model of N.*

Proof. Let $C \in N$. By the definition of clause semantics, $R_\infty^\uparrow \models C$ holds if and only if C is true in $R_\infty^\uparrow$ for all valuations ξ.

Let ξ be a valuation. We define a substitution θ as follows. For every type variable α occurring in C, we choose a ground type τ such that $[\![\tau]\!]_{\mathcal{I}_{\mathsf{ty}}^\uparrow} = \xi(\alpha)$ and define $\alpha\theta = \tau$. For every variable x occurring in C, we choose a ground term t such that $[\![t]\!]_{R_\infty^\uparrow} = \xi(x)$ and define $x\theta = t$. Such types and terms exist because $R_\infty^\uparrow$ is type- and term-generated. By the substitution lemma [65, Lemma 3.1], $C\theta$ is true in $R_\infty^\uparrow$ if and only if C is true in $R_\infty^\uparrow$ for ξ. Thus, since $C\theta \in \mathcal{G}_\Sigma(N)$ and $R_\infty^\uparrow \models \mathcal{G}_\Sigma(N)$ by Lemma 4.15, C is true in $R_\infty^\uparrow$ for ξ.

Hence, we have shown that $R_\infty^\uparrow \models C$ for all $C \in N$. □

Lemma 4.17 (Completeness of the extensionality axioms)**.** *If N contains the extensionality axiom, $R_\infty^\uparrow$ is extensional.*

Proof. Assume that the clause set N contains the extensionality axiom. By Lemma 4.16, the extensionality axiom is hence true in $R_\infty^\uparrow$.

Assume that $R_\infty^\uparrow$ is not extensional. Then $\mathcal{E}^\uparrow$ is not injective, i.e., there are $a \neq b \in U_{\tau\to\upsilon}$ for some τ and some υ, such that $\mathcal{E}^\uparrow(a) = \mathcal{E}^\uparrow(b)$. Let $\xi = \{\alpha \mapsto U_\tau, \beta \mapsto U_\upsilon, x \mapsto a, y \mapsto b\}$. Then the extensionality axiom

$$x\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \not\approx y\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y)) \vee x \approx y$$

is false in $R_\infty^\uparrow$ for ξ because

$$\begin{aligned}
[\![x\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y))]\!]^\xi_{R_\infty^\uparrow} &= \mathcal{E}^\uparrow(a)([\![\mathsf{diff}\langle\alpha,\beta\rangle(x,y)]\!]^\xi_{R_\infty^\uparrow}) \\
&= \mathcal{E}^\uparrow(b)([\![\mathsf{diff}\langle\alpha,\beta\rangle(x,y)]\!]^\xi_{R_\infty^\uparrow}) \\
&= [\![y\ (\mathsf{diff}\langle\alpha,\beta\rangle(x,y))]\!]^\xi_{R_\infty^\uparrow}
\end{aligned}$$

but this is impossible since the extensionality axioms are true in $R_\infty^\uparrow$. □

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

Theorem 4.18 (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 axiom.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ is a model of $\lfloor \mathcal{G}_\Sigma(N) \rfloor$ according to part (iii) of Lemma 4.12. By Lemma 4.15, $R_\infty^\uparrow$ is a type- and term-generated model of $\mathcal{G}_\Sigma(N)$. By Lemma 4.16, it is a model of N. If N contains the extensionality axioms, then $R_\infty^\uparrow$ is even an extensional model by Lemma 4.17. □

Refutational completeness of superposition can also be viewed dynamically, as a series of steps that derive new clauses or delete redundant clauses. Under the assumption that no inference is delayed indefinitely, one can show that starting from an unsatisfiable set of clauses, eventually the empty clause will be derived. This can be shown for our calculus exactly as in Waldmann's first-order proof [66].

5. Implementation

Zipperposition [26, 27] 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 [17]. 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 [61]. We have now also implemented a complete λ-free higher-order mode based on the four calculi described in this article.

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. One of the bugs we found during our implementation work occurred because argument positions shift when

[1] `https://github.com/sneeuwballen/zipperposition`

instantiating 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 number in the unification algorithm. To satisfy the requirements on selection, we avoid selecting literals that contain higher-order variables. To comply with our redundancy notion, we disabled rewriting of non-argument subterms. Finally, to improve term indexing of higher-order terms, we replaced the imperfect discrimination trees by fingerprint indices [55].

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 term-order-specific algorithm to determine whether there might exist a grounding 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 add the extensionality axiom to the clause set. 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 the NegExt rule described in Section 3.2, which resembles Vampire's extensionality resolution rule [35].

Using Zipperposition, we can quantify the disadvantage of the applicative encoding on the problem given at the end of Section 3.2. A well chosen KBO instance with argument coefficients allows 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 0.6 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 and Sledgehammer-generated Isabelle/HOL benchmarks. We compare our calculi with an applicative encoding mode, which performs the applicative encoding after the clausal normal form transformation and then proceeds with Zipperposition's first-order mode. The encoding makes all function symbols nullary and replaces all applications with a polymorphic binary `app` symbol.

Our experimental data is available online.[2] We used the development version of Zipperposition, revision `8bde3f6b`.[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. Many of the higher-order problems in the TPTP library are in fact

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

[3] `https://github.com/sneeuwballen/zipperposition/tree/8bde3f6b`

		# sat			# unsat			⌀ time		
		LPO	KBO	EPO	LPO	KBO	EPO	LPO	KBO	EPO
TFF	first-order mode	0	0	0	**209**	238	91	**1.5**	3.7	0.3
	applicative encoding	0	0	0	158	199	19	19.0	18.4	35.0
	nonpurifying calculus	0	0	0	207	237	92	1.7	3.8	0.3
	purifying calculus	0	0	0	206	233	92	1.8	6.0	0.3
SH256	applicative encoding	**1**	1	1	483	505	40	20.1	18.1	70.7
	nonpurifying calculus	**1**	1	1	**552**	596	493	**4.7**	3.2	0.1
	purifying calculus	**1**	1	1	541	560	487	7.1	4.9	0.1
SH16	applicative encoding	**240**	232	110	**392**	391	152	2.5	1.4	13.6
	nonpurifying calculus	193	252	180	383	392	374	**0.6**	1.1	0.0
	purifying calculus	112	142	112	370	371	365	2.6	3.2	0.0
THF	applicative encoding	**138**	122	120	**472**	476	393	1.6	0.9	2.5
	nonpurifying calculus	119	121	120	**472**	475	458	**0.6**	0.3	0.0
	purifying calculus	116	117	116	471	473	458	0.7	0.7	0.0

FIGURE 1. Evaluation of the intensional calculi

satisfiable for our λ-free logic, even though they may be unsatisfiable for full higher-order logic and labeled as such in the TPTP.

We instantiated all calculus variants with three different term orders. The most relevant order is LPO [18], because of its nonmonotonicity, which our calculi are designed to cope with. To evaluate the cost of nonmonotonicity, we also evaluated the calculi using the monotonic orders KBO [6] (without argument coefficients) and EPO [8]. However, when using a monotonic order, it may be more efficient to superpose at non-argument subterms directly instead of relying on the ArgCong rule.

From the TPTP, we collected 709 first-order problems in TFF format and 665 higher-order problems in THF format, both groups containing both monomorphic and polymorphic problems. We excluded all problems that contain arithmetic, tuples, the `$distinct` predicate, or the `$ite` symbol, as well as problems whose clausal normal form falls outside the lambda-free fragment described in Section 2.

The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite [21], were regenerated to target λ-free higher-order logic. They comprise 2506 problems in total, divided in two groups based on the number of Isabelle facts (lemmas, definitions, etc.) selected for inclusion in each problem: either 256 (SH256) or 16 facts (SH16). Each group was generated by encoding λ-expressions as λ-lifted supercombinators [45].

Figure 1 summarizes, for the intensional calculi, the number of solved satisfiable and unsatisfiable problems within 300 s, and the time taken to show unsatisfiability. Figure 2 presents the corresponding data for the extensional calculi. The average time is computed over the problems that all configurations for the respective benchmark set and term order found to be unsatisfiable within the time limit. The best result for LPO in each benchmark set is highlighted in bold. The evaluation was carried out on StarExec [60] using Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz.

		# sat			# unsat			⌀ time		
		LPO	KBO	EPO	LPO	KBO	EPO	LPO	KBO	EPO
TFF	first-order mode	0	0	0	**209**	238	91	**0.9**	3.4	0.3
	applicative encoding	0	0	0	156	193	18	20.6	15.1	36.7
	nonpurifying calculus	0	0	0	205	233	88	1.8	4.7	0.4
	purifying calculus	0	0	0	198	230	90	3.6	7.4	0.4
SH256	applicative encoding	**1**	1	1	482	502	40	19.7	16.1	70.6
	nonpurifying calculus	**1**	1	1	**558**	603	502	**3.9**	2.5	0.1
	purifying calculus	0	1	0	534	553	485	4.5	5.4	0.1
SH16	applicative encoding	**221**	214	95	**393**	394	148	2.3	2.6	10.5
	nonpurifying calculus	174	233	166	390	400	376	**0.8**	0.4	0.0
	purifying calculus	69	90	70	371	373	362	2.6	2.7	0.0
THF	applicative encoding	**120**	118	117	**476**	478	395	1.4	0.9	2.8
	nonpurifying calculus	113	117	116	471	472	455	0.5	0.4	0.0
	purifying calculus	99	104	100	468	467	457	**0.4**	1.2	0.0

FIGURE 2. Evaluation of the extensional calculi

The experimental results on the TFF part of the TPTP library confirm that our calculi handle the vast majority of problems that are solvable in first-order mode gracefully. On first-order problems, the calculi only rarely differ from the behavior of the first-order mode, due to the interaction of ArgCong with polymorphic types and due to the extensionality axiom. In contrast, the applicative encoding is comparatively inefficient on problems that are already first-order. For LPO, the success rate drops by about 25%, and the average time to show unsatisfiability is over ten times larger.

The SH256 benchmarks mostly consist of large, mildly higher-order problems—a practically relevant class of problems that is underrepresented in the TPTP library. On these problems, our calculi clearly outperform the applicative encoding. This is not surprising since the proving effort is dominated by first-order reasoning.

For the SH16 and THF benchmarks, the situation is less clear. The applicative encoding and our calculi are roughly neck-and-neck, but the applicative encoding finds much more saturations.

Across all benchmarks, the nonpurifying calculi outperform their purifying relatives. On first-order problems, where all term orders are monotonic, KBO outperforms LPO by a difference of roughly 15%. This margin is similar on the large higher-order benchmarks, where LPO is nonmonotonic, suggesting that there is no substantial cost associated with 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 [51] Isabelle formalization of the refutational completeness of a superposition calculus that operates on λ-free higher-order terms and that is parameterized by a monotonic term order. Extensions with polymorphism and induction, independently

developed by Cruanes [26,27] and Wand [67], contribute to increasing the power of automatic provers. Detection of inconsistencies in axioms, as suggested by Schulz et al. [57], is important for large axiomatizations.

Also of interest is Bofill and Rubio's [20] 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. However, coefficients can be useful 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 $(\lambda x.\ x + x)\ y \succ y + y$.

Many researchers have proposed or used encodings of higher-order logic constructs into first-order logic, including Robinson [53], Kerber [40], Dougherty [30], Dowek et al. [31], Hurd [39], Meng and Paulson [45], Obermeyer [50], and Czajka [28]. Encodings of types, such as those by Bobot and Paskevich [19] and Blanchette et al. [15], 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 [16].

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. [37] 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 [52], Andrews's [1], and Huet's [38] pioneering work stands out. Andrews [2] and Benzmüller and Miller [11] provide excellent surveys. The competitive higher-order automatic theorem provers include LEO-II [12] (based on RUE resolution), Satallax [23] (based on a tableau calculus and a SAT solver), AgsyHOL [44] (based on a focused sequent calculus and a generic narrowing engine), Leo-III [59] (based on a pragmatic higher-order version of ordered paramodulation with no completeness guarantees), CVC4 and veriT [5] (both based on satisfiability modulo theories), and Vampire [14] (based on superposition and SK-combinators).

The Isabelle proof assistant [49] (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 [61].

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 [56], SPASS [68], and Vampire [42], and integrate these in proof assistants to provide a high level of automation. With its stratified architecture, Otter-λ [7] is perhaps the closest to what we are aiming at, but it is limited to second-order logic and offers no completeness guarantees. As a first step, Vukmirović, Blanchette, and Schulz [64] have generalized E's data structures and algorithms to λ-free higher-order logic, assuming a monotonic KBO [6].

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. Especially on large problems, our experiments 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. Indeed, the subsequent work by Bentkamp et al. [9], which introduces support for λ-expressions, is largely based on our extensional nonpurifying calculus.

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] Barbosa, H., Reynolds, A., El Ouraoui, D., Tinelli, C., Barrett, C.: Extending SMT solvers to higher-order logic. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716. Springer (2019)
[6] 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)
[7] Beeson, M.: Lambda logic. In: Basin, D.A., Rusinowitch, M. (eds.) IJCAR 2004. LNCS, vol. 3097, pp. 460–474. Springer (2004)
[8] Bentkamp, A.: Formalization of the embedding path order for lambda-free higher-order terms. Archive of Formal Proofs (2018), `http://isa-afp.org/entries/Lambda_Free_EPO.html`
[9] Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716. Springer (2019)
[10] Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 28–46. Springer (2018)
[11] 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)
[12] 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)
[13] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)

[14] Bhayat, A., Reger, G.: Restricted combinatory unification. In: Fontaine, P. (ed.) CADE-27. LNCS, vol. 11716. Springer (2019)
[15] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. 12(4:13), 1–52 (2016)
[16] Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reas. 9(1), 101–148 (2016)
[17] 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)
[18] 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)
[19] 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)
[20] Bofill, M., Rubio, A.: Paramodulation with non-monotonic orderings and simplification. J. Autom. Reason. 50(1), 51–98 (2013)
[21] Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 107–121. Springer (2010)
[22] Brand, D.: Proving theorems with the modification method. SIAM J. Comput. 4, 412–430 (1975)
[23] 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)
[24] Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. 13(5), 639–688 (2003)
[25] Church, A.: A formulation of the simple theory of types. J. Symb. Log. 5(2), 56–68 (1940)
[26] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)
[27] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)
[28] 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)
[29] Digricoli, V.J., Harrison, M.C.: Equality-based binary resolution. J. ACM 33(2), 253–289 (1986)
[30] Dougherty, D.J.: Higher-order unification via combinators. Theor. Comput. Sci. 114(2), 273–298 (1993)
[31] Dowek, G., Hardin, T., Kirchner, C.: Higher-order unification via explicit substitutions (extended abstract). In: LICS '95. pp. 366–374. IEEE Computer Society (1995)
[32] 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)
[33] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)
[34] Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
[35] 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)
[36] Henkin, L.: Completeness in the theory of types. J. Symb. Log. 15(2), 81–91 (1950)
[37] Hirokawa, N., Middeldorp, A., Zankl, H.: Uncurrying for termination and complexity. J. Autom. Reasoning 50(3), 279–315 (2013)
[38] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)
[39] 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)
[40] 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)
[41] Kop, C.: Higher Order Termination: Automatable Techniques for Proving Termination of Higher-Order Term Rewriting Systems. Ph.D. thesis, Vrije Universiteit Amsterdam (2012)
[42] 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)
[43] 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)

[44] 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)
[45] Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. 40(1), 35–60 (2008)
[46] Miller, D.A.: A compact representation of proofs. Studia Logica 46(4), 347–370 (1987)
[47] Nieuwenhuis, R., Rubio, A.: Basic superposition is complete. In: Krieg-Brückner, B. (ed.) ESOP '92. LNCS, vol. 582, pp. 371–389. Springer (1992)
[48] 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)
[49] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[50] Obermeyer, F.H.: Automated Equational Reasoning in Nondeterministic λ-Calculi Modulo Theories $\mathcal{H}^*$. Ph.D. thesis, Carnegie Mellon University (2009)
[51] Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs (2016), `https://www.isa-afp.org/entries/SuperCalc.shtml`
[52] Robinson, J.: Mechanizing higher order logic. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)
[53] 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)
[54] Schmidt-Schauß, M.: Unification in a combination of arbitrary disjoint equational theories. J. Symb. Comput. 8, 51–99 (1989)
[55] Schulz, S.: Fingerprint indexing for paramodulation and rewriting. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 477–483. Springer (2012)
[56] 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)
[57] 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)
[58] Snyder, W., Lynch, C.: Goal directed strategies for paramodulation. In: Book, R.V. (ed.) RTA-91. LNCS, vol. 488, pp. 150–161. Springer (1991)
[59] Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 108–116. Springer (2018)
[60] 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)
[61] Sutcliffe, G.: The CADE-26 automated theorem proving system competition—CASC-26. AI Commun. 30(6), 419–432 (2017)
[62] 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)
[63] 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)
[64] Vukmirović, P., Blanchette, J.C., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. pp. 192–210. LNCS, Springer (2019)
[65] 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`
[66] 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`
[67] Wand, D.: Superposition: Types and Polymorphism. Ph.D. thesis, Universität des Saarlandes (2017)
[68] 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)